National Technical University of Athens
School of Electrical and Computer Engineering

Privacy-Oriented Cryptographic Primitives and Protocols for Electronic Voting

PhD Thesis

Panagiotis M. Grontas

Athens, Greece
December 2020

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Ιδιωτικοστρεφή Κρυπτογραφικά Σχήματα και Πρωτόκολλα για Ηλεκτρονικές Ψηφοφορίες

Διδακτορική Διατριβή

Παναγιώτης Μ. Γροντάς

Αθήνα,
Δεκέμβριος 2020

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Ιδιωτικοστρεφή Κρυπτογραφικά Σχήματα και Πρωτόκολλα για Ηλεκτρονικές Ψηφοφορίες

Παναγιώτης Μ. Γροντάς

Τριμελής Συμβουλευτική Επιτροπή:
Αριστείδης Παγουρτζής *(επιβλέπων)*
Ευστάθιος Ζάχος
Δημήτριος Φωτάκης

Εγκρίθηκε από την επταμελή εξεταστική επιτροπή:

Αριστείδης Παγουρτζής	**Δημήτριος Φωτάκης**	**Παναγιώτης Τσανάκας**
Καθηγητής, ΕΜΠ	Αν. Καθηγητής, ΕΜΠ	Καθηγητής, ΕΜΠ

Αντώνιος Συμβώνης	**Δημήτριος Πουλάκης**
Καθηγητής, ΕΜΠ	Καθηγητής, ΑΠΘ

Άγγελος Κιαγιάς	**Βασίλειος Ζήκας**
Professor, University of Edinburgh	Associate Professor (Sr. Lecturer), University of Edinburgh

Ημερομηνία Εξέτασης: **09/12/2020**

Abstract

Panagiotis M. Grontas

Privacy-Oriented Cryptographic Primitives and Protocols for Electronic Voting

We propose a new cryptographic primitive, *Publicly Auditable Conditional Blind Signatures (PACBS)*, which connects the verification of a digital signature to publicly available data. During signing, a predicate on these data is embedded into the signature, so that the latter is valid if and only if the former is true. Verification is performed by a designated verifier, in a strong manner, with the use of a private verification key. The privacy of the user requesting the signature is protected information-theoretically, because the message to be signed is blinded. Additionally, to avoid attacks from a malicious signer or verifier that disregards the predicate, all their operations are accompanied with evidence in the form of non-interactive zero-knowledge proofs of knowledge that force them to follow the protocol. We define a security model to capture the guarantees of our primitive and provide an instantiation.

We utilize PACBS in a remote electronic voting protocol. The conditional nature of PACBS enables us to build credentials that allow our protocol to provide coercion resistance in the re-voting with anonymous credentials paradigm of Juels, Catalano and Jakobsson. When coerced, a voter uses a fake credential to accompany the vote, while when the coercer is not watching, she can cast her real vote which is accompanied by the valid credential. Only the latter will be counted. All interactions are indistinguishable to the coercer, who cannot tell if his attack succeeded. The evidence generated by PACBS accompanied with standard evidence used in e-voting schemes allows each voter to individually verify that their votes were correctly cast and tallied. Vote counting is also universally verifiable by any interested party. Our overall architecture also provides strong privacy guarantees, since, contrary to the conventional e-voting paradigm, we do not assume that the talliers are trusted for privacy. This allows us, to extend our reasoning about privacy against a computationally unbounded attacker. We generalize our findings to express security models for everlasting privacy that also consider the data available to the adversary.

Περίληψη

Παναγιώτης Μ. Γροντάς

Ιδιωτικοστρεφή Κρυπτογραφικά Σχήματα και Πρωτόκολλα για Ηλεκτρονικές Ψηφοφορίες

Προτείνουμε τις *Δημόσια Ελέγξιμες Υπο-Συνθήκη Τυφλές Υπογραφές (ΔΕΥΤΥ)*, ένα νέο κρυπτογραφικό σχήμα, που συνδέει την επαλήθευση μιας ψηφιακής υπογραφής με δημόσια διαθέσιμα δεδομένα. Κατά τη διάρκεια της υπογραφής, ένα κατηγόρημα που σχετίζεται με αυτά τα δεδομένα ενσωματώνεται στην υπογραφή, έτσι ώστε η τελευταία να ισχύει εάν και μόνο εάν είναι το κατηγόρημα αποτιμάται ως αληθές. Η επαλήθευση πραγματοποιείται από έναν προκαθορισμένο επαληθευτή, με ισχυρό τρόπο, με τη χρήση ενός ιδιωτικού κλειδιού επαλήθευσης. Το απόρρητο του χρήστη που ζητά την υπογραφή προστατεύεται πληροφοριοθεωρητικά, τυφλώνοντας το προς-υπογραφή μήνυμα. Επιπλέον, για να αποφευχθούν επιθέσεις από κάποιον κακόβουλο υπογράφοντα ή επαληθευτή που αγνοούν το κατηγόρημα, η δημιουργία και ο έλεγχος των υπογραφών μας συνοδεύονται από στοιχεία με τη μορφή μη-διαδραστικών αποδείξων μηδενικής γνώσης που αναγκάζουν συμμόρφωση προς το πρωτόκολλο. Ορίζουμε ένα μοντέλο ασφαλείας για να αποτυπώσουμε τις εγγυήσεις των υπογραφών μας και παρέχουμε μια υλοποίηση.

Χρησιμοποιούμε τις ΔΕΥΤΥ σε ένα πρωτόκολλο απομακρυσμένης ηλεκτρονικής ψηφοφορίας. Η υπο-συνθήκη επαλήθευση μας βοηθά να δημιουργήσουμε ανώνυμα διαπιστευτήρια που επιτρέπουν στο σύστημά μας να αντιμετωπίζει επιθέσεις εξαναγκασμού στο υπόδειγμα πολλαπλών ψήφων ανά ψηφοφόρο σε συνδυασμό με ανώνυμα κανάλια και μια στιγμή ιδιωτικότητας. Κατά τον εξαναγκασμό, μία ψηφοφόρος χρησιμοποιεί ψεύτικο διαπιστευτήριο για να συνοδεύσει την επιλογή της, ενώ όταν ο εξαναγκαστής δεν παρακολουθεί, μπορεί να εισάγει την πραγματική ψήφο της που συνοδεύεται από έγκυρο διαπιστευτήριο. Φυσικά, μόνο η ψήφος με το έγκυρο διαπιστευτήριο θα μετρηθεί. Όλες οι αλληλεπιδράσεις είναι μη διακρίσιμες από τον εξαναγκαστή, ο οποίος δεν μπορεί να πει ποια ψήφος μέτρησε. Τα αποδεικτικά στοιχεία που δημιουργούνται από τις ΔΕΥΤΥ, μαζί με τα συνήθη αποδεικτικά στοιχεία που χρησιμοποιούνται σε συστήματα ηλεκτρονικής ψηφοφορίας, επιτρέπουν σε κάθε ψηφοφόρο να επαληθεύσει μεμονωμένα ότι η ψήφος

καταχωρήθηκε σωστά και μετρήθηκε. Η καταμέτρηση των ψήφων, επιπλέον, είναι καθολικά επαληθεύσιμη, από οποιαδήποτε ενδιαφερόμενη οντότητα. Η συνολική αρχιτεκτονική του συστήματος μας, παρέχει επίσης ισχυρότερες εγγυήσεις για την προστασία της μυστικότητας της ψήφου, καθώς, σε αντίθεση με τη συνήθη πρακτική στις ηλεκτρονικές ψηφοφορίες, δεν υποθέτουμε ότι οι καταμετρητές τηρούν το απόρρητο της ψήφου. Αυτό μας επιτρέπει να επεκτείνουμε την μυστικότητα εναντίον ενός υπολογιστικά αδέσμευτου αντιπάλου. Γενικεύουμε τα ευρήματά μας για να εκφράσουμε μοντέλα ασφαλείας για αέναη προστασία μυστικότητας, που λαμβάνουν επίσης υπόψιν τα διαθέσιμα δεδομένα που διαρρέονται από την υλοποίηση του πρωτοκόλλου.

Εκτεταμένη περίληψη

Εισαγωγή Οι ηλεκτρονικές ψηφοφορίες μπορούν να κάνουν καλύτερες τις εκλογές με πολλούς τρόπους: Επιταχύνοντας την καταμέτρηση, βελτιώνοντας την εμπειρία του χρήστη - ειδικά όταν τα ψηφοδέλτια είναι πολύπλοκα ή όταν οι ψηφοφόροι αντιμετωπίζουν φυσικούς περιορισμούς και χρειάζονται υποβοήθηση για να εκφράσουν τις προτιμήσεις τους. Οι απομακρυσμένες ηλεκτρονικές ψηφοφορίες μπορούν επίσης να προσθέσουν αμεσότητα, με αποτέλεσμα μεγαλύτερη συχνότητα και συμμετοχή στη συλλογική λήψη αποφάσεων κάτι που είναι πολύ σημαντικό στην παγκοσμιοποιημένη κοινωνία μας. Τέλος, μπορεί να οδηγήσουν σε νέα υποδείγματα εκλογών αλλάζοντας την δημοκρατία σε μικρή και μεγάλη κλίμακα.

Για να επιτευχθούν όλα αυτά, οι ηλεκτρονικές εκλογές πρέπει να είναι ασφαλείς, *ακόμα περισσότερο από τις φυσικές*, καθώς πρέπει να πείσουν ότι μόνο κέρδος προκύπτει από την χρήση τους και δεν διακινδυνεύονται κεκτημένα. Απέχουμε πολύ από αυτό το στόχο, παρά τις πολλές προσπάθειες και τα διάφορα συστήματα που έχουν αναπτυχθεί. Αυτό οφείλεται κυρίως στο ότι οι εκλογές είναι ένα πολύ δύσκολο πρόβλημα και οι δικλείδες ασφαλείας που εφαρμόζονται στον φυσικό κόσμο είναι αντικείμενο εξέλιξης και πειραματισμού εκατοντάδων ετών, στενά συνδεδεμένες με κοινωνικές δομές που απολαμβάνουν εμπιστοσύνη. Όλη αυτή η εξέλιξη πρέπει να αντικατασταθεί από συστήματα υπολογιστών, τα οποία, εκτός από ότι αριθμούν μερικές μόνο δεκαετίες ύπαρξης, είναι γνωστά για την ρευστότητά τους και το πόσο εύκολα εκτρέπονται ειδικά όταν υλοποιούνται σε λογισμικό. Επιπλέον, οι εκλογές ηλεκτρονικές ή όχι, είναι εξ' ορισμού ένα εχθρικό περιβάλλον, καθώς οι ενδιαφερόμενοι έχουν πολλά κίνητρα να επηρεάσουν το αποτέλεσμα προς όφελός τους. Κατά συνέπεια, δεν αρκεί κάποιο ηλεκτρονικό σύστημα ψηφοφορίας να είναι σωστό. Πρέπει επίσης να είναι επαληθεύσιμο, για να ωθήσει τους ψηφοφόρους (ειδικά τους υποστηρικτές των ηττημένων) να αποδεχθούν ότι η συνεισφορά τους λήφθηκε υπόψη, χωρίς τυχαία ή κακόβουλα λάθη.

Ιδιότητες Οι ιδιότητες που πρέπει να ικανοποιεί ένα σύστημα εκλογών φυσικό ή ηλεκτρονικό είναι οι παρακάτω:

- Ορθότητα (Correctness): Τα αποτελέσματα πρέπει να αντιστοιχούν στις προτιμήσεις των ψηφοφόρων.

- Επαληθευσιμότητα (Verifiability): Οι εκλογείς αλλά και οποιοσδήποτε ενδιαφερόμενος πρέπει να μπορεί να επαληθεύσει την ορθότητα. Γίνεται συνήθως σε 3 στάδια [Cha04; AN06]:
  - Επαλήθευση καταγραφής πρόθεσης (Cast-as-intended): Το εκλογικό σύστημα επιτρέπει στον ψηφοφόρο να ελέγξει ότι καταχωρήθηκε σωστά η πρόθεσή του.
  - Επαλήθευση κατάθεσης (Recorded-as-cast): Μπορεί να ελεγχθεί αν η καταχωρημένη ψήφος μεταφέρθηκε σωστά για καταμέτρηση.
  - Επαλήθευση καταμέτρησης (Tallied-as-recorded): Η καταμέτρηση αντιστοιχεί στις ψήφους που κατατέθηκαν - δεν υπάρχει δηλαδή κάποια αλλαγή στο ενδιάμεσο.

  Στα παραδοσιακά συστήματα ψηφοφορίας η επαληθευσιμότητα είναι κυρίως αρμοδιότητα έμπιστων τρίτων οντοτήτων (μελών της δικαστικής εξουσίας και αντιπροσώπων των υποψηφίων). Ο ψηφοφόρος δεν μπορεί άμεσα να επαληθεύσει την ψήφο του εκτός από το στάδιο καταγραφής πρόθεσης. Σε ένα ηλεκτρονικό περιβάλλον κανένα στάδιο δεν μπορεί να γίνει χωρίς τη βοήθεια υπολογιστικών συστημάτων. Κατά συνέπεια προστίθεται και μία ακόμα μη διαφανής οντότητα.

  Στη διατριβή αυτή δεν ασχολούμαστε με την επαλήθευση της πρόθεσης του ψηφοφόρου, καθώς για τον σκοπό αυτό μπορούν να χρησιμοποιηθούν πολλές από τις γνωστές σχετικές τεχνικές. Χρησιμοποιούμε τον όρο εκλογική επαληθευσιμότητα (election verifiability) [SFC15], η οποία μπορεί να αναλυθεί σε ατομική (individual) και καθολική (universal) επαληθευσιμότητα [Cor+16]. Στην τελευταία πολλοί συμπεριλαμβάνουν την επαληθευσιμότητα δικαιώματος ψήφου (eligibility verifiability) η οποία μπορούν να είναι δημόσια [SFC15] ή ιδιωτική [KTV15]. Επειδή η επαληθευσιμότητα προσπαθεί να προστατεύσει τους ψηφοφόρους από διεφθαρμένες αρχές και συσκευές που κάνουν λάθη - είτε επίτηδες, είτε κατά λάθος - κατά την τυπική μοντελοποίησή της όλες οι αρχές θεωρούνται ως ελεγχόμενες από τον αντίπαλο.

- Μυστικότητα: Βοηθά και αναγκάζει τους ψηφοφόρους να εκφράζουν ελεύθερα τη γνώμη τους. Ως απαίτηση ασφάλειας μάλιστα είναι κωδικοποιημένη στη νομοθεσία. Στις φυσικές ψηφοφορίες εφαρμόζεται με τον έλεγχο του περιβάλλοντος ψηφοφορίας με φυσικά μέσα (παραβάν, κάλπη) και έμπιστες τρίτες οντότητες που περιορίζουν τον ψηφοφόρο με τέτοιο τρόπο ώστε να διατηρηθεί το απόρρητο της ψηφοφορίας. Τα συγκεκριμένα αντίμετρα δεν ισχύουν σε απομακρυσμένο (ηλεκτρονικό) περιβάλλον, γεγονός που καθιστά την πώληση ψήφων και την εξαναγκασμένη ψηφοφορία πιθανά και επικίνδυνα ενδεχόμενα.

  Η μυστικότητα ορίζεται και αυτή σε διάφορα επίπεδα:

- μυστικότητα ψήφου (ballot secrecy / privacy) [Cor+14]: προστατεύει ενάντια σε έναν παθητικό αντίπαλο ο οποίος θέλει να μάθει τα περιεχόμενα της ψήφου ενός ψηφοφόρου. Τέτοιοι για παράδειγμα είναι οι καταμετρητές ή άλλοι ψηφοφόροι. Σε αυτό το επίπεδο, η μυστικότητα είναι εφήμερη με διάρκεια όση και οι εκλογές.
- μη-αποδειξιμότητα ψήφου (receipt-freeness) [BT94]: προστατεύει ενάντια σε έναν κακόβουλο ψηφοφόρο που θέλει να πουλήσει την ψήφο του.
- αντίσταση στον εξαναγκασμό (coercion-resistance) [JCJ05]: προστατεύει ενάντια σε έναν εξωτερικό ισχυρό αντίπαλο που υποδεικνύει στον ψηφοφόρο την επιλογή του ή τον προσομοιώνει ή τον αναγκάζει να απόσχει. Ο αντίπαλος αυτός μπορεί να βρίσκεται 'δίπλα' στον ψηφοφόρο καθώς ο τελευταίος ψηφίζει ή να παίζει το ρόλο του έχοντας υποκλέψει τα επίσημα διαπιστευτήρια. Οι δύο παραπάνω ιδιότητες ικανοποιούνται διαισθητικά με την παρακάτω προσέγγιση: ο αντίπαλος δεν έχει έχει κίνητρο να πραγματοποιήσει την επίθεσή του αν δεν μπορεί να είναι σίγουρος ότι θα επιτύχει. Έτσι ο ψηφοφόρος αποκτά διάφορα μέσα στη διάθεσή του ώστε να μπορεί να του δημιουργήσει αμφιβολία.
- αέναη ιδιωτικότητα (everlasting privacy) [MN06]: προστατεύει απέναντι σε έναν υπολογιστικά ισχυρό αντίπαλο ο οποίος αντιπροσωπεύει επιθέσεις που λαμβάνουν χώρα όταν οι διάφορες υπολογιστικές υποθέσεις που προστατεύουν τα διάφορα κρυπτοσυστήματα δεν θα ισχύουν πλέον.

Από την παραπάνω περιγραφή προκύπτει ότι οι θεμελιώδεις ιδιότητες ασφάλειας των εκλογών είναι εν γένει αντικρουόμενες. Η επαληθευσιμότητα χωρίς μυστικότητα είναι εύκολη και η πανάρχαια μέθοδος της ψηφοφορίας δι' ανατάσεως χειρός την υλοποιεί. Η μυστικότητα χωρίς επαληθευσιμότητα δεν έχει νόημα γιατί οι ψηφοφόροι δεν έχουν κίνητρο να εκφράσουν τη γνώμη τους αν ξέρουν ότι δεν θα καταμετρηθεί σίγουρα. Η μη αποδειξιμότητα ψήφου και η αντίσταση στον εξαναγκασμό έρχονται σε αντίθεση με την επαληθευσιμότητα. Στα διάφορα συστήματα που έχουν προταθεί, όπως το Helios [Adi08], προτιμάται η ισχυρή επαληθευσιμότητα παρά η ιδιωτικότητα. Κατά συνέπεια απαιτείται εμπιστοσύνη στις αρχές ότι θα εφαρμόσουν το πρωτόκολλο στην μυστικότητα. Αποτελεί όμως ανοικτό ερώτημα αν κάτι τέτοιο είναι αποδεκτό από τους ψηφοφόρους.

Άλλες ιδιότητες που πρέπει να ικανοποιούν τα διάφορα συστήματα εκλογών είναι η δικαιοσύνη (fairness), η ανθεκτικότητα (resiliency), η αποδοτικότητα (efficiency) και η ενθάρρυνση (enfranchisement).

Κρυπτογραφικά σχήματα για εκλογές Η βασική έννοια που έρχεται να αντιμετωπίσει τα προβλήματα που εισάγει η τεχνολογία στις ηλεκτρονικές ψηφοφορίες είναι η ανεξαρτησία από το λογισμικό (software independence) [Riv08], η οποία ορίζει ότι οποιαδήποτε μη ανιχνεύσιμη αλλαγή ή λάθος στο σύστημα εκλογών δεν οδηγεί σε μη ανιχνεύσιμη αλλαγή ή λάθος στο αποτέλεσμα των εκλογών. Ένας τρόπος επίτευξης της ιδιότητας της ανεξαρτησίας από λογισμικό είναι η ύπαρξη ελέγξιμων στοιχείων σε κάποιο φυσικό μέσο όπως χαρτί.

Η διατριβή αυτή ασχολείται με την κρυπτογραφική προσέγγιση στην διασφάλιση της ιδιότητας της ανεξαρτησίας από το λογισμικό καθώς και των υπόλοιπων ιδιοτήτων δίνοντας ιδιαίτερη έμφαση στην προστασία από τον εξαναγκασμό και την αέναη ιδιωτικότητα. Ειδικότερα η κρυπτογραφική προσέγγιση φτιάχνει τυπικά μοντέλα ασφάλειας για τις ιδιότητες των εκλογικών συστημάτων. Στην συνέχεια προτείνει πρωτόκολλα και τεχνικές που υλοποιούν τέτοια συστήματα και αποδεικνύει με αυστηρό τρόπο τις ιδιότητες ασφάλειας που ισχυρίζεται. Απώτερος στόχος της κρυπτογραφικής προσέγγισης είναι να μειωθεί ή να εξαλειφθεί εντελώς η εμπιστοσύνη που πρέπει να εναποτίθεται σε εξωτερικές αρχές για τη διεξαγωγή των εκλογών κάτι που είναι αδύνατο στα παραδοσιακά συστήματα ψηφοφορίας. Ταυτόχρονα όμως η μοντελοποίηση συμβάλλει στην καλύτερη κατανόηση των διαφόρων ιδιοτήτων ασφαλείας και αποκαλύπτει πολλές φορές απρόσμενες σχέσεις μεταξύ τους.

Στη διατριβή βασιζόμαστε στα παρακάτω κρυπτογραφικά σχήματα και πρωτόκολλα:

- Ομομορφικά κρυπτοσυστήματα δημοσίου κλειδιού: Επιτρέπουν την προστασία της μυστικότητας της ψήφου κρυπτογραφώντας την ψήφο με το δημόσιο κλειδί της εκλογικής αρχής. Επιπλέον επιτρέπουν την διενέργεια πράξεων μεταξύ κρυπτοκειμένων οι οποίες μεταφέρονται στα αρχικά μηνύματα. Στις εκλογές χρησιμοποιούνται κρυπτοσυστήματα τα οποία πολλαπλασιάζοντας δύο κρυπτοκείμενα αθροίζουν τα περιεχόμενα μηνύματα. Έτσι μπορεί να υπολογιστεί το αποτέλεσμα των εκλογών. Ένα τέτοιο κρυπτοσύστημα είναι το εκθετικό ElGamal [Gam85; CGS97].
- Τυφλές υπογραφές (blind signatures) [Cha83]: Είναι ψηφιακές υπογραφές, όπου η οντότητα που υπογράφει δεν έχει πρόσβαση στο μήνυμα και δεν μπορεί να συσχετίσει μηνύματα και συνόδους υπογραφής με τις τελικές υπογραφές. Σε ό,τι αφορά τη μη πλαστογράφηση, το μοντέλο ασφαλείας τους ορίζει ότι όποιος δεν έχει το ιδιωτικό κλειδί δεν μπορεί να παράγει μία επιπλέον υπογραφή από όσες έχει αιτηθεί ο χρήστης (μία επιπλέον πλαστογράφηση) [PS00]. Στις εκλογές ο υπογράφων είναι η εκλογική αρχή ενώ ο χρήστης είναι ο ψηφοφόρος.

Χρησιμοποιούνται από την εκλογική αρχή ώστε να επιτρέψουν τον έλεγχο του δικαιώματος ψήφου του ψηφοφόρου, χωρίς την αποκάλυψη της προτίμησής του. Στη διατριβή χρησιμοποιούμε ιδιαίτερα τις τυφλές υπογραφές Okamoto – Schnorr [Oka92].

– Αποδείξεις μηδενικής γνώσης (zero knowledge proofs) [GMR85]: Επιτρέπουν την απόδειξη γνώσης ενός 'μάρτυρα', δηλαδή μιας τιμής που κάνει μια σχέση να ισχύει, χωρίς την αποκάλυψη καμίας περαιτέρω πληροφορίας γι' αυτήν. Στην διατριβή αλλά και στα περισσότερα πρωτόκολλα ηλεκτρονικών ψηφοφοριών χρησιμοποιείται μια παραλλαγή τους: τα Σ-πρωτόκολλα - συστήματα αποδείξεων 3-γύρων με τίμιο επαληθευτή, τα οποία μπορούν να γίνουν μη διαδραστικά χρησιμοποιώντας την τεχνική Fiat – Shamir [FS86]. Στις εκλογές χρησιμοποιούνται ώστε να αποδειχθεί η ορθότητα της ψήφου χωρίς να θυσιαστεί η μυστικότητα αλλά και για να αποδειχθεί ο ορθός υπολογισμός του αποτελέσματος. Βασιζόμαστε εκτεταμένα στα πρωτόκολλα των Schnorr [Sch91] και Chaum – Pedersen [CP93].

– (Ισχυρές) υπογραφές καθορισμένου επαληθευτή ((strong) designated verifier signatures) [JSI96]: Σε αυτές εκτός από τον υπογράφων, και ο επαληθευτής διαθέτει ένα ζεύγος κλειδιών. Στην απλή έκδοσή τους, ο επαληθευτής χρησιμοποιεί το ιδιωτικό κλειδί του μόνο για να τις προσομοιώσει. Στην ισχυρή έκδοσή τους, δεν είναι δημόσια επαληθεύσιμες, όπως οι κανονικές ψηφιακές υπογραφές αλλά ο επαληθευτής πρέπει να χρησιμοποιήσει ένα ιδιωτικό κλειδί ελέγξει. Στα διάφορα κρυπτογραφικά συστήματα εκλογών έχουν στόχο κυρίως να παρέχουν προστασία από εξαναγκασμό. Οι καθορισμένοι επαληθευτές είναι οι ψηφοφόροι. Οι διάφορες αρχές (κυρίως εγγραφής) παρέχουν μια τέτοια υπογραφή ως απόδειξη για τη σωστή δημιουργία ενός διαπιστευτηρίου. Όταν τη ζητήσει ο εξαναγκαστής, τότε ο ψηφοφόρος δίνει μία προσομοίωσή της που δεν μπορεί να διαχωριστεί από την πραγματική. Η δική μας προσέγγιση είναι η αντίστροφη: Ο προκαθορισμένος επαληθευτής είναι ο καταμετρητής. Ο ψηφοφόρος δημιουργεί μια τέτοια υπογραφή, εγκαθιστώντας έτσι ένα ιδιωτικό κανάλι για να μεταφέρει την πληροφορία αν η ψήφος πρέπει να καταμετρηθεί ή όχι, ή ισοδύναμα αν αποτελεί προϊόν ελεύθερης επιλογής ή εξαναγκασμού. Η ιδιωτική επαλήθευση, έχει ως αποτέλεσμα, μόνο ο καταμετρητής να γνωρίζει αν η ψήφος πρέπει να καταμετρηθεί ή όχι. Για να είναι αυτό το γεγονός καθολικά επαληθεύσιμο όμως, πρέπει ο καθορισμένος επαληθευτής να παρέχει στοιχεία σε μορφή αποδείξεων μηδενικής γνώσης τα οποία θα αποδεικνύουν ότι ακολουθήθηκε το πρωτόκολλο.

– Έλεγχος ισοδυναμίας μηνυμάτων (Plaintext Equivalence Test) [JJ00; MPT20]: Ένα πρωτόκολλο το οποίο επιτρέπει σε ένα σύνολο οντοτήτων να ελέγξουν αν

δύο κρυπτοκείμενα περιέχουν το ίδιο μήνυμα. Οι οντότητες τυφλώνουν αρχικά το ομομορφικό πηλίκο των κρυπτοκειμένων. Στη συνέχεια τα αποκρυπτογραφούν μερικώς και συνδυάζουν τις κρυπτογραφήσεις τους. Τέλος, ελέγχουν αν το αποτέλεσμα είναι 1 που σημαίνει ότι τα κρυπτοκείμενα περιέχουν το ίδιο αρχικό μήνυμα. Σε διαφορετική περίπτωση το αποτέλεσμα θα είναι ένα τυχαίο στοιχείο της ομάδας.

Στην διατριβή, προτείνουμε ένα κρυπτογραφικό εργαλείο το οποίο συνδυάζει έναν έλεγχο ισοδυναμίας μηνυμάτων με τυφλές υπογραφές καθορισμένου επαληθευτή.

– Δίκτυα μίξης (mixnets) [Cha82]: Επιτρέπουν την ανωνυμοποίηση ενός συνόλου μηνυμάτων με τρόπο επαληθεύσιμο. Αποτελούνται από ένα σύνολο εξυπηρετητών οι οποίοι αλλάζουν τη μορφή και εφαρμόζουν μια τυχαία μετάθεση στις εισόδους τους, που συνήθως είναι κρυπτοκείμενα που έχουν παραχθεί με ένα ομομορφικό κρυπτοσύστημα. Για να αποφευχθεί η εξαπάτηση κάθε εξυπηρετητής παρέχει μία απόδειξη μηδενικής γνώσης ότι ακολούθησε το πρωτόκολλο.

Κρυπτογραφικά πρωτόκολλα ψηφοφορίας Σε ένα πρωτόκολλο ηλεκτρονικής ψηφοφορίας συμμετέχουν οι παρακάτω οντότητες:

– Ψηφοφόροι. Συνήθως χρησιμοποιούν ειδικό λογισμικό / υλικό για να καταθέσουν την ψήφο τους.

– Αρχές εγγραφής (registration authorities): Μοιράζουν απλά ή σύνθετα διαπιστευτήρια στους ψηφοφόρους και ελέγχουν αν έχουν δικαίωμα ψήφου.

– Αρχές καταμέτρησης (tallying authorities): Καταμετρούν τις ψήφους και εξάγουν το τελικό αποτέλεσμα.

– Αποθετήριο ψήφων (bulletin board): Ένα κανάλι εκπομπής με μνήμη, που λειτουργεί ως αποθετήριο ψήφων.

Η βασική συνεισφορά της διατριβής, είναι ένα πρωτόκολλο ψηφοφορίας που αντλεί έμπνευση από δύο γνωστά σχήματα της βιβλιογραφίας: το πρωτόκολλο των Fujioka, Okamoto και Ohta (FOO) [FOO92; Ohk+99] που προσφέρει ισχυρές εγγυήσεις ιδιωτικότητας και το σχήμα των Juels, Catalano και Jakobsson (JCJ) [JCJ05] που προσφέρει αντίσταση στον εξαναγκασμό. Για καλύτερη κατανόηση της συνεισφοράς μας θα τα περιγράψουμε περιληπτικά στη συνέχεια.

Το σχήμα FOO, χρησιμοποιεί τυφλές υπογραφές για να εγκρίνει τα ψηφοδέλτια και παρέχει προστασία της μυστικότητας, χωρίς να είναι αναγκαία η εμπιστοσύνη

στους καταμετρητές. Επιπλέον όταν η κατάθεση των ψήφων γίνεται χρησιμοποιώντας ανώνυμο κανάλι παρέχει και αένα ιδιωτικότητα. Ωστόσο, έχει πρόβλημα με εισαγωγή ψεύτικων ψήφων και δεν παρέχει καθολική επαληθευσιμότητα. Η ροή του πρωτοκόλλου, όπως έχει διαμορφωθεί μετά από αρκετές παραλλαγές έχει ως εξής:

- Κάθε ψηφοφόρος επιλέγει την ψήφο του.
- Κρυπτογραφεί την ψήφο χρησιμοποιώντας το δημόσιο κλειδί της αρχής καταμέτρησης. Έτσι σχηματίζεται το ψηφοδέλτιο.
- Το ψηφοδέλτιο τυφλώνεται και αποστέλλεται στην αρχή εγγραφής.
- Γίνεται έλεγχος αν ο ψηφοφόρος έχει δικαίωμα ψήφου ή όχι. Στην θετική περίπτωση υπογράφεται το τυφλωμένο ψηφοδέλτιο και επιστρέφει στον ψηφοφόρο. Η φάση αυτή του πρωτοκόλλου ονομάζεται φάση εξουσιοδότησης.
- Ο ψηφοφόρος αποτυφλώνει την υπογραφή και έτσι διαθέτει πλέον ένα ψηφοδέλτιο και μια υπογραφή σε αυτό, την οποία καταθέτει στην αρχή καταμέτρησης.
- Μόνο οι ψήφοι με έγκυρη υπογραφή καταμετρώνται στο αποτέλεσμα.

Στο σχήμα FOO, η χρήση των τυφλών υπογραφών παρέχει τέλεια μυστικότητα ως προς την αρχή εγγραφής. Επιπλέον ο συνδυασμός του με ένα ανώνυμο κανάλι μπορεί να παρέχει μυστικότητα χωρίς να υπάρχει ανάγκη οι καταμετρητές να θεωρούνται έμπιστες τρίτες οντότητες. Διαισθητικά αυτό οφείλεται στο ότι οι διάφορες αρχές δεν μπορούν να συσχετίσουν ψήφους σε ψηφοφόρους, οπότε η αποκρυπτογράφηση για την καταμέτρηση δεν τους παρέχει καμία επιπλέον πληροφορία.

Το σχήμα JCJ [JCJ05] παρέχει έναν ορισμό και τρόπους προστασίας για τις επιθέσεις εξαναγκασμού. Συγκεκριμένα, η αντιμετώπιση του εξαναγκασμού περιλαμβάνει προστασία έναντι της πώλησης ψήφων και αντίμετρα έναντι των επιθέσεων προσομοίωσης, τυχαίας ψήφου και αναγκαστικής αποχής. Ως μέσο αντιμετώπισης χρησιμοποιούνται πολλαπλές αντί για μοναδική ψήφο που συνοδεύονται με ανώνυμα διαπιστευτήρια. Η βασική ιδέα του σχήματος JCJ είναι η δημιουργία αμφιβολιών στον εξαναγκαστή για το αν η επίθεσή του πέτυχε με στόχο να του αφαιρεθεί το κίνητρο για την πραγματοποίησή της. Κάθε ψηφοφόρος έχει ένα έγκυρο διαπιστευτήριο το οποίο δημιουργεί σε συνεργασία με την αρχή εγγραφής. Αποκτά δυνατότητα να δημιουργεί νέα διαπιστευτήρια με χρήση ειδικού υλικού ή λογισμικού ή και χωρίς [UH12]. Έτσι, όταν δέχεται επίθεση, ο ψηφοφόρος μπορεί να ακολουθήσει τις οδηγίες του εξαναγκαστή χρησιμοποιώντας ένα ψεύτικο διαπιστευτήριο, το οποίο όμως δεν μπορεί να διαχωριστεί από το πραγματικό. Φαινομενικά, δηλαδή, υποκύπτει στην επίθεση του εξαναγκαστή. Σε μια *ιδιωτική στιγμή* όμως, η οποία είναι απαραίτητη προϋπόθεση για την επίτευξη της προστασίας από εξαναγκασμό,

μπορεί να επιλέξει τον υποψήφιο που πραγματικά επιθυμεί και να καταθέσει την πραγματική ψήφο με τα κανονικά της διαπιστευτήρια. Τα ψεύτικα και τα κανονικά διαπιστευτήρια είναι μη διακρίσιμα. Η μόνη διαφορά τους είναι ότι τα τελευταία έχουν δηλωθεί στην αρχή εγγραφής, ενώ τα πρώτα παράγονται από μια συσκευή που φέρει κάθε ψηφοφόρος. Κατά συνέπεια, ο αντίπαλος δεν είναι σε θέση να ξεχωρίσει εάν η επίθεσή του πέτυχε και ως εκ τούτου δεν θα έχει κανένα κίνητρο να την πραγματοποιήσει. Ακόμα με δεδομένα ότι ο ψηφοφόρος δεν έχει τρόπο να πείσει εάν τα διαπιστευτήρια είναι αληθινά, τότε δεν έχει αξία να πουλήσει και την ψήφο του. Κατά την καταμέτρηση η αρμόδια αρχή ελέγχει όλα τα διαπιστευτήρια που συνοδεύουν τις ψήφους με τα αρχικά δηλωμένα χρησιμοποιώντας τον έλεγχο ισοδυναμίας μηνυμάτων (PET) και καταμετρά μόνο αυτές στις οποίες ο έλεγχος είναι επιτυχής. Για να παρέχει προστασία από τον εξαναγκασμό το μοντέλο των JCJ υιοθετεί τις παρακάτω υποθέσεις:

- Ο εξαναγκαστής δεν παρακολουθεί τον ψηφοφόρο σε όλη τη διάρκεια της διαδικασίας. Δηλαδή ο ψηφοφόρος έχει μια στιγμή ιδιωτικότητας, οπότε και μπορεί να καταθέσει την κανονική του ψήφο.
- Η εγγραφή των ψηφοφόρων στους καταλόγους και η δημιουργία των διαπιστευτηρίων γίνεται μέσα από ένα κανάλι το οποίο δεν μπορεί να παρακολουθεί ο αντίπαλος. Ένα τέτοιο κανάλι είναι με φυσική παρουσία. Αν και αυτό φαίνεται ασύμβατο με τις ηλεκτρονικές ψηφοφορίες, δεν αποτελεί σημαντικό πρόβλημα καθώς μπορεί να γίνει μία φορά και τα διαπιστευτήρια να χρησιμοποιηθούν σε πολλές εκλογές.
- Επιπλέον για την αρχή εγγραφής ισχύουν οι εξής ειδικότερες υποθέσεις:
  - Τα μηνύματα που ανταλλάσσονται από το πρωτόκολλο διαγράφονται εξ' ολοκλήρου ή
  - Η αρχή εγγραφής είναι έμπιστη ή
  - ο ψηφοφόρος γνωρίζει ποια μέλη της αρχής συνεργάζονται με τον αντίπαλο και κατά συνέπεια μπορεί να χρησιμοποιήσει ένα από τα έντιμα ώστε να μπορεί να ξεγελάσει τον αντίπαλο με τα δεδομένα που λαμβάνει από αυτόν.
- Η κατάθεση της ψήφου γίνεται από ένα ανώνυμο κανάλι, στο οποίο δεν φαίνεται η ταυτότητα του ψηφοφόρου. Αυτό είναι αναγκαία συνθήκη για να αντιμετωπιστεί η επίθεση όπου ο αντίπαλος αναγκάζει τον ψηφοφόρο να απέχει από τις εκλογές.

- Η αρχή καταμέτρησης είναι κατά πλειοψηφία έντιμη. Όπως επισημαίνεται στο [MPT20] αντίστοιχα αποτελέσματα μπορεί να εξαχθούν ακόμα και αν όλα τα μέλη της αρχής καταμέτρησης είναι ελεγχόμενα από τον αντίπαλο.
- Ο αντίπαλος έχει μια αβεβαιότητα για το πώς ή για το εάν θα ψηφίσουν οι ψηφοφόροι που συνεργάζονται μαζί του. Σε διαφορετική περίπτωση θα μπορούσε να μάθε αν πέτυχε η επίθεσή του αφαιρώντας τις γνωστές σε αυτόν ψήφους από το αποτέλεσμα των εκλογών.

Το κύριο πρόβλημα με το σχήμα JCJ είναι ότι καθιστά την καταμέτρηση μη αποδοτική, καθώς απαιτεί τετραγωνικό πλήθος συγκρίσεων διαπιστευτηρίων, ώστε να ξεχωρίσει τα πραγματικά από τα ψεύτικα. Αυτό καθιστά το JCJ εφαρμόσιμο μόνο σε ψηφοφορίες που συμμετέχουν λίγοι ψηφοφόροι. Έχουν υπάρξει αρκετές προσπάθειες επιτάχυνσης του JCJ [Smi05; AFT07; Ara+10; AT13; KHF11] και δύο αξιόλογες υλοποιήσεις το σύστημα CIVITAS [CCM08] και το σύστημα Selections [UH12]. Στην τελευταία μάλιστα δίνεται και ένας πολύ φιλικός μηχανισμός δημιουργίας διαπιστευτηρίων, που δεν απαιτεί από τον χρήστη να διαθέτει λογισμικό το οποίο εκτελεί κρυπτογραφικές λειτουργίες για να τα δημιουργήσει και να τα χειριστεί, αλλά μπορεί να λειτουργήσει χρησιμοποιώντας ένα ειδικό σύστημα συνθηματικών το οποίο να αντιστοιχίζεται σε κρυπτογραφικά διαπιστευτήρια. Συγκεκριμένα η αρχή εγγραφής δηλώνει ένα σύνολο από πιθανά συνθηματικά (πχ. όλοι οι πιθανοί συνδυασμοί 5 λέξεων από ένα λεξικό [CH08]). Κατά την εγγραφή ο ψηφοφόρος δηλώνει ποιο θα είναι το έγκυρο συνθηματικό του. Όλα τα υπόλοιπα, αν χρησιμοποιηθούν υποδηλώνουν ότι είναι υπό εκβιασμό και δεν πρέπει να μετρήσει η ψήφος που τα συνοδεύει. Η διεπαφή χρήστη δεν αντιδρά διαφορετικά στις δύο αυτές περιπτώσεις. Αν δοθεί κάτι άλλο, θεωρείται ότι το λάθος προέκυψε κατά την πληκτρολόγηση και το συνθηματικό ξαναζητείται.

CBS και PACBS Η κύρια ιδέα που αναπτύσσεται στη διατριβή προέρχεται από την παρατήρηση ότι εάν ο έλεγχος για τον προσδιορισμό της εγκυρότητας ψήφου του JCJ μπορούσε να μετακινηθεί στη φάση εξουσιοδότησης, όπως αυτή που εκτελείται στο πρωτόκολλο FOO, τότε θα μπορούσαμε να χρησιμοποιήσουμε τις πληροφορίες ταυτότητας των ψηφοφόρων για να ομαδοποιήσουμε τα διαπιστευτήρια ανά ταυτότητα και να μειώσουμε τον τετραγωνικό αριθμό συγκρίσεων σε γραμμικό [GPZ17]. Επομένως, η εκλογική αρχή εγγραφής θα γνωρίζει εάν πρέπει να μετρηθεί μια ψήφος. Αυτό το γεγονός πρέπει να κοινοποιηθεί στην αρχή καταμέτρησης χωρίς να το καταλάβει ο εξαναγκαστής. Επιπλέον, επειδή καμία αρχή δεν πρέπει να είναι έμπιστη, αυτό πρέπει να γίνεται με επαληθεύσιμο τρόπο. Με αυτή την αρχιτεκτονική μπορούμε επιπλέον να εκμεταλλευτούμε τις ισχυρές εγγυήσεις ιδιωτικότητας

που παρέχει το FOO και να πετύχουμε μυστικότητα χωρίς να χρειάζεται να εμπιστευόμαστε τις διάφορες αρχές για την συγκεκριμένη ιδιότητα. Η πρότασή μας είναι η μεταβίβαση της πληροφορίας για το αν θα πρέπει να μετρηθεί η ψήφος να γίνει μέσω της υπογραφής της αρχής εγγραφής. Αυτή η υπογραφή πρέπει να είναι υπό συνθήκη ώστε να είναι έγκυρη αν και μόνο αν έχει χρησιμοποιηθεί το σωστό διαπιστευτήριο και κατά συνέπεια μόνο τότε να μετρηθεί. Επίσης πρέπει να είναι (ισχυρή) υπογραφή καθορισμένου επαληθευτή, ώστε το αποτέλεσμα να γίνει γνωστό μόνο στην αρχή καταμέτρησης.

Εφαρμόζουμε αυτή την ιδέα, σε μια πρώτη απλοποιημένη μορφή, ορίζοντας τις Υπό-Συνθήκη Τυφλές Υπογραφές (YTY) – Conditional Blind Signatures (CBS) [ZGP17]. Σε αυτές τις υπογραφές εκτός από τον υπογράφοντα συμμετέχει και ένας προκαθορισμένος επαληθευτής. Κάθε ένας διαθέτει ένα ιδιωτικό κλειδί και ένα δημόσιο κλειδί. Ο υπογράφων έχει επιπλέον μια ιδιωτική είσοδο b. Αν $b = 1$ τότε ο υπογράφων παρέχει έγκυρη υπογραφή, ενώ αν $b = 0$ ο υπογράφων παράγει μια άκυρη υπογραφή - ή ισοδύναμα στην περίπτωση της ψηφοφορίας αν $b = 1$ πρέπει να μετρήσει η ψήφος που συνοδεύει η υπογραφή, ενώ αν $b = 0$ δεν πρέπει να μετρήσει.

Οι CBS πρέπει να ικανοποιούν τις εξής τρεις ιδιότητες, για τις οποίες ορίζονται ακριβή μοντέλα ασφάλειας μέσω κρυπτογραφικών παιγνίων μεταξύ του συστήματος $\mathcal{C}$ και του αντιπάλου $\mathcal{A}$ στη διατριβή:

- Τυφλότητα: Ο υπογράφων δεν μπορεί να συσχετίσει μηνύματα με υπογραφές. Στο αντίστοιχο παίγνιο, ο $\mathcal{A}$ παίζει το ρόλο του υπογράφοντα και επιλέγει αρχικά τις παραμέτρους του συστήματος συστήματος CBS καθώς και δύο μηνύματα m_0, m_1. Ο $\mathcal{C}$ παίζει το ρόλο του χρήστη και διαλέγει ένα τυχαίο bit $\mathfrak{b}$ που υποδηλώνει τη σειρά υπογραφής. Αρχικά υπογράφεται το $m_{\mathfrak{b}}$ και μετά το $m_{1-\mathfrak{b}}$. Ο αντίπαλος κερδίζει εάν μπορεί να μαντέψει τη σειρά υπογραφής με μη-αμελητέα πιθανότητα.
- Μη-πλαστογραφησιμότητα: Όποιος δεν διαθέτει το ιδιωτικό κλειδί υπογραφής δεν μπορεί να δημιουργήσει παραπάνω υπογραφές από όσες αιτήσεις δέχτηκε. Συγκεκριμένα, ο αντίπαλος, που εδώ είναι ο πλαστογράφος, εκτελεί l συνόδους υπογραφής με τον υπογράφοντα (το πολύ). Στόχος του είναι να προσπαθήσει να δημιουργήσει $l + 1$ έγκυρες υπογραφές έχοντας στη διάθεσή του όλα στοιχεία που προέκυψαν από τις παραπάνω αλληλεπιδράσεις. Αν ο $\mathcal{A}$ δεν μπορεί να νικήσει στο παραπάνω παίγνιο με l πολυλογαριθμικό ως προς την παράμετρο ασφάλειας, τότε το σχήμα υπογραφών παρέχει προστασία από την επίθεση μίας επιπλέον πλαστογράφησης (strong one more forgery) [PS00].
- Υπό-συνθήκη επαληθευσιμότητα: Όποιος δεν διαθέτει το ιδιωτικό κλειδί επαλήθευσης δεν μπορεί να διαπιστώσει αν η υπογραφή είναι έγκυρη ή όχι. Για

τον τυπικό ορισμό της ιδιότητας αυτής ορίζουμε ένα παίγνιο ασφάλειας παρόμοιο με την ιδιότητα ασφάλειας IND-CPA των συστημάτων κρυπτογραφίας δημοσίου κλειδιού. Ο αντίπαλος $\mathcal{A}$ έχει στη διάθεσή του κάποιες υπογραφές τις οποίες χρησιμοποιεί για να κρίνει αν μία υπογραφή σε κάποιο μήνυμα της επιλογής του είναι έγκυρη. Αν ο αντίπαλος δεν καταφέρει να μαντέψει την εγκυρότητα της υπογραφής (δηλαδή το bit b) με μη αμελητέα πιθανότητα.

Στη συνέχεια προτείνουμε μια κατασκευή για το CBS, η οποία βασίζεται στις τυφλές υπογραφές Okamoto-Schnorr [Oka92]. Η κατασκευή λαμβάνει χώρα σε 5 φάσεις και λειτουργεί σε μια ομάδα $\mathbb{G}$ τάξης q, όπου ισχύει η υπόθεση DDH με γεννήτορες g_1, g_2. Συμμετέχουν τρεις οντότητες:

- Ο υπογράφων που έχει στη διάθεσή του το μυστικό κλειδί που αποτελείται από τα $(s_1, s_2) \in \mathbb{Z}_q^2$ και ένα ιδιωτικό bit b. Το δημόσιο κλειδί είναι το $v = g_1^{-s_1} g_2^{-s_2}$
- Ο χρήστης έχει ως ιδιωτική είσοδο το μήνυμα m το οποίο θέλει να υπογραφεί, χωρίς όμως να το 'δει' ο υπογράφων.
- Ο επαληθευτής που διαθέτει το μυστικό κλειδί s και το αντίστοιχο δημόσιο $k = g_1^s$

Η δημιουργία και η επαλήθευση της υπογραφής λαμβάνουν χώρα ως εξής:

- Δέσμευση: Ο υπογράφων επιλέγει $r_1, r_2 \leftarrow_\$ \mathbb{Z}_q$ και υπολογίζει το $x := g_1^{r_1} g_2^{r_2}$ το οποίο και αποστέλλει στον χρήστη.
- Τύφλωση: Ο χρήστης επιλέγει $u_1, u_2, d \leftarrow_\$ \mathbb{Z}_q$ και υπολογίζει τα
  - $x^* := x g_1^{u_1} g_2^{u_2} v^d$
  - $e^* := \mathsf{H}(m, x^*)$
  - $e := e^* - d$ το οποίο και αποστέλλεται στον υπογράφοντα.
- Υπογραφή: Υπολογίζονται οι τιμές $y_1 := r_1 + e s_1, y_2 := r_2 + e s_2$. Αν $b = 1$ τότε δημιουργούνται οι τιμές $(\beta_1, \beta_2) := (k^{y_1}, y_2)$ αλλιώς επιλέγονται $(\beta_1, \beta_2) \leftarrow_\$ \mathbb{G} \times \mathbb{Z}_q$. Η τυφλή υπογραφή είναι $\overline{\beta} := (x, e, \beta_1, \beta_2)$.
- Αποτύφλωση: Ο χρήστης υπολογίζει $\sigma_1 := \beta_1 \cdot k^{u_1}, \sigma_2 := \beta_2 + u_2$ και εξάγει την υπογραφή $\overline{\sigma} := (x^*, e^*, \sigma_1, \sigma_2)$
- Επαλήθευση: Ελέγχεται αν ισχύει η σχέση: $x^{*s} = \sigma_1 \cdot {g_2}^{\sigma_2 \cdot s} \cdot v^{e^* \cdot s}$

Με βάση αυτή την κατασκευή αποδεικνύουμε τα παρακάτω:

- Οι CBS παρέχουν τέλεια τυφλότητα. Συγκεκριμένα αποδεικνύεται ότι και για τις δύο όψεις του πρωτοκόλλου που μπορεί να έχει ο $\mathcal{A}$ μπορεί να βρει u_1, u_2, d

ώστε να μπορεί να δημιουργήσει οποιαδήποτε από τις δύο υπογραφές. Κατά συνέπεια δεν μπορεί να συσχετίσει υπογραφή με σύνοδο.

– Οι CBS παρέχουν προστασία ενάντια στην επίθεση μίας επιπλέον πλαστογράφησης με δεδομένο ότι ισχύει η υπόθεση CDH. Συγκεκριμένα αποδεικνύουμε ότι αν ο αντίπαλος καταφέρει να κερδίσει το παίγνιο μη πλαστογραφισιμότητας, τότε μπορεί να υπολογίσει το g_2^s από τα $g_1, g_2, k = g_1^s$. Η απόδειξή μας βασίζεται στην τεχνική επίθεσης με επανάληψη του μαντείου (oracle replay attack) των [PS00].

– Οι CBS παρέχουν υπό συνθήκη επαληθευσιμότητα με δεδομένο ότι ισχύει η υπόθεση DDH. Αποδεικνύουμε, ότι αν ο αντίπαλος μπορεί να διακρίνει το αν μια υπογραφή είναι έγκυρη ή όχι, τότε μπορεί να ελέγξει αν σε μια πλειάδα g, g^a, g^s, g^c ισχύει αν $c = as$.

Παρ' όλα αυτά οι υπογραφές CBS έχουν ένα πολύ σημαντικό πρόβλημα: Επειδή το bit που ελέγχει την εγκυρότητα της υπογραφής είναι ιδιωτικό input στον υπογράφοντα, μπορεί κάλλιστα να παρακαμφθεί και να δοθεί μια άκυρη υπογραφή, ενώ ισχύει $b = 1$, ή αντίστροφα να δοθεί μια έγκυρη υπογραφή ενώ ισχύει $b = 0$. Αντίστοιχα και ο επαληθευτής μπορεί να αγνοήσει την υπογραφή και να κρίνει αυθαίρετα την εγκυρότητά της. Για να λυθεί αυτό πρέπει να υπάρξουν δύο αλλαγές στις CBS: Η τιμή του b πρέπει να υπολογίζεται από εξωτερικά δεδομένα και οι διαδικασίες υπογραφής και επαλήθευσης πρέπει να είναι ελέγξιμες.

Στη διατριβή επιλύουμε τα συγκεκριμένα προβλήματα εισάγοντας ένα νέο κρυπτογραφικό εργαλείο, τις Δημόσια Ελέγξιμες Υπο-Συνθήκη Τυφλές Υπογραφές (ΔΕΥΤΥ) – Publicly Auditable Conditional Blind Signatures (PACBS) [Gro+18; Gro+20]. Η εισαγωγή των PACBS, όπως και των CBS γίνεται με αυτόνομο τρόπο έτσι ώστε να μπορούν να χρησιμοποιηθούν και σε άλλα πρωτόκολλα εκτός από τις ηλεκτρονικές ψηφοφορίες.

Οι PACBS είναι ψηφιακές υπογραφές με τα ακόλουθα χαρακτηριστικά:

– Η υπογραφή είναι έγκυρη, εάν και μόνο εάν ένα κατηγόρημα σε δημόσια διαθέσιμα αλλά κρυπτογραφημένα δεδομένα είναι αληθές. Το κατηγόρημα αντικαθιστά το κρυφό bit b των CBS. Στην περίπτωση των εκλογών, αυτά τα δεδομένα είναι το διαπιστευτήριο που έχει δηλωθεί από τον ψηφοφόρο στην φάση εγγραφής και αυτό που πραγματικά χρησιμοποιείται κατά τη διάρκεια της ψηφοφορίας. Η ιδιότητα που πρέπει να ικανοποιηθεί για να είναι έγκυρη η υπογραφή είναι ότι πρέπει να κρυπτογραφούν το ίδιο μήνυμα, δηλαδή να αντιστοιχούν στο ίδιο διαπιστευτήριο.

– Η υπογραφή δεν μπορεί να επαληθευτεί δημόσια, όπως συμβαίνει με τις ψηφιακές υπογραφές. Στην περίπτωση των εκλογών, αυτό θα είχε ως αποτέλεσμα ο εξαναγκαστής να μάθει εάν χρησιμοποιήθηκε το σωστό διαπιστευτήριο. Οι PACBS είναι επαληθεύσιμες μόνο από έναν προκαθορισμένο επαληθευτή, ο οποίος στην περίπτωση ψηφοφορίας είναι η αρμόδια αρχή καταμέτρησης. Με πιο απλά λόγια, οι PACBS ενσωματώνουν τη λειτουργικότητα ελέγχου ισοδυναμίας μηνυμάτων PET των Jakobsson και Juels σε περιβάλλον προκαθορισμένου επαληθευτή.

– Για προστασία από διεφθαρμένες αρχές που αγνοούν το κατηγόρημα κατά τη διάρκεια της υπογραφής και της επαλήθευσης, οι PACBS παράγουν αποδεικτικά στοιχεία με τη μορφή μη-διαδραστικών αποδείξεων μηδενικής γνώσης που τους αναγκάζουν να ακολουθήσουν το πρωτόκολλο. Αυτές οι αποδείξεις παρέχουν δυνατότητα καθολικής επαληθευσιμότητας.

– Τέλος, οι PACBS επιτρέπουν στο χρήστη να τυφλώσει το μήνυμα, έτσι ώστε ο υπογράφων να μην μπορεί να συνδέσει αιτήματα υπογραφής με υπογραφές. Αυτό επιτρέπει πληροφοριοθεωρητική μυστικότητα.

Εκφράζουμε τυπικά αυτές τις επιθυμητές ιδιότητες χρησιμοποιώντας τις έννοιες της τυφλότητας, της υπο-συνθήκη επαληθευσιμότητας και της δημόσιας ελεγξιμότητας που ορίζονται με κρυπτογραφικά παίγνια. Δεδομένου ότι οι PACBS είναι ψηφιακές υπογραφές, πρέπει επίσης να ικανοποιούν την μη-πλαστογραφησιμότητα. Οι κοινές ιδιότητες με τις CBS ορίζονται με παρόμοιο τρόπο, ενώ η δημόσια ελεγξιμότητα βασίζεται στο παρακάτω παίγνιο: Ο αντίπαλος προσπαθεί να δημιουργήσει μια υπογραφή η οποία να επαληθεύεται ορθά αλλά να μη σέβεται το κατηγόρημα ή μια υπογραφή η οποία το αποτέλεσμα της επαλήθευσης να είναι διαφορετικό από το κατηγόρημα το οποίο είχε ληφθεί υπόψιν κατά τη δημιουργία της.

Στη διατριβή παρέχουμε δύο κατασκευές για τις PACBS, με βάση τις CBS. Στην πρώτη υπογράφων και επαληθευτής χρησιμοποιούν κοινό ιδιωτικό κλειδί επαλήθευσης και υπογραφής ενώ στη δεύτερη διαφορετικό. Και οι δύο ορίζονται σε μία ομάδα $\mathbb{G}$ τάξης q, όπου ισχύει η υπόθεση DDH. Υποθέτουν ένα κρυπτοσύστημα δημοσίου κλειδιού το οποίο διαθέτει την ιδιότητα IND-CPA και έχει χρησιμοποιηθεί για τη δημιουργία των κρυπτοκειμένων $\boldsymbol{C}_1, \boldsymbol{C}_2$. Επίσης χρησιμοποιούν δύο συναρτήσεις σύνοψεις $\mathsf{H}_1 : \mathbb{G}^4 \times \mathbb{G} \to \mathbb{G}$, $\mathsf{H}_2 : m \times \mathbb{G} \to \mathbb{Z}_q$ που μοντελοποιούνται ως τυχαία μαντεία. Κατά την αρχικοποίηση του συστήματος επιλέγονται $g_1, g_2, v, h_1 \leftarrow_\$ \mathbb{G}$. Τα ιδιωτικά κλειδιά υπογραφής και κρυπτογράφησης ορίζονται ως $s, z \leftarrow_\$ \mathbb{Z}_q$ ενώ τα δημόσια υπολογίζονται ως $k := g_1^s, h := h_1^z$. Οι διαδικασίες υπογραφής και επαλήθευσης ορίζονται παρακάτω:

- Τύφλωση (OSPACBS.Blind). Ο χρήστης υπολογίζει το $x := \mathsf{H}_1(\boldsymbol{C}_1, \boldsymbol{C}_2)$. Στη συνέχεια διαλέγει $u_1, u_2, d \leftarrow_\$ \mathbb{Z}_q$ και υπολογίζει τα:
  - $x^* := x g_1^{u_1} g_2^{u_2} v^d$
  - $e^* := \mathsf{H}_2(m, x^*)$
  - $e := e^* - d$, το οποίο και αποστέλλει στον υπογράφοντα.
- Υπογραφή (OSPACBS.BlindSign). Ο υπογράφων υπολογίζει τα:
  - $x := \mathsf{H}_1(\boldsymbol{C}_1, \boldsymbol{C}_2)$
  - $n := x g_2^{-y_2} v^{-e}$ με $y_2 \leftarrow_\$ \mathbb{Z}_q$
  - $\boldsymbol{N} := \mathsf{Enc}_h(n; t)$ με $t \leftarrow_\$ \mathbb{Z}_q$
  - $\boldsymbol{W} := (\boldsymbol{C}_2/\boldsymbol{C}_1)^\alpha \cdot \mathsf{Enc}_h(1, \gamma)$ με $\alpha, \gamma \in \mathbb{Z}_q^*$
  - $\boldsymbol{B} := (\boldsymbol{N} \cdot \boldsymbol{W})^s$
  - και τις παρακάτω αποδείξεις μηδενικής γνώσης, οι οποίες αναφέρονται συνολικά ως π_{Sign}

$$
\begin{aligned}
\pi_1 &\leftarrow \mathsf{NIZK}\{(h_1, h, n, \boldsymbol{N}), (t) : \boldsymbol{N} = \mathsf{Enc}_h(n; t)\} \\
\pi_2 &\leftarrow \mathsf{NIZK}\{(\boldsymbol{C}_1, \boldsymbol{C}_2, \boldsymbol{W}), (\alpha, \gamma) : \boldsymbol{W} = (\boldsymbol{C}_2/\boldsymbol{C}_1)^\alpha \cdot \mathsf{Enc}_h(1; \gamma)\} \\
\pi_3 &\leftarrow \mathsf{NIZK}\{(h, k, \boldsymbol{N}, \boldsymbol{W}, \boldsymbol{B}), (s) : \boldsymbol{B} = (\boldsymbol{N} \cdot \boldsymbol{W})^s \ \text{AND} \ k = g_1^s\}
\end{aligned}
$$

  - Τελικά η τυφλή υπογραφή είναι: $\overline{\beta} := (((n, \boldsymbol{N}, \boldsymbol{W}, \boldsymbol{B}), y_2), \pi_1, \pi_2, \pi_3)$ η οποία και στέλνεται στο χρήστη.
- Αποτύφλωση (OSPACBS.Unblind). Αρχικά επαληθεύονται οι αποδείξεις π_1, π_2, π_3. Στη συνέχεια υπολογίζονται τα $\overline{\boldsymbol{\sigma_1}} := \boldsymbol{B} \cdot \mathsf{Enc}_h(k^{u_1})$ and $\sigma_2 := y_2 + u_2$. Η τελική υπογραφή είναι $\overline{\sigma} := (x^*, e^*, \overline{\boldsymbol{\sigma}}_{\boldsymbol{1}}, \sigma_2)$.
- Επαλήθευση (OSPACBS.Verify). Αρχικά ελέγχεται αν $\mathsf{H}_2(m, x^*) \neq e^*$ οπότε και η διαδικασία τερματίζει. Σε διαφορετική περίπτωση υπολογίζεται τα:
  - validity $:= x^* \cdot g_2^{-\sigma_2} \cdot v^{-e^*}$
  - $\boldsymbol{M} := \mathsf{Enc}_h(\text{validity}; r_1)$ με $r_1 \in \mathbb{Z}_q$
  - $\boldsymbol{V} := \boldsymbol{M}^s$
  - $\boldsymbol{R} := \left(\frac{\boldsymbol{V}}{\overline{\sigma}_{\boldsymbol{1}}}\right)^\gamma$ με $\gamma \leftarrow_\$ \mathbb{Z}_q$.
  - και result $:= \mathsf{Dec}_z(\boldsymbol{R})$

* Παράγονται και οι παρακάτω αποδείξεις μηδενικής γνώσης, οι οποίες αναφέρονται συνολικά ως π_{Verify}

$$\pi_1 \leftarrow \mathsf{NIZK}\{(h_1, h, \boldsymbol{M}, \text{validity}), (r_1) : \boldsymbol{M} = \mathsf{Enc}_h(\text{validity}; r_1)\}$$
$$\pi_2 \leftarrow \mathsf{NIZK}\{(\boldsymbol{V}, \boldsymbol{M}), (s) : \boldsymbol{V} = \boldsymbol{M}^s\}$$
$$\pi_3 \leftarrow \mathsf{NIZK}\{(\boldsymbol{V}, \overline{\sigma}_{\boldsymbol{1}}, \boldsymbol{R}), (\gamma) : \boldsymbol{R} = \left(\frac{\boldsymbol{V}}{\overline{\sigma}_{\boldsymbol{1}}}\right)^{\gamma}\}$$
$$\pi_4 \leftarrow \mathsf{NIZK}\{(h_1, h, \text{result}, \boldsymbol{R}), (z) : \text{result} = \mathsf{Dec}_z(\boldsymbol{R})\}$$

Επίσης ορίζουμε και δύο λειτουργίες τις AuditSign και AuditVrfy για τον έλεγχο της διαδικασίας υπογραφής και επαλήθευσης. Η AuditSign ελέγχει αν $n = \mathsf{H}_1(\boldsymbol{C}_1, \boldsymbol{C}_2) g_2^{-y_2} v^{-e}$ και την απόδειξη π_{Sign}. Η AuditVrfy ελέγχει αν $\mathsf{H}_2(m, x^*) = e^*$ OR validity $\neq x^* \cdot g_2^{-\sigma_2} \cdot v^{-e^*}$ και την απόδειξη π_{Verify}.

Αποδεικνύουμε την ασφάλεια αυτών των κατασκευών βασιζόμενοι σε γνωστές κρυπτογραφικές υποθέσεις. Συγκεκριμένα:

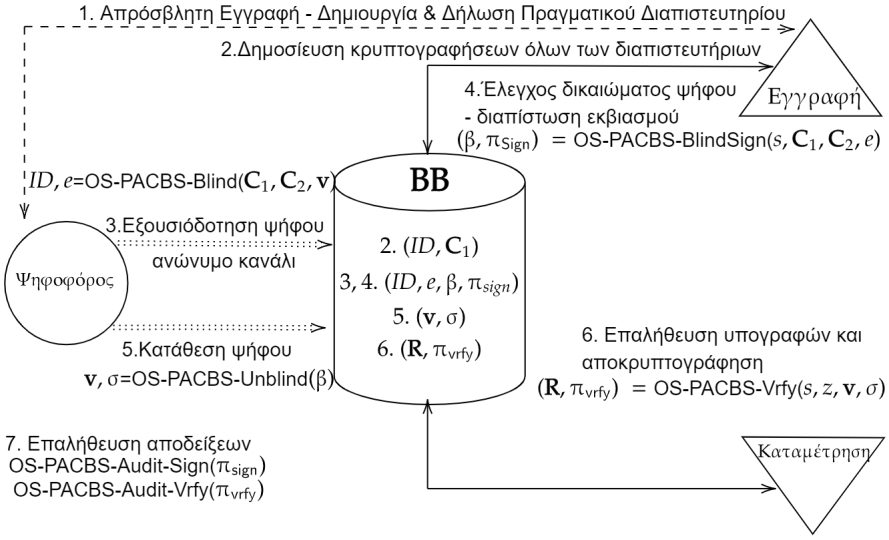
- Η τυφλότητα αποδεικνύεται χωρίς υποθέσεις με αντίστοιχο τρόπο, όπως έγινε στις CBS. Δηλαδή για κάθε 2 όψεις πρωτοκόλλων $_i$ που οδηγούν σε υπογραφές $\overline{\sigma}_j$, μπορούν να βρεθούν μοναδικά u_1, u_2, d που αντιστοιχίζουν οποιαδήποτε όψη σε οποιαδήποτε υπογραφή.
- Η μη-πλαστογραφησιμότητα αποδεικνύεται με την υπόθεση ότι η κατασκευή OSCBS είναι μη-πλαστογραφήσιμη. Συγκεκριμένα κατασκευάζουμε μία αναγωγή στην οποία ο αντίπαλος χρησιμοποιεί μια πλαστογράφηση OSPACBS για να κατασκευάσει μία πλαστογράφηση OSCBS με το ίδιο ακριβώς πλεονέκτημα. Αυτό όμως σύμφωνα με την απόδειξη της μη-πλαστογραφησιμότητας του CBS δεν είναι εφικτό.
- Η υπο συνθήκη επαληθευσιμότητα αποδεικνύεται με την υπόθεση ότι το χρησιμοποιούμενο κρυπτοσύστημα διαθέτει την ιδιότητα IND-CPA.
- Η δημόσια ελεγξιμότητα αποδεικνύεται με βάση την ορθότητα των αποδείξεων μηδενικής γνώσης.

Ψηφοφορίες με PACBS Η κύρια συνεισφορά της εργασίας είναι ένα πρωτόκολλο ηλεκτρονικών ψηφοφοριών που συνδυάζει τα σχήματα FOO και JCJ χρησιμοποιώντας τις PACBS. Η γενική αρχιτεκτονική του πρωτοκόλλου [GPZ17] έχει τους εξής στόχους:

- Να μειώσει την πολυπλοκότητα βημάτων του JCJ, καθιστώντας την γραμμική στο πλήθος των ψηφοφόρων.

– Να παρέχει πιο ισχυρές εγγυήσεις μυστικότητας ψήφου χωρίς να υπάρχει ανάγκη εμπιστοσύνης στους καταμετρητές για τη συγκεκριμένη ιδιότητα.

Οι παραπάνω στόχοι επιτυγχάνονται όπως προαναφέραμε με τις PACBS στο πρωτόκολλο ψηφοφοριών $\mathsf{VS}_{\mathsf{PACBS}}$ [Gro+18] που αποτελείται από τα εξής στάδια:



Σχήμα 1: Σχήμα ψηφοφορίας με PACBS από την πλευρά ενός ψηφοφόρου

– *Αρχικοποίηση* που υλοποιείται από τις λειτουργίες $\mathsf{VS}_{\mathsf{PACBS}}$.Setup και $\mathsf{VS}_{\mathsf{PACBS}}$.SetupElection.

– *Εγγραφή* μέσω της $\mathsf{VS}_{\mathsf{PACBS}}$.Register.

– *Ψηφοφορία* που χωρίζεται στις φάσεις της εξουσιοδότησης και κατάθεσης μέσω των λειτουργιών $\mathsf{VS}_{\mathsf{PACBS}}$.Vote, $\mathsf{VS}_{\mathsf{PACBS}}$.Cast. Επίσης χρησιμοποιείται η $\mathsf{VS}_{\mathsf{PACBS}}$.Valid η οποία απομακρύνει ταυτόσημα ψηφοδέλτια από αποθετήριο αμέσως μετά την κατάθεση. Εδώ χρησιμοποιούμε και τις βοηθητικές λειτουργίες $\mathsf{VS}_{\mathsf{PACBS}}$.fakekey, $\mathsf{VS}_{\mathsf{PACBS}}$.chaffvote, $\mathsf{VS}_{\mathsf{PACBS}}$.dupauth για την δημιουργία ψεύτικου διαπιστευτηρίου, την εισαγωγή άκυρων ψήφων και το ξεκαθάρισμα διπλότυπων αιτήσεων εξουσιοδότησης αντίστοιχα.

– *Καταμέτρηση*, που υλοποιείται από την λειτουργία $\mathsf{VS}_{\mathsf{PACBS}}$.Tally.

– *Επαλήθευση* που αποτελείται από τις $\mathsf{VS}_{\mathsf{PACBS}}$.VerifyBallot, $\mathsf{VS}_{\mathsf{PACBS}}$.Verify.

Το πρωτόκολλο εκτελείται από την αρχή εκλογών EA η οποία διαχωρίζεται στις αρχές εγγραφής και καταμέτρησης (αντίστοιχα RA, TA). Με τη σειρά τους αυτές απαρτίζονται από πολλά μέλη με αντικρουόμενα συμφέροντα, οι οποίες μοιράζονται κρυπτογραφικά κλειδιά.

Πιο αναλυτικά οι φάσεις του πρωτοκόλλου εκτελούνται ως εξής:

Αρχικοποίηση Οι RA, TA εκτελούν την λειτουργία $\mathsf{VS}_{\mathsf{PACBS}}$.Setup η οποία παράγει τις παραμέτρους του PACBS, μεταξύ των οποίων τα ιδιωτικά κλειδιά υπογραφής s και κρυπτογράφησης z που μοιράζονται στα μέλη τους.

Εγγραφή Η RA και κάθε ψηφοφόρος V_i εκτελούν την λειτουργία $\mathsf{VS}_{\mathsf{PACBS}}$.Register μέσω ενός απρόσβλητου καναλιού (πχ. με φυσική παρουσία). Έτσι δημιουργούνται τα πραγματικά διαπιστευτήρια. Πιο συγκεκριμένα:

- Ο ψηφοφόρος V_i εγγράφεται στις εκλογές και λαμβάνει το κρυπτογραφικό διαπιστευτήριο που θα τον αντιπροσωπεύει. Υποστηρίζεται τόσο το πρωτόκολλο με συνθηματικά του Selections [UH12] αλλά και το πρωτόκολλο με υπογραφές προκαθορισμένου επαληθευτή του CIVITAS [CCM08]. Τα διαπιστευτήρια αυτά μπορούν να χρησιμοποιηθούν σε πολλές εκλογές. Για τη φάση της εγγραφής ισχύουν οι υποθέσεις του JCJ. Θα περιγράψουμε την φάση της εγγραφής που βασίζεται στο πρωτόκολλο Selections και χρησιμοποιεί *συνθηματικά πανικού* [CH08]:
  - Η αρχή εγγραφής επιλέγει ένα κοινό σύνολο από λέξεις (λεξικό).
  - Τα συνθηματικά θα αποτελούνται από τον συνδυασμό k λέξεων του λεξικού.
  - Κάθε ψηφοφόρος επιλέγει έναν συνδυασμό για το πραγματικό συνθηματικό.
  - Το συνθηματικό αυτό αντιστοιχίζεται σε διαπιστευτήριο $\theta_i \in \mathbb{Z}_q$ χρησιμοποιώντας μια συνάρτηση $\phi : \{0,1\}^* \to \mathbb{Z}_q$.
  - Η αρχή κρυπτογραφεί το g^{θ_i} και παρέχει μια απόδειξη ορθής κρυπτογράφησης.
  - Η διαδικασία επαναλαμβάνεται α φορές για επαλήθευση της ορθότητας.
- Μετά την ολοκλήρωση της φάσης της εγγραφής η αρχή επανακρυπτογραφεί όλα τα διαπιστευτήρια και τα εκχωρεί στο κεντρικό αποθετήριο των εκλογών το οποίο πλέον είναι ένα σύνολο $\{(i, \boldsymbol{C}_{i1})\}$.

Μόλις ολοκληρωθεί η περίοδος εγγραφής η EA εκτελεί την $\mathsf{VS}_{\mathsf{PACBS}}$.SetupElection όπου δημοσιεύονται τα δημόσια κλειδιά των αρχών εγγραφής και καταμέτρησης καθώς και η λίστα των υποψηφίων με την κατάλληλη κωδικοποίηση.

Ψηφοφορία

- Η διαδικασία ψηφοφορίας ξεκινά με την αίτηση εξουσιοδότησης μέσω της $\mathsf{VS}_{\mathsf{PACBS}}$.Vote.

* Ο ψηφοφόρος V_i εισάγει το συνθηματικό. Αν καταχωρήσει τον συνδυασμό που δήλωσε κατά την εγγραφή του, τότε μέσω της ϕ παράγεται το ίδιο διαπιστευτήριο θ_i.

* Οποιοσδήποτε άλλος συνδυασμός θεωρείται συνθηματικό πανικού και δηλώνει ότι ο ψηφοφόρος παρακολουθείται από τον εξαναγκαστή. Αυτό σημαίνει ότι δεν απορρίπτεται από το σύστημα. Εκτελείται μια παρόμοια διαδικασία με αυτή που ακολουθήθηκε κατά την εγγραφή και η οποία οδηγεί σε διαφορετικό διαπιστευτήριο $\theta_i' \in \mathbb{Z}_q$. Η διαδικασία αυτή υλοποιείται μέσω της $\mathsf{VS}_{\mathsf{PACBS}}$.fakekey.

* Αν δοθεί συνδυασμός λέξεων που δεν ανήκει στο λεξικό, τότε θεωρείται ότι έχει γίνει λάθος πληκτρολόγηση και ζητείται επαναεισαγωγή.

Από την διαδικασία αυτή παράγεται ένα κρυπτογραφικό διαπιστευτήριο το οποίο και κρυπτογραφείται ως $\boldsymbol{C}_{i2}$. Οι ψηφοφόροι αφού επιλέξουν την προτίμησή τους vt_i, την κρυπτογραφούν δημιουργώντας έτσι το κρυπτοκείμενο $\boldsymbol{v}_i$. Το ψηφοδέλτιο b_i περιέχει επιπλέον μια απόδειξη μηδενικής γνώσης $\pi_{\boldsymbol{v}_i}$ ότι η vt_i είναι έγκυρη και γνωστή. Στη συνέχεια δημιουργούν την αίτηση εξουσιοδότησης, μέσω του πρωτοκόλλου OSPACBS.Sign σε συνεργασία με την RA. Κατ' αρχήν τυφλώνουν το ψηφοδέλτιο τους, με τον αλγόριθμο OSPACBS.Blind παράγοντας την τιμή e_i. Μαζί με αυτή επισυνάπτουν στην αίτηση τους την ταυτότητά του i, το $\boldsymbol{C}_{i2}$. Το $\boldsymbol{C}_{i1}$ μπορεί να ανακτηθεί από το κεντρικό αποθετήριο.

– Για να μην είναι δυνατή η επίθεση εξαναγκασμένης αποχής υποθέτουμε ότι στο πρωτόκολλο συμμετέχουν και εξωτερικές οντότητες (π.χ. μη κυβερνητικές οργανώσεις). Αυτές, αλλά και ενδιαφερόμενοι ψηφοφόροι, καταθέτουν ψηφοδέλτια με τυχαία διαπιστευτήρια για όλους τους ψηφοφόρους, εκτελώντας την λειτουργικότητα $\mathsf{VS}_{\mathsf{PACBS}}$.chaffvote. Αυτοί οι ψήφοι φυσικά δε θα μετρήσουν καθώς η πιθανότητα να είναι έγκυρο το ψηφοδέλτιο είναι αμελητέα.

– Η αρχή εγγραφής, ελέγχει για διπλότυπες αιτήσεις που περιέχουν το ίδιο διαπιστευτήριο χρησιμοποιώντας τη λειτουργία $\mathsf{VS}_{\mathsf{PACBS}}$.dupauth, αν ο ψηφοφόρος έχει δικαίωμα ψήφου και υπογράφει την αίτηση χρησιμοποιώντας τον αλγόριθμο OSPACBS.BlindSign. Παράγεται η υπογραφή $\overline{\beta}_i$ η οποία σύμφωνα με το PACBS είναι έγκυρη αν και μόνο $\mathsf{pred}(\boldsymbol{C}_{i1}, \boldsymbol{C}_{i2}) = 1$ ή ισοδύναμα $\mathsf{Dec}(\boldsymbol{C}_{i1}) = \mathsf{Dec}(\boldsymbol{C}_{i2})$. Ταυτόχρονα παράγεται και η απόδειξη έγκυρης υπογραφής $\pi_{i,\mathsf{Sign}}$.

– Ο ψηφοφόρος επαληθεύει την απόδειξη $\pi_{i,\mathsf{Sign}}$. Στη συνέχεια αποτυφλώνει την υπογραφή με τον αλγόριθμο OSPACBS.Unblind και καταθέτει το ψηφοδέλτιο του. Εδώ πάλι πρέπει να χρησιμοποιηθεί ένα ανώνυμο κανάλι. Άλλωστε σύμφωνα με το JCJ το ανώνυμο κανάλι είναι αναγκαία συνθήκη για αντίσταση

στον εξαναγκασμό στο πλαίσιο JCJ. Κατά συνέπεια, το ψηφοδέλτιο περιέχει μια υπογραφή $\overline{\sigma}_i$ που ορίζει εάν η ψήφος πρέπει να μετρηθεί ή όχι. Τελικά δηλαδή $\mathsf{b}_i = (\boldsymbol{v}_i, \pi_{v_i}, \overline{\sigma}_i)$.

– Μόλις κατατεθεί η ψήφος, BB ελέγχει ότι δεν υπάρχει ακριβές αντίγραφό της ήδη κατατεθειμένο χρησιμοποιώντας τη λειτουργία $\mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Valid}$.

Καταμέτρηση Εκτελείται με τον αλγόριθμο $\mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Tally}$ ο οποίος εκτελείται από την TA. Στόχος είναι να επαληθευθούν οι υπογραφές ώστε να αποφασιστεί ποιες ψήφους θα μετρηθούν. Εσωτερικά η $\mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Tally}$ εκτελεί την $\mathsf{OSPACBS}.\mathsf{Verify}$ ώστε να καταμετρηθούν μόνο εκείνες οι ψήφοι που συνοδεύονται από έγκυρες υπογραφές. Για να μην γίνει αυτό αντιληπτό από τον εξαναγκαστή η συγκεκριμένη λειτουργία εκτελείται σε δύο φάσεις: Στην πρώτη, παράγεται το αποτέλεσμα της επαλήθευσης κρυπτογραφημένο δηλαδή η τιμή $\boldsymbol{R}_i$ και οι αποδείξεις $\pi_{i,\mathsf{Verify}} = (\pi_{i1}, \pi_{i2}, \pi_{i3})$. Στη συνέχεια μεσολαβεί ένα δίκτυο μίξης με στόχο να μην μπορεί να συσχετιστεί το αποτέλεσμα της υπογραφής με τον ψηφοφόρο που την κατέθεσε. Μετά από αυτό εκτελείται η λειτουργικότητα αποκρυπτογράφησης $\mathsf{Dec}(\boldsymbol{R_i})$ η οποία επιπλέον παράγει την απόδειξη π_{i4}. Η αντίστοιχη ψήφος θα καταμετρηθεί μόνο αν το αποτέλεσμα της αποκρυπτογράφησης είναι 1. Η ιδιότητα της υπό όρους επαλήθευσης των PACBS, αλλά και τα υπόλοιπα συστατικά του πρωτοκόλλου, καθιστούν τις περιπτώσεις έγκυρων και άκυρων υπογραφών αδιάκριτες για τον εξαναγκαστή.

Επαλήθευση Κάθε ψηφοφόρος μπορεί να επαληθεύσει την ψήφο του χρησιμοποιώντας τον αλγόριθμο $\mathsf{VS}_{\mathsf{PACBS}}.\mathsf{VerifyBallot}$ (ατομική επαληθευσιμότητα). Για τον σκοπό αυτό χρησιμοποιείται η τυχαιότητα της κρυπτογράφησης του $\boldsymbol{v}_i$ και του $\boldsymbol{C}_{i2}$ ώστε να διαπιστωθεί αν το ψηφοδέλτιο και το διαπιστευτήριο κρυπτογραφήθηκαν σωστά. Χρησιμοποιείται επίσης ο αλγόριθμος $\mathsf{AuditSign}$ για να επαληθευτεί η διαδικασία της υπογραφής. Οποιοσδήποτε ενδιαφερόμενος μπορεί να εκτελέσει τον αλγόριθμο $\mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Verify}$ για να ελέγξει τη λειτουργία του πρωτοκόλλου (καθολική επαληθευσιμότητα). Ο συγκεκριμένος αλγόριθμος ελέγχει όλες τις αποδείξεις που παράγονται κατά το πρωτόκολλο. Περιλαμβάνει επίσης και την λειτουργικότητα του $\mathsf{AuditVrfy}$ τροποποιημένη βέβαια ώστε να λαμβάνει υπόψιν τη διάσπαση των λειτουργιών του λόγω του δικτύου μίξης. Αν οι παραπάνω έλεγχοι εκτελεστούν με επιτυχία, τότε όλοι μπορούν να είναι σίγουροι ότι το πρωτόκολλο εκτελέστηκε σωστά. Ο ψηφοφόρος όμως ο οποίος γνωρίζει με βάση το συνθηματικό που έδωσε αν έχει έγκυρο διαπιστευτήριο ή όχι, οδηγείται επιπλέον στο συμπέρασμα ότι η ψήφος του μετρήθηκε, χωρίς κάτι τέτοιο να αποκαλύπτεται στον εξαναγκαστή.

Ανάλυση ασφάλειας Για να αποδείξουμε την ασφάλεια του συστήματός μας, εξετάζουμε διάφορους τυπικούς ορισμούς για την επαληθευσιμότητα [Cor+16; KZZ15a;

SFC15], τη μυστικότητα [Ber+15] και την αντίσταση στον εξαναγκασμό [JCJ05; Alw+15; FQS19]. Προσαρμόζουμε αυτούς τους ορισμούς στο πρωτόκολλό μας, διατηρώντας όμως τις κύριες ιδέες τους. Με την ευκαιρία διερευνούμε επίσης τις σχέσεις μεταξύ των ιδιοτήτων ασφαλείας αυτών.

Τα αποτελέσματά μας σχετικά με την ασφάλεια του πρωτοκόλλου μας περιγράφονται παρακάτω:

Επαληθευσιμότητα Παρέχεται ατομική και καθολική επαληθευσιμότητα.

Η ατομική επαληθευσιμότητα παρέχεται στο μοντέλο του [SFC15]. Ο αντίπαλος προσπαθεί να επιτύχει στις δύο παρακάτω επιθέσεις: Πρώτον, προσπαθεί να δημιουργήσει μία σύγκρουση (clash), όπου σε δύο ή περισσότερους διαφορετικούς ψηφοφόρους ανατίθεται το ίδιο ψηφοδέλτιο, με αποτέλεσμα ο αντίπαλος να έχει στη διάθεσή του ένα τουλάχιστον ψηφοδέλτιο το οποίο μπορεί να χρησιμοποιήσει για να καταθέσει την ψήφο της προτίμησής του. Η επίθεση αυτή δεν μπορεί να επιτύχει στο σύστημά μας γιατί υποθέτουμε ότι ο η φάση εγγραφής έχει αμελητέο λάθος ορθότητας (αφού επαναλαμβάνεται α φορές) και ότι ο ψηφοφόρος εισάγει τυχαιότητα κατά την φάση της δημιουργίας του ψηφοδελτίου. Με αυτόν τον τρόπο η πιθανότητα να ταυτιστούν δύο ψηφοδέλτια είναι αμελητέα. Σε ένα συνηθισμένο σύστημα, η προστασία από αυτή την επίθεση θα αρκούσε, καθώς όλες οι ψήφοι που βρίσκονται στο BB και έχουν έγκυρες αποδείξεις θα καταμετρηθούν. Στο σύστημα VS.PACBS όμως αυτό δεν ισχύει - άλλωστε για τον λόγο αυτό οι αποδείξεις κρυπτογράφησης ψήφου και διαπιστευτήριου δεν μπορούν να χρησιμοποιηθούν ως απόδειξη για πώληση ψήφο ή εξαναγκασμό. Ο αντίπαλος λοιπόν θα μπορούσε να προσπαθήσει να αφήσει ένα ψηφοδέλτιο στο BB αλλά με τέτοιον τρόπο ώστε να είναι άκυρο, επηρεάζοντας πάλι το αποτέλεσμα. Αυτό θα μπορούσε να γίνει αν ο αντίπαλος μπορούσε να ανακτήσει το διαπιστευτήριο από την κρυπτογραφημένη του μορφή στο BB. Αν και ως κακόβουλη RA διαθέτει τα κλειδιά αποκρυπτογράφησης, για να το υπολογίσει θα πρέπει να επιλύσει ένα διακριτό λογάριθμο. Εναλλακτικά θα μπορούσε να χαρακτηρίσει την ψήφο ως διπλότυπη. Αν υποθέσουμε όμως ότι το BB διατηρεί τη σειρά κατάθεσης των ψήφων, τότε αυτή η επίθεση εμποδίζεται από την ορθότητα των αποδείξεων μηδενικής γνώσης που παρέχονται κατά τη διαδικασία εντοπισμού των διπλότυπων ψήφων. Τέλος, ως διεφθαρμένη αρχή θα μπορούσε να δώσει μια άκυρη υπογραφή σε μια έγκυρη ψήφο, κάτι που εμποδίζεται όμως από τη δημόσια επαληθευσιμότητα των PACBS.

Σχετικά με την καθολική επαληθευσιμότητα βασιζόμαστε στην έννοια της ισχυρής επαληθευσιμότητας του [Cor+14]. Όπως προαναφέραμε, η καθολική επαληθευσιμότητα επιτρέπει στους ψηφοφόρους να επαληθεύσουν τη φάση της καταμέτρησης. Οι περισσότεροι ορισμοί όμως αφορούν συστήματα στα οποία δεν υπάρχει αρχή

εγγραφής. Η τελευταία, αν υφίσταται, μπορεί να επηρεάσει την φάση της καταμέτρησης αν και δε συμμετέχει ενεργά σε αυτήν, πχ. δίνοντας το ίδιο διαπιστευτήριο σε πολλούς ψηφοφόρους ή δημιουργώντας ψεύτικα διαπιστευτήρια για να εισάγει ψήφους που δεν αντιστοιχούν σε πραγματικούς ψηφοφόρους. Το μοντέλο της ισχυρής επαληθευσιμότητας μπορεί να εφαρμοστεί σε τέτοια πρωτόκολλα. Για να αποδείξουμε ότι το σύστημά μας διαθέτει αυτή την ιδιότητα, το τροποποιούμε ώστε τα διαπιστευτήρια να μην παράγονται μόνο από την αρχή εγγραφής, αλλά σε συνεργασία με κάθε ψηφοφόρο. Η βασική υπόθεση ασφάλειας είναι ότι η αρχή εγγραφής και το δεν είναι ταυτόχρονα διεφθαρμένες. Στην περίπτωσή μας υποθέτουμε ότι ο αντίπαλος ελέγχει την αρχή καταμέτρησης αλλά και εγγραφής, αλλά όχι το BB, το οποίο αποτιμά σωστά τις αποδείξεις μηδενικής γνώσης και τον έλεγχο διπλοτύπων. Επίσης υποθέτουμε ότι το BB, δεν εισάγει νέες ψήφους αλλά πιο σημαντικά διατηρεί τη σειρά εισαγωγής των υπάρχοντων. Στόχος του, αντιπάλου, όπως και στο [Cor+14], είναι να κατασκευάσει ένα αποτέλεσμα το οποίο να περνάει επιτυχώς την επαλήθευση, αλλά να περιέχει είτε περισσότερες, είτε αλλαγμένες είτε διεγραμμένες ψήφους. Κάτι τέτοιο δεν είναι δυνατόν, όμως για τους παρακάτω λόγους:

- Για να αλλαχθεί το περιεχόμενο μιας ψήφου, πρέπει να παραβιαστεί η ορθότητα της απόδειξης π_v.
- Ο μόνος τρόπος να διαγραφεί μια ψήφος, με δεδομένο ότι το BB, είναι έντιμο είναι να ακυρωθεί. Αυτό δεν μπορεί να συμβεί ούτε στη φάση της εξουσιοδότησης, ούτε στη φάση της καταμέτρησης, χωρίς να παραβιαστεί η δημόσια επαληθευσιμότητα των υπογραφών PACBS. Επίσης δεν μπορεί να χαρακτηριστεί διπλότυπη, χωρίς να παραβιαστεί η ορθότητα των αποδείξεων της αρχής εγγραφής.
- Τέλος, δεν μπορούν να προστεθούν επιπλέον ψήφοι χωρίς να είναι γνωστά τα αντίστοιχα διαπιστευτήρια καθώς κάτι τέτοιο προϋποθέτει την επίλυση ενός προβλήματος διακριτού λογαρίθμου.

Επιπλέον, για να μην είναι δυνατές τέτοιες επιθέσεις στην καταμέτρηση πρέπει και το δίκτυο μίξης που χρησιμοποιείται εκεί να είναι επαληθεύσιμο.

Τέλος, το σύστημά μας *δεν* παρέχει δημόσια επαληθευσιμότητα καταλληλότητας αν μπορούσε να εξακριβωθεί δημόσια *αν μία συγκεκριμένη ψήφος* θα καταμετρηθεί, οποιοσδήποτε θα μπορούσε να διαπιστώσει αν έχει χρησιμοποιηθεί το σωστό διαπιστευτήριο. Κατά συνέπεια δεν θα υπήρχε δυνατότητα για προστασία από εξαναγκασμό. Η επαληθευσιμότητα καταλληλότητας έρχεται έμμεσα μέσω της καθολικής επαληθευσιμότητας για όλους τους ψηφοφόρους και όχι για κάθε έναν ξεχωριστά. Το σύστημά μας, ωστόσο, παρέχει ιδιωτική επαληθευσιμότητα καταλληλότητας, το

οποίο μπορεί να αποδειχθεί με μία διαδικασία παρόμοια με την μεμονωμένη επαληθευσιμότητα.

Προστασία από εξαναγκασμό Το πρωτόκολλο ηλεκτρονικής ψηφοφορίας που προτείνουμε με βάση τις PACBS παρέχει προστασία από εξαναγκασμό στο μοντέλο JCJ, υπό την υπόθεση ότι οι PACBS διαθέτουν υπό συνθήκη επαληθευσιμότητα και τις υπόλοιπες υποθέσεις του JCJ περί ορθής φάσης εγγραφής, τίμιων αρχών, ανώνυμης κατάθεσης ψήφου και αβεβαιότητα του εξαναγκαστή για την συνολική συμπεριφορά των ψηφοφόρων. Για να αποδείξουμε αυτή την ιδιότητα χρησιμοποιούμε τη μεθοδολογία του [JCJ05] και συγκρίνουμε την πιθανότητα επιτυχίας του αντιπάλου σε δύο κρυπτογραφικά παίγνια, όπου λαμβάνεται ένα τυχαίο bit b. Όταν αυτό λάβει την τιμή 0 ο ψηφοφόρος παράγει ένα ψεύτικο διαπιστευτήριο χρησιμοποιώντας την λειτουργία fakekey το οποίο και δίνει στον αντίπαλο. Σε κάποια άλλη ιδιωτική στιγμή καταθέτει την πραγματική του ψήφο με το κανονικό διαπιστευτήριο. Όταν λάβει την τιμή 1, ο ψηφοφόρος παραδίδει το πραγματικό διαπιστευτήριο στον αντίπαλο, επιτρέποντας να προσομοιωθεί και δεν καταθέτει την δική του ψήφο. Και στις δύο περιπτώσεις ο αντίπαλος επιλέγει την ψήφο, καθώς ο στόχος είναι να μαντέψει ο αντίπαλος αν χρησιμοποιήθηκε το σωστό διαπιστευτήριο. Επιπλέον μπορεί να χρησιμοποιήσει για τον σκοπό αυτό ψηφοφόρους που έχει υπό τον έλεγχό του. Το γεγονός ότι όταν $b = 0$ υπάρχει μία παραπάνω ψήφος, καλύπτεται από την υπόθεση ότι ο αντίπαλος δεν μπορεί να γνωρίζει με βεβαιότητα την συμπεριφορά των μη ελεγχόμενων ψηφοφόρων. Χρησιμοποιούνται δύο παίγνια, ώστε να μην υπάρξει περίπτωση να μπορεί να εξαχθεί η συμπεριφορά του ψηφοφόρου από το αποτέλεσμα - αν π.χ. πρέπει να ψηφιστεί κάποιος υποψήφιος ο οποίος τελικά δεν λάβει καμία ψήφο, Έτσι συγκρίνουμε την πιθανότητα επιτυχίας του εξαναγκαστή στο 'πραγματικό' παίγνιο που αναπαριστά το πρωτόκολλο με αυτήν που έχει σε ένα 'ιδανικό' παίγνιο στο οποίο η καταμέτρηση γίνεται από μια έμπιστη, ιδανική λειτουργικότητα και ο αντίπαλος δεν έχει πρόσβαση σε κρυπτογραφικά δεδομένα. Βλέπει δηλαδή μόνο το αποτέλεσμα. Το ιδανικό παίγνιο εκφράζει δηλαδή την μέγιστη προστασία από εξαναγκασμό που μπορεί να παρέχει ένα πρωτόκολλο ψηφοφορίας. Η σύγκριση αυτή αποκαλύπτει το πλεονέκτημα επιτυχίας του αντιπάλου που οφείλεται στην χρήση των συγκεκριμένων κρυπτογραφικών κατασκευών μας.

Στη διατριβή αποδεικνύουμε ότι το πλεονέκτημα του αντιπάλου στο πραγματικό παίγνιο διαφέρει από το αντίστοιχο στο ιδανικό με αμελητέα τρόπο. Η απόδειξη αποτελείται από τρία στάδια: Πρώτον, δείχνουμε ότι η ψήφος η οποία συνοδεύεται από το πραγματικό διαπιστευτήριο δεν βοηθά τον εκβιαστή, καθώς δεν μπορεί να την ξεχωρίσει και να δει ότι μέτρησε. Αυτό οφείλεται στην ανώνυμη κατάθεση ψήφων αλλά κυρίως στην υπό-συνθήκη επαληθευσιμότητα των PACBS. Η μόνη

διαφορά της ψήφου αυτής από εκείνη που κατέθεσε ο εκβιαστής είναι ότι συνοδεύεται από μια έγκυρη υπογραφή, κάτι που όμως μπορεί να διαπιστώσει ο αντίπαλος. Δεύτερον, δείχνουμε ότι ο αντίπαλος δεν μπορεί να ξεχωρίσει αν λαμβάνει ψεύτικο ή αληθινό διαπιστευτήριο. Η κρυπτογράφηση του αληθινού είναι διαθέσιμη σε όλους στο αποθετήριο σε κρυπτογραφημένη μορφή. Αν ο αντίπαλος μπορούσε να το διαχωρίσει από την κρυπτογράφηση που λαμβάνει από την fakekey τότε θα μπορούσε να σπάσει την ιδιότητα IND-CPA του χρησιμοποιούμενου κρυπτοσυστήματος. Επίσης, ούτε οι ψήφοι των μη ελεγχόμενων ψηφοφόρων δεν μπορούν να βοηθήσουν τον αντίπαλο, καθώς αν στο πραγματικό παίγνιο τις αντικασταστήσουμε από τυχαίες τιμές και το αποτέλεσμα βγει από τις αρχικές τους τιμές, ο αντίπαλος δεν μπορεί να καταλάβει τη διαφορά λόγω των ιδιοτήτων της υπο συνθήκης επαληθευσιμότητας και IND-CPA. Επιπλέον λόγω του ανώνυμου καναλιού, ο εξαναγκαστής δεν μπορεί να αποφανθεί αν ο στόχος του ψήφισε ή όχι. Άρα σύμφωνα με το μοντέλο JCJ, το πρωτόκολλό μας παρέχει προστασία από εξαναγκασμό.

Μυστικότητα Αναλύουμε την μυστικότητα ακολουθώντας το μοντέλο BPRIV [Ber+15]. Στόχος αυτού του μοντέλου είναι να εξετάσει αν τα κρυπτογραφικά δεδομένα που υπάρχουν στο BB βοηθούν τον αντίπαλο να μαντέψει την προτίμηση κάποιου ψηφοφόρου, περισσότερο από ό,τι θα μπορούσε κρίνοντας μόνο από το αποτέλεσμα της καταμέτρησης.

Αυτό εκφράζεται μέσω ενός παίγνιου στο οποίο ο αντίπαλος βλέπει δύο αποθετήρια τα $\mathsf{BB}_0, \mathsf{BB}_1$. Οι κανονικοί ψηφοφόροι καταθέτουν διαφορετικές ψήφους (επιλεγμένες από τον αντίπαλο) στο κάθε ένα. Στο τέλος, γίνεται η καταμέτρηση πάντα στο BB_0. Με ομοιόμορφη πιθανότητα το αποτέλεσμα και ένα από τα δύο αποθετήρια παρουσιάζεται στον αντίπαλο, ο οποίος πρέπει να μαντέψει ποιο είδε.

Στην περίπτωσή μας τροποποιούμε το συγκεκριμένο μοντέλο, ώστε να εκφράζει μη έμπιστη ιδιωτική καταμέτρηση - να εκτελείται δηλαδή από τον αντίπαλο. Ονομάζουμε το συγκεκριμένο μοντέλο U-BPRIV και με βάση αυτό αποδεικνύουμε ότι το πρωτόκολλο ηλεκτρονικό ψηφοφοριών δίνει στον αντίπαλο αμελητέο πλεονέκτημα ώστε να νικήσει στο συγκεκριμένο παίγνιο με την υπόθεση ότι οι PACBS παρέχουν τυφλότητα και δημόσια επαληθευσιμότητα ώστε να μην μπορεί να ξεχωρίσει τα δύο αποθετήρια χρησιμοποιώντας ένα ακυρωμένο ψηφοδέλτιο. Αξίζει να σημειωθεί ότι το συγκεκριμένο αποτέλεσμα είναι ένα ακόμα βήμα το οποίο συσχετίζει την μυστικότητα με την επαληθευσιμότητα μετά το [CGG19]. Άλλες υποθέσεις για να ισχύει το συγκεκριμένο αποτέλεσμα είναι να μην απορρίπτεται καμία ψήφος και να μην υπάρχει συσχέτιση της σειράς κατάθεσης τους. Αυτό μπορεί να γίνει μέσω ενός ανώνυμου καναλιού.

Αέναη ιδιωτικότητα Το μοντέλο U-BPRIV μας οδηγεί στην εξής παρατήρηση: Αν δεν είναι απαραίτητο να εμπιστευθούμε τους καταμετρητές για την μυστικότητα της ψήφου, σημαίνει ότι το εκλογικό σύστημα παρέχει προστασία ακόμα και εναντίον ενός υπολογιστικά ισχυρού αντιπάλου $\hat{\mathcal{A}}$, καθώς είτε ο αντίπαλος αποκρυπτογραφεί με τα κλειδιά είτε επειδή παρακάμπτει την κρυπτογραφική προστασία το αποτέλεσμα είναι το ίδιο. Ένα τέτοιο πρωτόκολλο λοιπόν, αν συνδυαστεί με ανώνυμα κανάλια επικοινωνίας,μπορεί να παρέχει αέναη ιδιωτικότητα.

Στη βιβλιογραφία, μέχρι τώρα η ανάλυση της αέναης ιδιωτικότητας ήταν μονοδιάστατη. Συγκεκριμένα, στο [Ara+13] ορίστηκε η έννοια της *πρακτικής αέναης ιδιωτικότητας*, όπου ο $\hat{\mathcal{A}}$ έχει πρόσβαση μόνο στα δημόσια εκλογικά δεδομένα του BB και όχι στα ιδιωτικά δεδομένα που ανταλλάσσονται μεταξύ των ψηφοφόρων και των αρχών, ή μεταξύ των διάφορων παικτών που απαρτίζουν τις εκλογικές αρχές. Με αυτή την έννοια αποδεικνύεται ότι διαθέτουν αέναη ιδιωτικότητα συστήματα όπως το [MN06; MN10; DGA12]. Το συγκεκριμένο μοντέλο δεν είναι πλήρες, καθώς δεν λαμβάνει υπ' όψιν άλλα δεδομένα, 'εσωτερικής πληροφόρησης', που μπορεί να έχει στη διάθεσή του ο μελλοντικός αντίπαλος. Για παράδειγμα, ένα μελλοντικό απολυταρχικό καθεστώς μπορεί να εκμεταλλευτεί δεδομένα των παρόχων πρόσβασης στο Internet, τα οποία έχουν συλλεχθεί στο παρόν, ή δεδομένα τα οποία είναι στη διάθεση των εκλογικών αρχών. Οπλισμένος με τέτοια πληροφορία ο μελλοντικός ισχυρός αντίπαλος μπορεί να καταργήσει την μυστικότητα σε όλα τα σχήματα που έχουν αποδειχθεί ασφαλή στο [Ara+13].

Στη διατριβή, ορίζουμε τρεις παραλλαγές για την αέναη ιδιωτικότητα, ανάλογα με τα είδη πρόσβασης που μπορεί να έχει ο αντίπαλος στα δεδομένα και την σχέση του με τον υπολογιστικά περιορισμένο αντίπαλο $\mathcal{A}$ της μυστικότητας. Η μοντελοποίησή τους γίνεται χρησιμοποιώντας κρυπτογραφικά παίγνια, κάτι που αποτελεί μια ακόμη συνεισφορά της διατριβής καθώς μέχρι τώρα η αέναη ιδιωτικότητα είχε τυποποιηθεί μόνο μέσω συμβολικών μοντέλων, όπως ο εφαρμοσμένος π-λογισμός [Ara+13].

- Στο πρώτο το οποίο εκφράζει την ασθενέστερη μορφή της ο ισχυρός αντίπαλος έχει πρόσβαση μόνο στα δημόσια δεδομένα των εκλογών - όσα δηλαδή βρίσκονται στο BB.
- Στην κανονική μορφή της ο ισχυρός αντίπαλος $\hat{\mathcal{A}}$ έχει πρόσβαση τόσο στα δημόσια δεδομένα, αλλά μπορεί να χρησιμοποιήσει και ψηφοφόρους που είχε υπό τον έλεγχό κατά τη διάρκεια των εκλογών ο $\mathcal{A}$. Αυτή η μορφή της αέναης ιδιωτικότητας μπορεί να θεωρηθεί ως επέκταση της μυστικότητας των εκλογών όταν ο αντίπαλος είναι υπολογιστικά απεριόριστος.
- Στην ισχυρή μορφή της, ο αντίπαλος μπορεί να έχει πρόσβαση επιπλέον και

στα κανάλια επικοινωνίας που χρησιμοποιήθηκαν κατά τη διενέργεια των εκλογών αλλά και σε δεδομένα που βρίσκονται υπό τον έλεγχο των αρχών.

Με βάση την παραπάνω μοντελοποίηση, αναλύουμε τα δύο είδη κρυπτογραφικών πρωτοκόλλων που έχουν προταθεί σε σχέση με την αέναη ιδιωτικότητα: Αποδεικνύουμε ότι το πρωτόκολλο των [FOO92; Ohk+99] που χρησιμοποιεί τυφλές υπογραφές ικανοποιεί τον ορισμό U-BPRIVκαι επίσης παρέχει ισχυρή αέναη ιδιωτικότητα αν χρησιμοποιηθεί ανώνυμο κανάλι για την κατάθεση των ψήφων. Αντίθετα, η ανάλυση του πρωτοκόλλου των [DGA12] που προσθέτει ένα σχήμα δέσμευσης στο Helios, παρέχει κανονική αέναη ιδιωτικότητα. Επεκτείνοντας το παραπάνω αποτέλεσμα, όλα τα σχήματα που χρησιμοποιούν σχήματα δέσμευσης με πληροφοριοθεωρητικά ισχυρή απόκρυψη, δεν μπορούν να πετύχουν ισχυρή αέναη ιδιωτικότητα καθώς τα ανοίγματα των δεσμεύσεων θα είναι στη διάθεση των αρχών και κατά συνέπεια διαθέσιμα στον ισχυρό αντίπαλο. Αντίθετα, τα σχήματα με ανώνυμη κατάθεση ψήφων φαίνεται να παρέχουν ισχυρότερες εγγυήσεις ιδιωτικότητας.

Από την άλλη πλευρά, μπορεί να τεθεί στην προσέγγιση μέσω της ανωνυμίας, η κριτική ότι δεν επιλύει πραγματικά το πρόβλημα, αλλά ότι το αντικαθιστά με κάποιο άλλο. Αντί δηλαδή να απαιτεί τέλεια μυστικότητα (μέσω σχημάτων δέσμευσης) απαιτεί τέλεια ανωνυμία. Η διαφορά, είναι ότι το δεύτερο είναι πιο εύκολο να επιτευχθεί με την έννοια ότι είναι πιο δύσκολο να είναι στην πλήρη κατοχή του αντιπάλου. Ένα ανώνυμο κανάλι, μπορεί να είναι κατανεμημένο και να λειτουργεί από μέρη τα οποία είναι δύσκολο να ελεγχθούν ακόμα και από έναν ισχυρό αντίπαλο - για παράδειγμα μπορεί να βρίσκονται σε διαφορετικά κράτη. Μπορούν να εφαρμοστούν και εναλλακτικές μέθοδοι ανωνυμίας στα άκρα, όπως για παράδειγμα υπογραφές δακτυλίου (ring signatures) ώστε οι ψηφοφόροι σε ένα εκλογικό τμήμα να μπορούν να σχηματίσουν ένα σύνολο ανωνυμίας. Κατά συνέπεια δεν χρειάζεται τέλεια ανωνυμία, αλλά τουλάχιστον ένα συστατικό της να διατηρηθεί εκτός του ελέγχου του αντιπάλου.

Τέλος, το μοντέλο μυστικότητας U-BPRIV που ορίσαμε στην εργασία μπορεί να χρησιμοποιηθεί για την έρευνα και αποτίμηση πρωτοκόλλων τα οποία δεν χρειάζονται εμπιστοσύνη στους καταμετρητές για την διατήρηση της μυστικότητας. Αυτό έρχεται σε αντίθεση με την πλειοψηφία των εργασιών στην βιβλιογραφία των ηλεκτρονικών ψηφοφοριών καθώς προτιμάται να υπάρχει εμπιστοσύνη στους καταμετρητές για επαληθευσιμότητα και όχι για μυστικότητα. Υπάρχει όμως αμφιβολία για το αν αυτή η υπόθεση είναι αποδεκτή από τους ψηφοφόρους. Το μοντέλο U-BPRIV μπορεί λοιπόν να χρησιμοποιηθεί ώστε να εξερευνηθεί περισσότερο αυτή η κατεύθυνση.

Τα κρυπτογραφικά εργαλεία και πρωτόκολλα που αναπτύχθηκαν στην παρούσα εργασία, δημιουργούν αρκετές ευκαιρίες για μελλοντική δουλειά. Ένας αρχικός

στόχος είναι η προσαρμογή του πρωτοκόλλου ηλεκτρονικών ψηφοφοριών σε αντίστοιχα περιβάλλοντα με τις ηλεκτρονικές ψηφοφορίες. Μάλιστα σε πολλά από αυτά, όπως για παράδειγμα σε περιβάλλοντα ανώνυμης συμπλήρωσης ερωτηματολογίων, οι απαιτήσεις ασφάλειας είναι πιο ελαστικές. Κατά συνέπεια, μπορεί να χρησιμοποιηθούν και οι υπογραφές CBS με αρκετή βελτίωση στην απόδοση. Επίσης, θα αναζητηθούν και άλλες υλοποιήσεις των CBS, PACBS με καλύτερη απόδοση, ιδιαίτερα σε ότι αφορά τον αριθμό των παράλληλων συνόδων λόγω της ανάλυσης των [PS00; Ben+20]. Μια άλλη κατεύθυνση, που έχει ήδη ξεκινήσει [PBS20] είναι η μεταφορά των PACBS σε αποκεντρωμένο περιβάλλον, μέσω υπογραφών δακτυλίου (ring signatures) διατηρώντας τη βασική συνεισφορά της διατριβής αυτής, ότι δηλαδή μπορεί να υπάρξει προστασία από εξαναγκασμό μέσω ιδιωτικής καταμέτρησης και αποδείξεων μηδενικής γνώσης. Η προστασία από εξαναγκασμό σε τέτοιου είδους εκλογές, αποτελεί προς το παρόν ανοικτό ερώτημα. Ένα τέτοιο πρωτόκολλο θα μπορούσε να εφαρμοστεί χρησιμοποιώντας μια αλυσίδα ομάδων συναλλαγών (blockchain) όπως αυτή εφαρμόζεται στα κρυπτονομίσματα όπως το bitcoin [Nak08]. Κάτι τέτοιο δεν είναι καθόλου απλό, καθώς όπως αναλύουμε στο [GP19] η ομοιότητα που υπάρχει μεταξύ της έννοιας της αλυσίδας συναλλαγών και του αποθετηρίου BB δεν αρκεί από μόνη της για να ικανοποιήσει τις πολύ απαιτητικές ιδιότητες ασφάλειας που χαρακτηρίζουν τις ψηφοφορίες. Με δεδομένο αυτό, όμως, η έρευνα προς αυτή την κατεύθυνση θα έχει το πλεονέκτημα ότι μπορεί να οδηγήσει σε νέα υποδείγματα ηλεκτρονικών ψηφοφοριών.

Αντιστοιχία όρων

Αγγλικός όρος	Ελληνικός όρος
public-key cryptosystem	κρυπτοσύστημα δημοσίου κλειδιού
multiplicative homomorphism	πολλαπλασιαστικός ομομορφισμός
additive homomorphism	αθροιστικός ομομορφισμός
digital signature	ψηφιακή υπογραφή
blind signature	τυφλή υπογραφή
designated verifier signature	υπογραφή καθορισμένου επαληθευτή
hash function	συνάρτηση σύνοψης
non-interactive zero knowledge proof	μη-αλληλεπιδραστική απόδειξη μηδενικής γνώσης
commitment scheme	σχήμα δέσμευσης
perfectly hiding	τέλεια απόκρυψη
perfectly binding	τέλεια δέσμευση
plaintext equivalence test	έλεγχος ισοδυναμίας μηνυμάτων
mixnet, shuffle	δίκτυο μίξης
ring signature	υπογραφή δακτυλίου
panic password	συνθηματικό πανικού
formal model	τυπικό μοντέλο
security game	κρυπτογραφικό παίγνιο
unforgeability	μη-πλαστογραφησιμότητα
strong one-more-forgery	ισχυρή μία-ακόμα-πλαστογράφηση
conditional verifiability	υπό-συνθήκη επαληθευσιμότητα
registration authority	αρχή εγγραφής
vote authorization	εξουσιοδότηση ψήφου
tallying authority	αρχή καταμέτρησης
bulletin board	αποθετήριο (ψήφων - μηνυμάτων)
blockchain	αλυσίδα ομάδων συναλλαγών
credential	διαπιστευτήριο
conditional blind signature	υπό συνθήκη τυφλή υπογραφή
publicly auditable conditional blind signature	δημόσια ελέγξιμη υπό-συνθήκη τυφλή υπογραφή
verifiability	επαληθευσιμότητα
cast-as-intended verifiability	επαλήθευση σωστής κατάθεσης πρόθεσης
recorded-as-cast verifiability	επαλήθευση σωστής καταγραφής κατάθεσης

tallied-as-recorded verifiability	επαλήθευση σωστής καταγραφής καταμέτρησης
software independence	ανεξαρτησία από το λογισμικό
individual verifiability	ατομική επαληθευσιμότητα
universal verifiability	καθολική επαληθευσιμότητα
eligibility verifiability	επαληθευσιμότητα δικαιώματος ψήφου
ballot secrecy	μυστικότητα ψήφου
receipt - freeness	μη αποδειξιμότητα ψήφου
coercion resistance	αντίσταση στον εξαναγκασμό
forced abstention attack	επίθεση αναγκαστικής αποχής
everlasting privacy	αέναη ιδιωτικότητα

Ευχαριστίες

Η παρούσα διατριβή ολοκληρώνει μια εννιάχρονη διαδρομή προσωπικής επιστροφής στον ακαδημαϊκό χώρο, μετά από δεκαετή απουσία. Σε αυτήν είχαν σημαντικό μερίδιο συμμετοχής μια σειρά ανθρώπων των οποίων η παρουσία τους ως συμβουλευτικά και εξεταστικά μέλη της διατριβής αυτής είναι ιδιαίτερη τιμή για μένα και τους οποίους οφείλω να ευχαριστήσω.

Κατ' αρχήν το μεγαλύτερο μερίδιο ευχαριστιών απευθύνεται στον επιβλέποντα καθηγητή της εργασίας κ. Αριστείδη Παγουρτζή για την οκτάχρονη και πλέον συνεργασία στα πλαίσια πρώτα του μεταπτυχιακού στο ΜΠΛΑ και στην συνέχεια κατά τη διάρκεια της διδακτορικής διατριβής. Η βοήθειά του ήταν καθοριστική, τόσο σε επιστημονικό επίπεδο, αφού ήταν αυτός που με ώθησε να ασχοληθώ με τις ηλεκτρονικές ψηφοφορίες δίνοντας έμφαση στην ιδιωτικότητα, όσο και σε προσωπικό επίπεδο, αποτελώντας ένα σημείο ισορροπίας απέναντι σε αντίρροπες δυνάμεις. Επίσης τον ευχαριστώ και για τις υπόλοιπες ευκαιρίες που μου έδωσε καθώς και για τη συμβολή του στην ολοκλήρωση της διατριβής.

Σημαντικές ευχαριστίες οφείλονται και στον κ. Στάθη Ζάχο, ο οποίος στην πρώτη μας συνάντηση, στην συνέντευξη για το ΜΠΛΑ είχε μία πολύ θετική αντίδραση στην υποψηφιότητά μου, και του οποίου ο τρόπος διδασκαλίας είναι παράδειγμα για μένα καθώς επίσης και για το ακαδημαϊκό περιβάλλον το οποίο έχει δημιουργήσει στο εργαστήριο το οποίο διατηρείται μέχρι σήμερα. Επιπλέον, θα ήθελα να ευχαριστήσω ιδιαίτερα τον κ. Δημήτρη Φωτάκη τόσο για την συμμετοχή του στην τριμελή επιτροπή επίβλεψης της διατριβής όσο και για το ιδιαίτερο ενδιαφέρον του και την άμεση βοήθεια που μου προσέφερε όποτε του την ζήτησα.

Ιδιαίτερες ευχαριστίες απευθύνονται στον κ. Παναγιώτη Τσανάκα, τον οποίο πρωτοσυνάντησα ως πρωτοετής φοιτητής στο Πανεπιστήμιο Πειραιά το 1996, και φυσικά στον κ. Άγγελο Κιαγιά, του οποίου το μάθημα στο Πανεπιστήμιο Αθηνών ήταν η πρώτη επαφή μου με την Θεωρητική Κρυπτογραφία και την επιστημονική της θεμελίωση. Με τιμά ιδιαίτερα επίσης η συμμετοχή των κ.κ. Συμβώνη, Πουλάκη, Ζήκα στην επταμελή επιτροπή εξέτασης της διατριβής μου, τους οποίους και ευχαριστώ.

Η διατριβή αυτή περιλαμβάνει αποτελέσματα τα οποία προέκυψαν στα πλαίσια συνεργασίας με τον κ. Αλέξανδρο Ζαχαράκη, τώρα υποψήφιο διδάκτορα στο Universitat

Pompeu Fabra. Τον ευχαριστώ ιδιαίτερα για την εποικοδομητική συνεργασία και τις πολλές συζητήσεις, που είχαν καθοριστική συμβολή στο αποτέλεσμα. Ιδιαίτερες ευχαριστίες και στον συν-συγγραφέα μας Bingsheng Zhang, για τη σημαντική βοήθεια που προσέφερε, καθώς επίσης και στους συναδέλφους από το CoReLab με τους οποίους συνεργάστηκαμε σε προτάσεις, διδασκαλίες και άλλα θέματα και ιδιαίτερα στους/στις κ.κ. Γιάννη Παπαϊωάννου, Πέτρο Ποτίκα, Pourandokht Behrouz, Μαριάννα Σπυράκου και Θωμά Σουλιώτη.

Τέλος, θα ήθελα να ευχαριστήσω την Αργυρώ και τη Λένα για την στήριξη και την τεράστια κατανόηση που έδειξαν για τις ατέλειωτες ώρες που αφιέρωσα σε αυτή τη διατριβή.

©2020, Παναγιώτης Μ. Γροντάς (Panagiotis M. Grontas)
Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται στον συγγραφέα (pgrontas@corelab.ntua.gr).

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Εθνικού Μετσόβιου Πολυτεχνείου.

Contents

List of Figures

List of Symbols

λ	security parameter
negl	A negligible function
n	number of voters
m	number of candidates
t	number of members of authorities
$[n]$	the set of integers $\{1, \cdots, n\}$
$:=$	Assignment or Definition
$\leftarrow$	Output of an algorithm
$\leftarrow_{\$}$	Selection of an element from a set uniformly at random
$\Leftarrow$	Append operation
$\mathcal{A}$, $\mathcal{B}$	Adversaries
$\Pi\langle\mathcal{A}(\alpha), \mathcal{B}(\beta), \gamma\rangle$	A protocol Π that requires interaction between two algorithms $\mathcal{A}, \mathcal{B}$ with secret input α, β respectively and public input γ
NIZK	Non-interactive Zero-Knowledge Proof of Knowledge
$\overline{\sigma}$	Signature
$\overline{\beta}$	Blind Signature
$V = \{\mathsf{V}_i\}_{i=1}^{n}$	The set of voters
$\mathtt{vt}$	Plaintext vote for a voter
$V_{\mathtt{Corr}}$	The set of corrupted voters
$V_{\mathtt{Coer}}$	The set of coerced voters
$V_{\mathtt{Hon}}$	The set of honest voters
$V_{\mathtt{Chck}}$	The set of voters who verify their votes
$V_{\mathtt{El}} = \{i\}_{i=1}^{n}$	set of identities of all eligible voters
i	Index & identity of voter
BB	Bulletin Board
EA	Election Authority
RA	Registration Authority
TA	Tallying Authority

Due to the extent of this thesis, we were forced to reuse various symbols. Each such symbol has a local scope. Encrypted values are denoted in ***bold***. Tuples are denoted with $\vec{\cdot}$.

1 Introduction

> The medium is the message
>
> Marshall McLuhan

Voting is a distributed decision-making process, where a set of agents select an option and reach consensus on the most preferred input. The result is binding for the group, as everyone is expected to adhere to it until the process is repeated. The algorithm to implement the basic process is remarkably simple; it involves two elementary operations - comparisons and counting. Each input is compared to the available options and the respective counter is incremented. To deal with scaling issues many technologies have been employed for assisting the realization of the process: from pebbles and paper to lever machines and electronic computers. It is usually the case that technologies affect deeply, to the point of total transformation, the process they are assisting.

The simplicity of voting has cast it as a ubiquitous method to reconcile differing opinions. Its use ranges from informally deciding where to hold simple gatherings to selecting a national government. In all settings, the fact that the vote result is binding, can motivate participants to influence the process to their benefit. Human 'ingenuity' has come up with many methods to achieve this. They can try to convince the voters for their favored opinion before the 'election' begins by using (logically sound) arguments or by spreading misinformation. If this fails, they can step up their game by coercing the agents using threats or countermeasures. They can even try to affect the process that computes the results itself, by changing the inputs in an unauthorized manner or by tampering with the internals of the computations. In some cases, there is no need to carry out an attack; the fact that there is a perceptible danger of interference casts doubt to the legitimacy of the results. As a result, the voting process must be augmented with another goal: to compute the results in a manner that convinces every participant and especially the 'losers', that the elections were conducted according to the rules and their results indeed expresses the majority of the agents.

It is evident that since technology is used to realize the voting process, technology can also be used to carry out the attacks. But technology can also be used as a means

of defense in a twofold manner: by providing specific countermeasures and more importantly by analyzing and modeling the process thereby contributing to its better understanding. In electronic voting, the goal of any assisting technology must be to eliminate (or greatly reduce) trust in particular agents or system components. This results in making attacks more difficult as the possible targets have diminishing importance. Even more importantly, reducing the trust required to accept the results will reduce the perception of threat, which impacts the legitimacy of the results.

This thesis proposes cryptographic primitives and protocols that can be used to secure the voting process. As one of the goals of modern cryptography is to reduce trust, it provides a great match for electronic voting.

1.1 Election technologies

We begin by reviewing the most important technologies that have been employed to implement elections. We use the term technology in broad terms; it does not refer only to modern computer systems. A piece of paper, a wooden ballot box, a voting booth separated by a curtain are all examples of technologies.

The simplest and probably one of the earliest methods of voting works by requiring *a show of hands* from those who agree with the proposition put forth. Tallying takes place by simply counting raised hands in the presence of all voters. This method, while technologically primitive, has an important advantage; it requires no trust in the tallying algorithm, as all is needed is the snapshot of raised hands. As a result, everybody can verify that the result is correct, by *self-tallying*. This makes the show of hands the definitive template for trustless voting systems, as they provide a central, immutable and publicly available repository of cast votes that everybody can use to compute the result on their own. On the other hand, such systems are not easy to use because of scaling issues. They cannot be used in large numbers, as it becomes more difficult to capture the snapshot of votes. Of course, they are also prone to errors. However, the most important problem with this voting method is the same as its advantage: The fact the everyone can check how everyone voted; this can help coercers to actively force voters to select a particular option, by threatening them with retributions if they don't. The reason behind this, is simply that the coercers can more easily monitor the process and check if their targets complied with their directions and if not, carry out their threats.

The secret ballot The most important technological innovation in voting is the introduction of the Australian or 'secret ballot' in the 1850s [Ben13], that aims to counter the implication of revealing one's vote. In such systems the voters mark

their preferences on an anonymous token; nowadays a piece of paper is used where the choices (candidate names) are preprinted. Ballot casting takes place inside an isolated *voting booth*, where the voter is free to cast their vote from the prying eyes of coercers. Additionally, the voting booth is a technology that protects elections from malicious voters, as it prevents the latter from selling their votes and producing relevant receipts to receive their rewards. After casting, the voters of a secret ballot system create a ballot by enclosing their vote in an unmarked envelope and posting it on a sealed box named *ballot box*. The envelope-ballot box combination makes votes both *anonymous*, unlinkable to a real-world identity, as well as *secret*. After the voting period ends the talliers unseal the ballot box, open the envelopes and count the votes.

Supervised voting In order to scale the secret ballot, to millions of users, an infrastructure is required. First of all the voters must go through a registration phase that serves two purposes: to check if the voters are eligible to participate in the elections and, if this is the case, to receive an authentication token. These tokens can be used for single or multiple elections and in some cases even as all-purpose identities. During the actual elections these tokens are used to authenticate the voters, check eligibility and prevent double voting. However, the credentials are not included in the ballot. Authentication and tallying is performed by a group of individuals with conflicting interests, typically representatives of different voting options, supervised by a *trusted third party*. The voters are also supervised to adhere to the rules of procedure, e.g to enter the voting booth on their own, not to leave special marks on their ballot. The voting infrastructure is distributed in a hierarchical scheme to reach millions of voters; the votes are collected and counted at the edges (tree leaves), where each voting precinct computes and announces its tallies. The partial results are then merged to create the final tally. Despite this distribution of labor, vote counting is a time consuming and error-prone process especially in voting systems with complex selection rules. In order to improve tallying times, the paper ballot has been replaced with mechanical lever machines in large jurisdictions. This supervised infrastructure must be set up and dismantled after every election, which incurs a large monetary cost.

The threat model against supervised voting rests on two assumptions. To begin with, an adversary needs to corrupt many partial tallies to affect the results. The same applies to an adversary aiming to disrupt the elections. The distributed hierarchical organization makes the scheme resilient. Furthermore, within a voting center, the separation of roles to agents with conflicting interests creates self-enforcing motives for each player to monitor the others. Additionally, such a voting system cannot operate on its own but requires trusted third parties to monitor the process and enforce

the rules. Many [Ben13; Wil17; KWV19] have noted, that the threat model of supervised voting is safe against an adversary with limited technological capabilities. But this might not be the case anymore. The proliferation of cheap 'smart devices', everyday devices (like glasses and watches) equipped with microprocessors and I/O sensors with network connectivity modules fundamentally changes the capabilities of the attackers.

Remote (physical) voting In order to increase participation, many jurisdictions have enabled remote voting, usually by employing the postal system (vote by mail). The voters receive special ballots by mail, mark their choices and re-post their ballots or deliver them directly to a voting center. During counting the ballots are stripped of identifying information and tallied, sometimes using software components. Again, counting is performed by agents with conflicting interests supervised by trusted third parties, i.e. the back end remains the same. However, the voter experience is fundamentally different, as there is no supervision or privacy enforcing voting booths. As a result, vote-selling and coercion becomes easier. Note that the attacker in these threats is not an abstract formal adversary; these threats can easily be performed by family members, employers, and associates. Vote by mail has been initially deployed to serve a minority of voters whose employment required frequent mandatory absence from elections (e.g. armed forces, diplomatic personnel). This meant that the percentage of votes cast remotely was a negligible percentage of the electorate, unable to affect the result. However, as the mobility of the modern workforce increases, the application of vote by mail increases as well, exacerbating the problems of vote by mail.

Electronic voting With the proliferation of computers, the Internet and the digitization of all processes, it was expected that voting would follow suit. However, this transition did not proceed with the speed and scale that was typical of other activities e.g. banking. Many researchers [Sch] claim that voting has some unique security properties that cannot be satisfied by computer systems, because of their inherent characteristics and must, therefore, remain (partly at least) paper-based.

The first way to integrate computers into electronic voting, replaces the paper ballot and the talliers with computers while maintaining the rest of the infrastructure in supervised procedures. This means that both the ballot and the ballot box are electronic. The voting booth remains physical and votes are cast using touch screens or other input devices on *DRE (Direct Recording Electronic)* machines. Receipts can be optionally printed and cast independently in ballot boxes, thus creating a paper trail. Another alternative allows voters to mark their votes on paper ballots and digitize them using optical scan machines. In both cases the votes are electronically stored

and transferred to a computer functioning as a tabulator to produce the result. However, reusing the existing infrastructure with computers instead of human talliers is not the only options. Information technology can transform the voting process by enabling different or novel voting methods to be applied, that would be prohibitive otherwise, because of scaling issues and other physical limitations.

The discussion around electronic voting usually revolves around the casting and tallying phases. However, a nationwide supervised system has many more components that can be digitized. For instance, in order to obtain registration information and create the voter rolls, public databases of voter information can be used which are often in electronic format. A simple way to affect the result of an election is to alter the contents of these databases and exclude legitimate voters are add fake ones. Furthermore, computers are also used after computing the local results at the voting precincts, to transmit the partial tallies to the central authority for aggregation. The vulnerabilities of using software also apply to these subsystems of electronic voting as well.

In order to understand the problems with electronic voting, one must note that voting with computers is essentially, *voting by proxy*. As a result, voters delegate their choices to third parties and trust them to record, transmit and count them correctly. However, this trust is misguided as a computer might change votes simply because an error occurred or even due to malice. The lack of trust is exacerbated by the universal nature of computers and software. For instance, the computer could run variations of the recording and tallying programs that correlate the order of the votes with the order of authentication, thus linking voter identity with vote contents or randomly alter votes. Proposed solutions such as open-source software and certification are necessary but not sufficient conditions, as one cannot be sure whether the recording or tallying machine runs the code that was examined.

[Riv08] observes that the result of an election should not depend on the computer systems that is used to run it, so that software problems do not propagate to the election results and defines the notion of software independence to characterize a major desideratum of electronic elections:

Definition 1.1: Software independence

A voting system is (weakly) software-independent if an undetected change or error in its software cannot cause an undetectable change or error in the election outcome.
A voting system is *strongly* software-independent if it is software-independent and a change or error in its software can be detected and recovered so that rerunning the election is not required.

A simple way to achieve software independence is by utilizing the paper trail that some DRE machines leave and allowing the voter to verify it. The resulting system is software-independent, because the software can be replaced by a hand count, in the case the official results show discrepancies, or after a manual audit of a small part of the ballots using statistical audit techniques, like Risk Limiting Audits [LS12]. In fact, experts argue [Ber+17], that currently this is the best way to vote, assuming that certain rules are followed for the handling of paper ballots. While we agree that this is currently the best way, it is tied to voting using the traditional supervised infrastructure. Another way to achieve software independence is to use cryptographic methods. This thesis is built around the belief that cryptographic voting can (ultimately) be as secure as traditional supervised voting.

Remote electronic voting The electronic analogue of vote-by-mail is Internet voting. The voters cast the ballot through their Internet-connected personal computers or smartphones and the vote is relayed to the tallying computer through the computer network. Unlike supervised voting, the voters can cast the ballot in the comfort of their homes or their offices without any official being present. The software used must be platform-independent, so it usually takes place through a web browser.

Internet voting has many advantages: Firstly, it allows people to make their choices without having to physically visit the polling station, which usually mean waiting in long queues. It can also be accessible from any part of the world, allowing greater turnout and replacing the need for vote by mail. It can also be more usable, as the voting software can provide guidance over the process. This is especially true for voters with disabilities. As it is implemented in computer software it has significant cost savings in the long run. The administrative costs of setting up an election are also quite low, leading to frequent impromptu elections. The proponents of Internet voting have even gone as far as claiming that it will bring back the direct democracy witnessed in classic Athens 2500 years ago.

On the other hand, Internet voting has many disadvantages. The most important one is that it sacrifices the mandatory imposition of privacy offered in voting booths. Since voting is done without official supervision, it is open to coercion and vote selling as the attacker can be present and oversee the voting process. Moreover, Internet voting enables a weaker kind of coercion to take place: pressure from family or close friends, as the voter is never alone. These are dangers present in all types of remote voting. Another important problem is that its security depends on the security of the voters' computer. The security research literature is full of vulnerabilities and attacks leading the safe conclusion that computer client software cannot be trusted for security. Malicious software might alter votes, fail to submit them (without reporting

it) and many other things can go terribly wrong. Things are not as bright on the server side as well. As the voting server resides on an open network it is a target of attacks as well, that are not restricted inside a country, but can come from external opponents as well. Furthermore, the software used for voting might be inaccessible to certain groups of people such as the elderly as they are not comfortable with using a computer. We can see why many computer security and voting researchers think that Internet voting is a bad idea [Sch].

There are indeed many hurdles that one must overcome and applying internet voting to nationwide governmental elections seems terrifying. However, electronic voting in a network setting can transform collective decision making and create scenarios that would be impossible to implement at scale, otherwise. For instance, in [KY02], a new type of voting system is proposed: *self-tallying elections* where everybody (including outsiders) can count everybody's vote. Such a scenario would only be feasible on a small time and space scale. To sum up, despite the fact the electronic voting is not secure yet to implement nationwide governmental elections it can be applied for issues of less importance and enable of new types of decision making. This is the motivation behind this thesis.

1.2 Security properties of voting systems

The characteristics of computers that make electronic voting a hard problem to solve, require a rigorous security analysis of any proposed e-voting scheme, beyond what is standard in any software engineering problem. This process checks that the scheme conforms to a set of security properties that any voting scheme must possess, either electronic or physical. This makes electronic voting not merely a digital analogue for traditional elections, but a somewhat improved version brought about by formally defining, analyzing and seeking to satisfy difficult and conflicting security properties. In this section, we provide a high-level overview of the security characteristics of any voting system. In chapter 4 we will provide a formal analysis of these properties.

1.2.1 Verifiability

The main properties grouped under this section deal with the correctness of the result of the election and the conviction of voters around it.

Correctness/Integrity The voting system must correctly compute the tally, by incorporating the choices of the voters into the results. This must take place irrespective of the result function. However, this is not enough; elections shareholders must be convinced that the announced result is correct.

Verifiability This property aims to *assure* candidates and voters that all votes have been considered. As a result, verifiability requires evidence that allow the voters to verify that the output of the tally function corresponds to their collective desires. Voters must check this evidence in order to make sure that their votes [Ben+15] were:

- **Cast as intended**: The user interface of the voting system should aid the voter to express his choice without interfering to change it and help the voter verify that it was correctly captured. As a result, there should be no doubt about what the voter input to the voting system was and how it is being stored. One way to achieve this is the cast-or-audit mechanism proposed in [Ben06]; the voter creates an arbitrary number ballots, but eventually casts only one and audits the rest. The election system does not know which ones will be eventually audited so it is motivated to follow the protocol on all.
- **Recorded as cast**: The voting system should convince its users that their recorded votes have been correctly transferred to the counting stage, by enabling voters to *pinpoint* their own ballot in a list of ready-to-be-tallied ballots. This is usually done by providing voters with receipts to be compared with the list of cast ballots.
- **Tallied as recorded**: Voters and all interested parties (e.g. external observers and auditors, pro-democracy organizations) should be convinced that all valid votes have been included in the final tally.

These checks together constitute the security requirement of *End-To-End (E2E) Verifiability* . Cast as intended and recorded as cast verifiability are usually collectively referred as *individual verifiability*, since the voter herself initiates the checks. Tallied as recorded checks are performed by everybody so they are collectively referred to as *universal verifiability*.

Another type of verifiability check is *eligibility verifiability* , where everybody must be able to verify that all the votes coming from legitimate voters (i.e. that have the right to vote) have been included in the tally. This is part of a tallied as recorded check and aims to prevent ballot stuffing and double voting. Eligibility verifiability requires a way to provide publicly auditable credentials to voters that allow for the eligible voters to stand out. To receive them the voters must *authenticate* themselves to a voting system. However, authentication requires some form of identification and this might contrast the ballot secrecy requirement. To go around this limitation, the voter cannot be individually identified, but instead associated with a set of specific characteristics, which must be the largest possible for each participant (*anonymity set*). The next step is *authorization* which makes sure that the participant behaves as

expected so as not cheat the system. For instance, a voter should vote as many times (usually one) as the election law mandates and no more.

In traditional supervised systems, all the checks that are required to satisfy the verifiability requirements are delegated to trusted third parties observed by representatives of all shareholders that have conflicting interests. The individual voter cannot verify the process and the result herself but delegates these tasks. This type of verifiability is called *administrative verifiability* . Typical electronic voting systems try to do away with this type of verifiability. However, this is not always possible, because voters must actively participate in the auditing both for their own votes as well as for votes cast by others. In both cases this is not something that can be expected from them, as research shows [KZZ15b; Ber+17].

Its close relationship with the integrity of the election process, its perception, and the acceptance of its output, makes verifiability a very important property, extensively studied [Cor+16] and implemented in many protocols under computational assumptions or unconditionally.

Accountability This property is a stronger form of verifiability [KTV10]. Instead of simply producing evidence that something went wrong with an election (phase), accountability seems to pinpoint the perpetrator of this malicious behavior. This can be used as a counter-motive and force all players to execute the protocols honestly. Accountability is a property that is extremely hard to get right and As a result, only a few voting systems have been designed to possess it [Küs+16]. Accountability is sometimes described as collection accountability and dispute resolution [Ber+17].

1.2.2 Confidentiality

Privacy Since the invention of the secret ballot, the protection of its contents has been encoded into law for all democracies, as a means to guarantee that the voter indeed expresses her free will. Vote privacy aims to hide the choice of a voter from the talliers, other voters, or external agents in order to free her from external pressure and enable her to cast a ballot that represents her true choice.

It is particularly important to note that privacy is not absolute, as the election result leaks information. For instance, one can infer that the probability of one voter having cast a particular ballot is close to the percentage of total votes this particular choice has received. If there is a partial (local) tally this estimation is more accurate. In extreme cases, one's preferences might be completely revealed. For instanced in a unanimous result, everybody knows how everybody voted. If an election result is

unanimous except for one vote, then this particular voter knows how everybody else voted.

Furthermore, it is easy to observe that the secret ballot, hinders verifiability. The most verifiable voting scheme is the 'show of hands' we described in section 1.1 where all aspects of elections are publicly auditable. By hiding the contents of the ballots, we also hinder the transparency of casting and tallying [Ber+17]. However, focusing solely on verifiability without privacy makes no sense. If one assumes that the contents of all votes are publicly known and linked to individuals, as in the case of a show of hands, then they can in effect be dictated by external agents applying emotional, personal, social and economic pressures. As a result, one cannot be sure that a vote represents the true will of a voter, as the voter could have yielded to these external forces. Thus, the vote cast would not be the one that was intended. In that sense, it would not differ that much from a vote altered by a malicious entity, as is the case with the verifiability threat model. The converse can also be shown at least for individual verifiability [CL18]: By auditing ballots one can prevent malicious players from changing ballots, by avoiding the attack where every vote is changed except for one targeted vote. Then applying the tallying function can reveal its contents.

Vote privacy has been studied in many variations, concerning the capabilities of an adversary, its relation to the voters and its intended duration.

Ballot secrecy A first layer of privacy protections aims to guard against passive adversaries that want to learn the behavior of a particular voter (subset). This has been implemented in two ways: by hiding the contents of the vote or by disassociating the voter identity from the ballot. The former is usually achieved using a threshold cryptosystem with homomorphic properties, while for the latter an anonymity primitive such as mixnets [Cha81] or blind signatures [Cha83] is applied (cf. subsection 2.5.1 and section 2.7). The actual level of privacy offered depends on the implementation, which usually rests on *computational* and *trust* assumptions, as it is generally assumed that there will be an honest subset of participants that will follow the protocol and not try to break the secrecy. This means that they will refrain from opening individual votes but will decrypt only the result of the final stage. Blind signatures, on the other hand, can offer information-theoretic protection. The maximal protection of the choices of the voters is the concept of *Perfect ballot secrecy* [KY02] proposed in the context of self-tallying boardroom voting schemes, which guarantees that knowledge about the partial tally of a subset of the voters can be computed only by a coalition of all the remaining voters. This notion provides vote secrecy regardless of trust assumptions on the talliers' honest behavior.

Receipt-Freeness This stronger form of privacy has been proposed by [BT94] protects the voters against 'themselves', providing privacy even if the voter does not wish for it. To be receipt-free a voting system should not provide the voter with a receipt that indicates how she voted, because such a receipt could be utilized as a proof if the (malicious) voter wants to sell her vote. Its absence means that the potential vote buyer will not be able to be convinced that his money was well spent. As a result, receipt-freeness discourages vote selling. However, the conflict with verifiability reappears; a generated receipt can function as evidence that allows the voter to verify the election system and vice versa such evidence can be used as a receipt. Receipt-freeness (and privacy) does not apply solely to electronic voting schemes. Technological advances such as camera-equipped glasses [Ben13] or audio side channels [Wil17], allow a voter to effectively sell her vote and record the 'transaction' to be traded as a receipt. In general in [Che+10] it is proved that a voting system cannot achieve simultaneously universal verifiability and receipt-freeness, unless there are private channels between the voters and the election authorities. Such private channels allow the voter to validate the receipt privately and deniably, in a manner that casts doubt to the probable vote buyer, while allowing her to verify the vote.

Coercion resistance Another serious threat to voting schemes is an active adversary that constantly monitors a voter. His aim is to fully to dictate the voter's behavior to his wishes with the goal to make the voter abstain or to vote randomly or to fully impersonate her. To defend against such attacks, voting systems should possess a property called *(over the shoulder) coercion resistance*. The general method to achieve this property is to cast doubt on the coercer about the success of his attack, enabled by the voter applying a deception strategy. For instance, in many voting systems the voter is allowed to re-vote [MM06; Adi08; Tso+13; LQAT20], so the voter can obey when the coercer is present, but with a later vote can undo her previous choice. The re-voting technique has been augmented and generalized in the Juels Catalano Jakobsson coercion resistance framework [JCJ05], where the voter can vote multiple times using anonymous credentials each time. One credential is registered as authentic in a manner invisible to the coercer (e.g. during in-person registration), while the rest of the credentials are considered fake and therefore the votes that accompany them do not count. As a result, the coercer cannot be sure if his attack succeeded or not, because he cannot tell if which credential is used, even if he monitors the voter for the entire voting period. Fake credential-based schemes make assumptions about the voter having a moment of privacy and that the registration is untappable (more details in section 4.4). Furthermore, in order to apply the credentials the voter must have a token that can perform cryptographic operations with Selections[UH12] being the most notable exception. On the other hand, re-voting - based schemes, require a

moment of privacy too, but make no assumptions about the presence of the adversary during registration [LQAT20]. However, they make the strong and rather inflexible assumption that the coercer can monitor the voter until the end of the voting period. This can be achieved if the coercer is human, but it is more problematic in the case of the coercer being a software program.

In any case, schemes that are based on multiple votes-per-voter (either through re-voting or through fake captured), are yet not allowed in many jurisdictions and there have been reports that users will have trouble understanding this functionality [Wil17]. Note that coercion resistance, largely depends on the voting method used. A voting system that supports *write-ins* cannot be coercion-resistant as the coercer can force the voter to include a vote for a random string and check that this random string is included in the results. Furthermore, in schemes that voter selects or ranks a subset of k out of m candidates the *Italian attack* can be mounted; the coercer can dictate that a particular permutation of candidates appears in the results, in effect watermarking each ballot with a specific pattern per voter. If such a permutation is not found, then it is evident that his directions were not followed,and repercussions can follow. To counter the Italian attack a *short ballot* assumption is sometimes used [KTV12a].

Everlasting privacy [1] The variation of privacy, where the adversary is computationally unbounded is called *everlasting privacy*. Its study, formally initiated by Moran and Naor in [MN06], focuses on preventing secrecy attacks by powerful future adversaries. It is motivated by the observation that in most cases, vote privacy is only protected by a cryptosystem the security of which is based on computational assumptions such as the intractability of the Diffie-Hellman problem (cf. subsection 2.1.1). These assumptions, however, may be broken or rendered obsolete in the (not too) distant future, as both the theory and the practice of cryptographic attacks always get better. This means that votes encrypted with small keys are in danger of being revealed, even without the computational assumption being broken. As famously conjectured by Shamir, at the 2006 RSA Conference cryptographers' panel, all cryptographic keys used at that time would remain secure for less than thirty years (cf. [MN06]).

The situation is made worse because verifiability requires utilizing public evidence generated by the election system. These pieces of data are meant to be widely available and thus it is easy for an adversary to obtain them, even in part. However, one must bear in mind that the adversaries against voting systems are potentially powerful state agencies with enormous budgets and without time constraints. As a result, they have the capability to collect and store large amounts of election-related data.

[1] Based on [GPZ19]

Furthermore, as large-scale elections are organized by the government, these agencies can be considered 'insiders', having access to even private parts of the election transcript. Finally, these agencies can obtain information exchanged through computer and communication networks, both through mass surveillance as well as with the cooperation of telecommunication companies.

The problem of privacy is exacerbated, as the information concealed in voting does not lose its value, contrary to protected messages in other common cryptographic scenarios. Indeed, one can easily imagine a future authoritarian regime that tries to gather evidence about its subjects based on past democratic elections in cooperation with the state intelligence agency. This evidence might prompt actions ranging from surveillance to questioning and even more severe repercussions. As noted in [MN06], such dangers constitute an indirect coercion attempt. In fact, since there are many potential coercers the only rational reaction from a voter fearing all possible adverse scenarios is to abstain. Everlasting privacy seeks to protect the secrecy of individual votes in such scenarios.

Finally, a recent property the is in the crossroads of verifiability and privacy is *participation privacy* [Cor+14; KTV15]. In many jurisdictions, it is illegal to reveal if a voter abstained or really voted. This is incompatible with eligibility verifiability, where voter pseudonyms, that are linked to real-world identities accompany the votes. It can also be considered a form of privacy, should we consider that abstention is a special candidate to be chosen. The combination of eligibility verifiability with participation privacy is called *private eligibility verifiability*.

1.2.3 Other properties

Except for variations of integrity and secrecy a voting system must satisfy other important security properties. We list some of them in this section. Note that the fact that less space is devoted to them, has nothing to do with their importance; it is related to the focus of this thesis.

Fairness A voting system must not produce early or partial results, as such could affect voters who have not yet cast a vote. The tally must be announced simultaneously to all voters. Fairness also implies that elections should not be repeated, as the previous election sets a precedent, even if no result is announced. This means that the voting system must be robust and that from the moment the election starts it must be concluded. This is especially important for self-tallying schemes where - in a simple implementation - the last voter might be able to know the partial result before casting her vote.

Enfranchisement All entities that have a stake in the result of an election should participate. This is easier said than done. For example, in order to be authenticated, voters should issue some form of credentials. These excludes the people that cannot issue these credentials for some reason. In addition, voters should not be intimidated by the voting system and be discouraged to participate. This is particularly true for electronic voting, since technologically illiterate people might find it hard to use the system. Cryptographic voting exacerbates the problem, since instead of general ICT skills voters should now possess (general) knowledge of cryptography.

Enabling enfranchisement should not contradict properties such integrity or privacy. Assistance in voting for example might help some voters, but ballot secrecy must be maintained. Enfranchisement implies that voters trust the system to compute their intent. Complex voting systems might succeed in implementing some or most of the requirements but might fail in convincing the voters. We do not address, the question if the solutions described in this thesis, actually convince voters.

Resiliency and Efficiency The voting system should always be available to receive input from the participants and it must output the result of the computation in a reasonable time. Availability is crucial since the repetition of the vote casting phase sheds doubts on fairness. Availability in the presence of an active and persistent adversary is often called *robustness*. The adversary might pose as a voter or as the authority or both. In addition, lack of availability hinders enfranchisement, since a voting system that has 'ups and downs' in its operation, is perceived as untrustworthy. It is important to point out that a voting system must be robust in its complete lifecycle and not only on the vote casting stage. For example, a verifiable voting system must have a detailed process to deal with 'alleged' verification problems, or else verifiability can be used against the system.

Efficiency is one of the reasons put forth by electronic voting proponents, especially when the voting population is extremely large. It refers to the resources used by the voting system in all its workflow. Such resources are referred to as *cost* and might be time, money, requirements on infrastructure and people participation etc.

1.3 Contribution

This thesis builds on the basis that remote electronic voting using cryptographic methods for software independence is a goal worthy to pursue. Despite, that we are far from realizing this goal yet for national elections, voting is a ubiquitous process that can benefit from such innovations, starting from smaller-scale elections. To this end, we make three novel contributions:

- A cryptographic primitive called *Publicly Auditable Conditional Blind Signatures (PACBS)*.
- The first-ever game-based definition of the property of everlasting privacy.
- A voting scheme that provides coercion resistance and everlasting privacy without sacrificing verifiability.

Other minor contributions are a comprehensive review of all the formal models provided in the literature for the security properties of electronic voting systems. We express these models against a generic voting system that encompasses all functionalities defined so far. The relations between these properties, as made evident from their formal definitions, are also explored.

We now analyze the major contributions in more detail.

1.3.1 Publicly Auditable Conditional Blind Signatures

We define Publicly Auditable Conditional Blind Signatures, a standalone signature scheme that connects the validity of a signature with a predicate applied on publicly available data. More specifically, instead of creating a digital signature by solely applying a function parameterized with a secret key to a message, publicly available data is embedded in a way that the signature is valid if and only if a predicate on them is satisfied (and the proper keys have been used, of course). In this sense, the validity of the signature is *conditional*, which means that the signature can be thought of as carrying an extra bit of information that determines if it verifies correctly or not. Furthermore, the signature verification is done by a designated verifier, recognized by the possession of a secret verification key. These specifications are also captured in *Conditional Blind Signatures (CBS)*, a precursor of PACBS which is also presented. The disadvantage of CBS is that a malicious signer or a malicious verifier could disregard the predicate and the related public data and provide arbitrary signatures and verification results. To defend against this weakness of CBS, PACBS include signature creation and verification *audit* functions. They produce evidence that can be checked by anyone to verify that the predicate was correctly computed, embedded and checked. PACBS supports blindness to increase the privacy of the user. A security model of PACBS is also defined that reflects its desired properties. To this end, blindness and unforgeability - the standard properties of blind signatures - are complemented with a new property, *conditional verifiability*, that incorporates the predicate to the validation procedure. Moreover, a construction is provided based on the Okamoto-Schnorr blind signatures [Oka92]. Finally, security proofs are presented that reduce the security of PACBS to well-known cryptographic assumptions.

1.3.2 Everlasting privacy

We also propose the first game-based definitions for everlasting privacy. Our definitions are generic, which means that they do not consider the cryptographic primitives that will be used in order to achieve this property. This has not been the case so far, where everlasting privacy was defined only in the symbolic model of security.

More specifically, we consider the adversarial capabilities in terms of both data collection and computational power. To model this, we assume two adversaries: The first is contemporary to the election, where he can participate actively (using corrupted voters) and passively (by monitoring communications between the voters and the authorities). He is computationally bounded, though. The second adversary is computationally unbounded but operates (long) after the election is over. The two adversaries can communicate and As a result, the future adversary can obtain election transcripts and auxiliary information collected in the present from corrupted entities such as voters or even insiders to the election systems.

The motivation for this capability stems from the reasonable assumption that there exist powerful entities (e.g. governmental agencies) that might passively hoard election-related data such as protocol and communication transcripts (among other things as demonstrated by mass surveillance revelations such as Snowden's). It is realistic to assume that a future totalitarian regime will also take control of these agencies as well and have access to their collected data.

By elaborating on the communication options between the present and the future adversary we define three variations of everlasting privacy:

- *weak* everlasting privacy: There is no communication between the present and the future. As a result, the authoritarian regime can only access generally available data, such as public bulletin boards. In effect, this scenario is the same as the one contemplated in the definition of *practical everlasting privacy* of [Ara+13].
- everlasting privacy: The future adversary can only take advantage of corrupted external agents to the election system. In reality, these agents are the present cooperators of the oppressor that passively collect or take advantage of current attacks to extract information in the future. This model can be viewed as a direct extension of normal privacy against a more powerful adversary
- *strong* everlasting privacy: There is full communication between the present and the future. This means that the totalitarian regime has insider access to private election data gathered by agencies 'internal' to the election. These include election authorities, telecommunication providers, political parties etc.

This model is motivated by the observation that usually power changes, bring control changes for such agencies.

We express these adversarial models against a generic voting scheme to provide game-based definitions of everlasting privacy. Our definitions are the first-ever in the computational model, as the work of [Ara+13] is based on the symbolic setting. We discuss the implications of our definitions and observe that perfectly hiding commitment schemes do not offer the same levels of protection as anonymous channels, since they cannot hide auxiliary communication information, that can be utilized by a powerful future adversary with insider information. Our approach has the added side effect that it associates everlasting privacy with contemporary privacy, which is a relation that, to the best of our knowledge, has not been explored in the literature.

1.3.3 PACBS Voting

We propose a a voting protocol based on the architecture of FOO [FOO92], one of the most privacy-aware voting schemes in the literature, augmented with an efficient implementation of the coercion resistance properties of JCJ [JCJ05]. In particular, we take advantage of the fact that in [FOO92], voting occurs in two phases, namely authorization and counting, and use it to overcome the performance bottleneck of JCJ. We achieve this by using the idea of [GPZ17], i.e. marking the fake credentials during the authorization phase where voter identification is available. By using the voter ID the correct credential can be efficiently retrieved and compared to the supplied one with no need to check all credentials. Of course, during this phase the ballot contents must be blinded, as they can be correlated with the voter ID. The fact that the credential is invalid is conveyed to the counting phase by applying PACBS. The counter receives the ballot and authorization in the form of a blind signature, that contains a bit that specifies if the vote is valid or under coercion. The perfect blindness property of the CBS scheme combined with an anonymous channel enable us to achieve the everlasting privacy property, without residing to dedicated channels between the authorities. Our protocol achieves verifiability, coercion resistance and everlasting privacy with minimal assumptions.

1.4 Thesis structure

This thesis' topic is how techniques from cryptography can be used to build electronic voting systems that implement all the features described in this chapter, in a non-conflicting manner. We introduce approaches to electronic voting by continuously adding layers one on top of the other. A road map follows:

- In chapter 2 we introduce the cryptographic building blocks that are utilized in the building of PACBS and subsequently in the proposed voting scheme. We cover basic notions such as computational security assumptions, public-key encryption schemes, digital signatures, and their variations as well as zero-knowledge proofs and verifiable shuffles. We examine such primitives under the lens of voting systems focusing on how they can affect the security of elections. We review security models and instantiations that we employ in our further constructions.
- In chapter 3 we introduce Publicly Auditable Conditional Blind Signatures (PACBS) which is the novel cryptographic scheme proposed in this thesis. Our exposition is based on its evolution from a first version, Conditional Blind Signatures (CBS), that clearly shows the basic problem this primitive intends to solve but has certain weaknesses. We define a security model and provide an instantiation and variations. Then we refine the security models and constructions in order to resolve the problems of CBS. Thus we arrive to PACBS.
- In chapter 4 we focus on formal security models for the most important properties of voting systems, namely verifiability, privacy, coercion resistance and everlasting privacy. Each model is accompanied with example voting systems that aim to justify it. In this chapter we also introduce our game-based definitions for everlasting privacy.
- In chapter 5 we describe the voting system that is built around PACBS by providing detailed specifications of all voting phases. We also analyze its security properties by adapting the security models introduced in chapter 4.
- In chapter 6 we conclude this thesis and describe important avenues for future work.

2 Cryptographic Preliminaries

Do not roll your own crypto

Anonymous

In this chapter we present the cryptographic building blocks of the work in this thesis. We review basic cryptographic primitives such as public-key encryption schemes, digital signatures, and their variations as well as (non-interactive) zero-knowledge proofs of knowledge. Our emphasis is both on constructions and security models. Our exposition follows [KL14; BS20; Sti19; Sch20; Gro14; GPZ15].

2.1 Basic notions

Cryptographic protocols protect the secrecy and integrity of data. All possible attempts to circumvent these properties are carried out by an adversary $\mathcal{A}$, who can be either passive but curious or active. In the first case $\mathcal{A}$ follows the protocol, but tries to extract extra information from its transcript. In the second case, the $\mathcal{A}$ can deviate from the protocol. The adversary can be computationally restricted, typically only being able to perform probabilistic polynomial-time computations or he can be computationally unlimited. Cryptographic schemes that defend against the first type of adversaries offer *computational security* while the ones that defend against the second offer *information theoretic security*. In this thesis we deal with both types of adversaries. In the case of schemes offering computational security, the guarantees rest on computational hardness assumptions which we now detail.

2.1.1 Security Assumptions

Discrete Logarithm (DL) Assumption The discrete logarithm assumption intuitively states that it is difficult to retrieve x from $y = g^x$ in a q order group $\mathbb{G}$ generated by g. More formally, it is expressed using the game in Algorithm 2.1.

The discrete logarithm assumption states that it should be hard to win the game in Algorithm 2.1.

Algorithm 2.1: $DL_{\mathcal{A},Gen}$

Input : security parameter λ
Output: $\{0,1\}$

$(\mathbb{G}, g, q) \leftarrow Gen(1^\lambda)$
$y \leftarrow_\$ \mathbb{G}$
$x \leftarrow \mathcal{A}(\textbf{guess}, \mathbb{G}, g, q, h)$
if $y = g^x$ **then**
  return 1
else
  return 0
end

Definition 2.1: DL Assumption

For all probabilistic polynomial time adversaries $\mathcal{A}$:

$$\Pr\left[DL_{\mathcal{A},Gen}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$$

Computational Diffie Hellman (CDH) Assumption Informally, the CDH assumption states that given a q-order group $\mathbb{G}$, a generator g and two group elements g^a, g^b, the value g^{ab} cannot be efficiently computed. More formally:

Algorithm 2.2: $CDH_{\mathcal{A},Gen}$

Input : security parameter λ
Output: $\{0,1\}$

$(\mathbb{G}, g, q) \leftarrow Gen(1^\lambda)$
$a, b \leftarrow_\$ \mathbb{Z}_q$
$y \leftarrow \mathcal{A}(\textbf{guess}, \mathbb{G}, g, q, g^a, g^b)$
if $y = g^{ab}$ **then**
  return 1
else
  return 0
end

Definition 2.2: CDH Assumption

For all probabilistic polynomial time adversaries $\mathcal{A}$:

$$\Pr\left[CDH_{\mathcal{A},Gen}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$$

Decisional Diffie Hellman (DDH) Assumption The DDH assumption intuitively states that triples of group elements (g^a, g^b, g^{ab}) and (g^a, g^b, g^c) of q-order group $\mathbb{G}$ generated by g cannot be efficiently distinguished.

Algorithm 2.3: $DDH_{\mathcal{A},Gen}$

Input : security parameter λ
Output: $\{0,1\}$

$(\mathbb{G}, g, q) \leftarrow Gen(1^\lambda)$
$\mathfrak{b} \leftarrow_\$ \{0,1\}$
$a, b, c \leftarrow_\$ \mathbb{Z}_q$
if $b = 0$ **then**
  $y := g^{ab}$
else
  $y := g^c$
end
$\mathfrak{b}' \leftarrow \mathcal{A}(\textbf{guess}, \mathbb{G}, g, q, g^a, g^b, y)$
if $\mathfrak{b} = \mathfrak{b}'$ **then**
  return 1
else
  return 0
end

Definition 2.3: DDH Assumption

For all probabilistic polynomial time adversaries $\mathcal{A}$:

$$\Pr\left[DDH_{\mathcal{A},Gen}(\lambda) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

There are many groups where the Decisional Diffie Hellman Assumption is believed to hold [Bon98]. One such group is the q-order subgroup of quadratic residues in $\mathbb{Z}_p^*$ where $q, p = 2q + 1$ are primes.

2.1.2 Communication channels

In every voting scheme there must be a way for the voters to communicate with the various election authorities (EA, RA, TA) and other system components. In the literature, the following types of channels have been used:

- **Broadcast channel with memory**: The message is relayed to all agents of the system and is appended to their state. In the voting literature such a channel is known as a bulletin board (BB). While the existence of such a channel is 'folklore' in the electronic voting literature its security properties were recently defined [CS14]. Except for the integrity of the contents, they include that this channel should be append-only, meaning that if an item is posted then it cannot be deleted or altered. Furthermore, only authorized participants should be able to post items and everybody should reach *consensus* about the accepted

contents. Many real-world systems [Adi08] utilize a shared database and a corresponding website; however, this requires trust in the database owner. In order to distribute the trust into peers,one must solve a consensus problem between these agents so that everybody agrees that a particular vote, for instance, is accepted and must be appended to the election log. This problem is difficult to achieve in the presence of Byzantine (malicious players) and a definitive solution eludes computer scientists since its introduction in [LSP82] despite many attempts. A distributed BB was proposed in [Kia+18]. Finally, the invention of the blockchain [Nak08] and the proliferation of distributed ledgers has caused many to propose blockchain voting schemes where the BB is implemented using a blockchain. Despite the technical similarities, such proposals do little to capture the guarantees that would satisfy the conflicting security properties we mentioned in section 1.2 [GP19].

- **Authenticated channel**: The communication infrastructure provides message integrity and authentication, thus the message contents cannot be changed en route and the sender is known. It can be implemented using cryptographic primitives such as digital signatures (cf. section 2.5). In this setting, known sender actually means pseudonymous sender, as he can be associated to a pseudonym, such as the public key of a digital signature scheme. However, all communications will be linkable to this pseudonym. A simpler, but not entirely secure, way to implement an authenticated channel is through a username and a password.
- **Secret (Private) channel**: The channel hides the contents of the message typically through the use of a cryptographic scheme.
- **Anonymous channel**: The channel provides untraceability, so that a message cannot be monitored through it and linked to a sender. This type of anonymity is usually called *sender anonymity*. Other variations can also be defined, regarding the receiver, the contents and the metadata of the communication (e.g. frequency, direction and duration). In the case of electronic voting we are mainly interested in sender and content anonymity since the receiver is well know (the EA) and the communication occurs once or in general only a few times. As a result, the anonymous channel is meant to hide the sender identity and the contents of the message. We are also interested in schemes that protect the real-world identity of a user or relevant information that can leak it (e.g. network addresses). As the identity information cannot disappear, the main approach of realizing such a channel is mixing the identity of a voter with other similar identities so that it cannot be distinguished. Anonymous channels can be implement physically (i.e. using publicly available computers in libraries, schools,

internet cafes) or using general anonymity primitives such as mixnets (cf. section 2.7) and anonymous credentials (cf. subsection 2.5.1) or existing services such as Tor [DMS04]. Formally, an anonymous channel can be considered an external factor of uncertainty, so that an $\mathcal{A}$ is not sure about the behavior of voters (e.g. if they abstained). Consequently, its existence is a *minimal* requirement for coercion resistance [JCJ05] and everlasting privacy [GPZ19] and As a result, it plays an important part in this thesis.

– **Untappable channel**: The channel is information-theoretically secure against eavesdropping. This communication mode has the strongest security requirements and is usually implemented without any use of technology i.e. in-person, or using some physical medium (the postal system). As a result, protocols that require such a channel suffer from scalability problems which in turn means that it should be used infrequently (i.e. once for many elections). Alternatively, an untappable channel can be the interaction of the voter using *local* tamper-resistant hardware. [Oka97] considers an untappable channel, a channel that is only used for the exchange of a single message - if an interaction is required, the respective channel is named a *voting booth*.

2.2 Public Key Encryption

The protection of the privacy property of electronic voting systems usually involves a public key encryption scheme. The voters use the public key of the election authority to encrypt their votes. The EA then decrypts the result.

Definition 2.4: Encryption Scheme

A public key encryption scheme $\mathcal{ES}$ is a triple of algorithms $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ and three sets $\mathbb{K}, \mathbb{M}, \mathbb{C}$ such that:

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{ES}.\mathsf{KGen}(1^\lambda)$, generates the public and secret key $\mathsf{pk}, \mathsf{sk} \in \mathbb{K}$
- $c := \mathcal{ES}.\mathsf{Enc}_{\mathsf{pk}}(m)$, encrypts the message $m \in \mathbb{M}$ using pk
- $m := \mathcal{ES}.\mathsf{Dec}_{\mathsf{sk}}(c)$, decrypts the ciphertext $c \in \mathbb{C}$ using sk

Usually the encryption algorithm is randomized, which means that a message can have many ciphertexts. We will denote such algorithms as $\mathsf{Enc}_{\mathsf{pk}}(r, m)$ where r is the randomness used. The decryption algorithm is deterministic.

The basic security notion for any encryption scheme is IND-CPA proposed in [GM84], which intuitively states that the cryptosystem must not leak anything about the ciphertext. Formally, it is defined using the experiment in Algorithm 2.4, where m_0, m_1 are of the same length. Since the encryption key is public, the adversary is free to

create encryptions of plaintexts of his choosing. We denote this by allowing the adversary access to an encryption oracle.

Algorithm 2.4: IND-CPA$_{\mathcal{A},\mathcal{ES}}$

Input : security parameter λ
Output: $\{0,1\}$

$\mathfrak{b} \leftarrow_\$ \{0,1\}$
$(\mathsf{pk},\mathsf{sk}) \leftarrow \mathcal{ES}.\mathsf{KGen}(1^\lambda)$
$(m_0,m_1) \leftarrow \mathcal{A}^{\mathcal{ES}.\mathsf{Enc}_{\mathsf{pk}}(\cdot)}(\textbf{issue},\mathsf{pk})$
$c \leftarrow \mathcal{ES}.\mathsf{Enc}_{\mathsf{pk}}(m_\mathfrak{b})$
$\mathfrak{b}' \leftarrow \mathcal{A}^{\mathcal{ES}.\mathsf{Enc}_{\mathsf{pk}}(\cdot)}(\textbf{guess},\mathsf{pk},c)$
if $\mathfrak{b} = \mathfrak{b}'$ **then**
  return 1
else
  return 0
end

Definition 2.5: IND-CPA

A public key encryption scheme $\mathcal{ES} = (\mathsf{KGen},\mathsf{Enc},\mathsf{Dec})$ is IND-CPA secure if for all probabilistic polynomial time adversaries $\mathcal{A}$:

$$\Pr[\text{IND-CPA}_{\mathcal{A},\mathcal{ES}}(\lambda) = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

ElGamal A scheme that has the IND-CPA property and is used throughout this thesis is the ElGamal cryptosystem proposed in [Gam85]. The key generation algorithm selects a group $\mathbb{G}$ of order q generated by g where the DDH assumption holds, and computes $\mathsf{pk} = g^{\mathsf{sk}}$ where $\mathsf{sk} \leftarrow_\$ \mathbb{Z}_q$. A message $m \in \mathbb{G}$ is encrypted as $\mathsf{Enc}_{\mathsf{pk}}(m) = (g^r, m \cdot \mathsf{pk}^r)$ where $r \leftarrow_\$ \mathbb{Z}_q$. To decrypt a ciphertext $\boldsymbol{c} = (c_1,c_2) \in \mathbb{G}^2$, the decryption function computes $\mathsf{Dec}_{\mathsf{sk}}(\boldsymbol{c}) = c_2 \cdot c_1^{-\mathsf{sk}}$.

ElGamal encryption requires 2 exponentiations, while decryption requires 1 exponentiation.

Malleability An interesting property of the ElGamal cryptosystem is that it is multiplicatively *homomorphic*. This means that if one multiplies (element-wise) two ciphertexts (encrypted with the same public key), one gets the encryption of the product of the corresponding plaintexts:

$$\begin{aligned}\mathsf{Enc}_{\mathsf{pk}}(m_1) \cdot \mathsf{Enc}_{\mathsf{pk}}(m_2) &= (g^{r_1}, m_1\mathsf{pk}^{r_1}) \cdot (g^{r_2}, m_2\mathsf{pk}^{r_2}) \\ &= (g^{r_1+r_2}, m_1m_2 \cdot \mathsf{pk}^{r_1+r_2}) = \mathsf{Enc}_{\mathsf{pk}}(m_1m_2)\end{aligned}$$

This implies that a ciphertext can change form (be reencrypted) without using the secret key.

$$\mathsf{ReEnc}(m_1) = \mathsf{Enc}_{\mathsf{pk}}(m_1) \cdot \mathsf{Enc}_{\mathsf{pk}}(1) = (g^{r_1+r_2}, m_1 \mathsf{pk}^{r_1+r_2}) = \mathsf{Enc}_{\mathsf{pk}}(m_1)'$$

In this thesis, we abuse notation when it comes to computations between ciphertexts. So for instance, we write $\boldsymbol{c_1 c_2}$ to mean the element-wise multiplication of two ciphertexts: $\boldsymbol{c_1 c_2} = (c_1, c_2) \cdot (c_3, c_4) = (c_1 c_3, c_2 c_4)$. By using the same rationale $\boldsymbol{c}^s = (c_1, c_2)^s = (c_1^s, c_2^s)$ and $\frac{\boldsymbol{c_1}}{\boldsymbol{c_2}} = \boldsymbol{c_1 c_2}^{-1}$.

The notion of malleability was first examined in [DDN91]. It implies that the adversary can manipulate a target-message by transforming its ciphertext into another ciphertext that somehow relates to the original. Formally, the property NM-CPA for relation R can be defined using the game in Algorithm 2.5 from [Bel+98], where $\overline{\boldsymbol{c}}, \overline{m}$ denotes a vector of $\mathsf{poly}(\lambda)$ ciphertexts and plaintexts respectively and $\mathsf{Dec}_{\mathsf{sk}}(\overline{\boldsymbol{c}})$ is the decryption of each item in the vector.

Algorithm 2.5: NM-CPA$_{\mathcal{A},\mathcal{ES}}$

Input : security parameter λ
Output: $\{0,1\}$

$\mathfrak{b} \leftarrow\!\!\$\ \{0,1\}$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{ES}.\mathsf{KGen}(1^\lambda)$
$(m_0, m_1) \leftarrow \mathcal{A}(\textbf{issue}, \mathsf{pk})$
$\boldsymbol{c} \leftarrow \mathcal{ES}.\mathsf{Enc}_{\mathsf{pk}}(m_b)$
$\overline{\boldsymbol{c}} \leftarrow \mathcal{A}^{\mathcal{ES}.\mathsf{Enc}_{\mathsf{pk}}(\cdot)}(\textbf{issue}, \mathsf{pk}, \boldsymbol{c}, m_0, m_1, \overline{m})$
if $\boldsymbol{c} \in \overline{\boldsymbol{c}}$ **then**
  return $\perp$
end
$\overline{m} \leftarrow \mathsf{Dec}_{\mathsf{sk}}(\overline{\boldsymbol{c}})$
$\mathfrak{b}' \leftarrow R(m_{\mathfrak{b}}, \overline{m})$
if $\mathfrak{b} = \mathfrak{b}'$ **then**
  return 1
else
  return 0
end

Definition 2.6: NM-CPA

A public key encryption scheme $\mathcal{ES} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ is NM-CPA secure if for all probabilistic polynomial time adversaries $\mathcal{A}$:

$$\Pr[\text{NM-CPA}_{\mathcal{A},\mathcal{ES}}(\lambda) = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

If ElGamal was additively homomorphic we could employ the homomorphic property, to aggregate the votes to compute the tally (sum) of the votes. While this cannot be done in the original version, a variation has been proposed in [CGS97], where instead of encrypting a group element, m is selected from $\mathbb{Z}_q$ and g^m is instead encrypted, i.e. $\mathsf{Enc}_{\mathsf{pk}}(m) = (g^r, g^m \cdot \mathsf{pk}^r)$. In turn, decryption yields g^m which means that in order to retrieve the plaintext one must computed the discrete logarithm of g^m. While this is assumed to be difficult in the general case, it is feasible for small values of m. Henceforth we will refer to this variation as *exponential* or *lifted* or *additive* ElGamal.

This malleability of ElGamal, can also create problems in electronic voting, as reencryption allows one to replay a vote ciphertext, without knowing its contents. The notion of IND-CPA is not enough to protect against this attack. As a result, a stronger security notion, IND-CCA, is required.

There are many types of IND-CCA security that can be defined. In this thesis, we consider adaptive IND-CCA or IND-CCA$_2$ formally defined in Algorithm 2.6 from [Bel+98]. The difference with Algorithm 2.4 is that the adversary has access to a decryption oracle that can decrypt all messages of his choice except for the challenge c. This oracle captures the intuition that $\mathcal{A}$ can learn (a function of) the plaintext.

Algorithm 2.6: IND-CCA$_{\mathcal{A},\mathcal{ES}}$

Input : security parameter λ
Output: $\{0,1\}$

$\mathfrak{b} \leftarrow_\$ \{0,1\}$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{ES}.\mathsf{KGen}(1^\lambda)$
$(m_0, m_1) \leftarrow \mathcal{A}^{\mathcal{ES}.\mathsf{Dec}_{\mathsf{sk}}, \mathcal{ES}.\mathsf{Enc}_{\mathsf{pk}}(\cdot)}(\textbf{issue}, \mathsf{pk})$
$c \leftarrow \mathcal{ES}.\mathsf{Enc}_{\mathsf{pk}}(m_b)$
$\mathfrak{b}' \leftarrow \mathcal{A}^{\mathcal{ES}.\mathsf{Dec}_{\mathsf{sk}}, \mathcal{ES}.\mathsf{Enc}_{\mathsf{pk}}(\cdot)}(\textbf{guess}, \mathsf{pk}, c)$
if $\mathfrak{b} = \mathfrak{b}'$ **then**
  return 1
else
  return 0
end

Definition 2.7: IND-CCA

A public key encryption scheme $\mathcal{ES} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ is IND-CCA secure if for all probabilistic polynomial time adversaries $\mathcal{A}$:

$$\Pr[\text{IND-CCA}_{\mathcal{A},\mathcal{ES}}(\lambda) = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

A similar game for NM-CCA can be defined based on Algorithm 2.5 and Algorithm 2.6. Non-malleability implies indistinguishability both for CPA and CCA attacks [DDN91; Bel+98]. The inverse does not hold, in general. For CPA, one can construct an NM-CPA secure cryptosystem, by using the Enc + PoK transformation, that is equipping a cryptosystem that is IND-CPA secure with a simulation-sound extractable NIZK-PoK[BPW12] (cf. section 2.4.1).

Distributed Decryption Homomorphic voting systems have an especially important problem. Since the EA can decrypt the result, it can also decrypt each individual vote, but it must be trusted *not* to do so. In order to reduce the amount of trust required, the secret decryption key should not be controlled by a single entity, since then this entity would have to be completely trusted. For this reason election schemes with distributed decryption are used.

ElGamal can be made distributed by changing the KGen and Dec functionalities. Assume that there are t decryptors that must cooperate in order to decrypt a ciphertext. In the KGen algorithm each decryptor selects $\mathsf{sk}_i \leftarrow\$ \mathbb{Z}_q$ and computes $\mathsf{pk}_i := g^{sk_i}$. The encryption public keys is $\mathsf{pk} = \prod_{i=1}^{t} \mathsf{pk}_i$ - which in turns makes the secret key $\mathsf{sk} = \sum_{i=1}^{t} g_i^{\mathsf{sk}}$. In order to decrypt a ciphertext $\boldsymbol{c} = (c_1, c_2)$, each decryptor computes and publishes $c_{1i} = c_1^{\mathsf{sk}_i}$. Everybody computes $C_1 = \prod_{i=1}^{n} c_{1i}$ and retrieves the ciphertext by $c_2 \cdot C_1^{-1}$.

While this version is distributed, it is not resilient, since a single decryptor can block the process. This problem can be fixed by using threshold secret sharing schemes (cf. section 2.6)

2.3 Commitment schemes

Encryption schemes deal with the hiding of a message. Furthermore, a ciphertext computed from a particular message with a specific key and specific randomness is binding to the message, as the sender cannot find another one that maps to the same ciphertext (under the same parameters). As a result, they can be used for the user to commit to messages, in a way that the messages are not changed and not revealed. The same functionality can be realized by a dedicated commitment scheme, defined in Definition 2.8.

Definition 2.8: Commitment scheme

A commitment scheme $\mathcal{CS}$ consists of three algorithms $(\mathsf{KGen}, \mathsf{Commit}, \mathsf{Open})$ and three sets $\mathbb{K}, \mathbb{M}, \mathbb{C}$ such that:

- $\mathsf{ck} \leftarrow \mathsf{KGen}(1^\lambda)$, generates the public commitment key $\mathsf{ck} \in \mathbb{K}$
- $(c, o) := \mathsf{Commit}_{\mathsf{ck}}(m)$, commits to the message $m \in \mathbb{M}$ using ck and generates the opening value o
- $\{0,1\} := \mathsf{Open}_{\mathsf{ck}}(c, o, m)$, validates if the commitment $c \in \mathbb{C}$ corresponds to $m \in \mathbb{M}$

The security properties of commitment schemes are that they must be *hiding* and *binding*, ensuring intuitively that a message cannot be leaked by the commitment and that given the commitment the message cannot be changed. The hiding property can be defined using a game similar to Algorithm 2.4, where o plays the role of the decryption key, while the binding property is defined in Algorithm 2.7

Algorithm 2.7: $Bind_{\mathcal{A},\mathcal{CS}}$ Game

Input : security parameter λ
Output: $\{0,1\}$

$\mathsf{ck} \leftarrow \mathcal{CS}.\mathsf{KGen}(1^\lambda)$
$(m_1, m_2, o_1, o_2, \boldsymbol{c}) \leftarrow \mathcal{A}(\mathsf{ck}, \textbf{guess})$
if $\mathcal{CS}.\mathsf{Open}_{\mathsf{ck}}(c, o_1, m_1) = 1$ AND $\mathcal{CS}.\mathsf{Open}_{\mathsf{ck}}(c, o_2, m_2) = 1$ AND $m_1 \neq m_2$ **then**
  return 1
else
  return 0
end

Definition 2.9: Binding

A commitment scheme $\mathcal{CS} = (\mathsf{KGen}, \mathsf{Commit}, \mathsf{Open})$ is binding if for all adversaries $\mathcal{A}$:

$$\Pr[Bind_{\mathcal{A},\mathcal{CS}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$$

A commitment scheme is *computationally* binding if the $\mathcal{A}$ in Definition 2.9 is probabilistic polynomial time. If this is not the case the commitment scheme is *perfectly* binding. Respectively, it can also be computationally or perfectly hiding.

Hash functions Cryptographic hash functions (denoted as H) take as input a binary string of arbitrary length $m \in \{0,1\}^*$ and produce a fixed-length l output $h \in \{0,1\}^l$ in such a way that the following requirement is met [GPZ15]:

- Collision Resistance: It is computationally infeasible to find m_1, m_2 such that $\mathsf{H}(m_1) = m_2$

Some weaker requirements that are implied by coercion resistance:

- Preimage Resistance: Given h, it is computationally infeasible to find m such that $h = \mathsf{H}(m)$
- Second Preimage Resistance: Given h, m_1, it is computationally infeasible to find m_2 such that $h = \mathsf{H}(m_1) = \mathsf{H}(m_2)$

We can instantiate a commitment scheme with a cryptographic hash function by setting $\mathsf{Commit}_{\mathsf{ck}}(m) = (\mathsf{H}(m||\mathsf{ck}), \mathsf{ck})$ and $\mathsf{Open}_{\mathsf{ck}}(c, \mathsf{ck}, m) := (c = \mathsf{H}(m||\mathsf{ck}))$. The Preimage Resistance property guarantees computational hiding, while the Collision Resistance property guarantees binding.

An ideal representation of a hash function is a *Random Oracle* (RO) [BR93], which is a black box function that when given the input x (for the first time) returns a uniformly selected string s. However, when queried again for the same x it consistently returns the same s. The random oracle hypothesis is stronger than collision resistance, as every random oracle is collision-resistant.

Pedersen commitments The most popular commitment scheme in the discrete logarithm setting was proposed in [Ped91]. The key generation algorithm selects a q order group $\mathbb{G}$ where the discrete logarithm assumptions holds and two random generators $\mathsf{ck} = (g, y)$. In order to commit to a message $m \in \mathbb{G}$, the sender selects an element $r \leftarrow_{\$} \mathbb{Z}_q$ and computes $c = \mathsf{Commit}_{\mathsf{ck}}(m, r) = g^m y^r$. Opening the commitment simply reveals the randomness r and the message m and checks if $c = g^m h^r$.

The Pedersen commitment scheme can be proved to be computationally binding and perfectly hiding. To see the first property assume that the adversary can win the game in Algorithm 2.7 and produce m_0, m_1 such that $c = \mathsf{Commit}_{\mathsf{ck}}(m_1, r_1) = g^{m_1} h^{r_1} = g^{m_2} h^{r_2}$. The the adversary can compute the discrete logarithm x, of $y = g^x$ as $x = \frac{m_1 - m_2}{r_2 - r_1}$ which contradicts that in $\mathbb{G}$ the discrete logarithm assumption holds. To see that Pedersen commitment is perfectly hiding note that $\forall (c, m) \in \mathbb{G}\ \exists!\ r = log_y(cg^{m-1}) : c = \mathsf{Commit}_{\mathsf{ck}}(m, r) = g^m y^r$. As a result, the sender can always produce pairs m, r that successfully open the commitment, thus fooling any adversary.

It is very important to note that the commitment key must be randomly (honestly) generated in order for the binding property to hold. If that is not the case then the sender can cheat. However, this has not always been the case, resulting in the breaking of real-world voting schemes [Cul+19].

2.4 Zero-Knowledge Proofs of Knowledge

The vote copying vulnerability that occurs because of the malleability of homomorphic cryptosystems can be thwarted, if the voter could somehow prove that she has access to the plaintext corresponding to an encrypted ballot, at the time of casting. Of course, this proof must not reveal its actual contents. Such a situation exactly matches the guarantees of a *Zero-Knowledge Proof*, introduced in [GMR85], where a prover P uses an interactive protocol to convince a V about the validity of a statement without disclosing anything else. Such a protocol must possess the following properties:

- *Completeness:* Honest provers (i.e. whose statement is valid) always convince honest verifiers.
- *Soundness:* Dishonest provers (i.e. who hold invalid statements) cannot convince verifiers, except with negligible probability.
- *Zero-Knowledge:* Dishonest verifiers (i.e. who want to learn more than the validity of the statement) succeed with negligible probability.

The statement to be proved is formally modelled as a binary relation $R = \{(\mathsf{prms}, w) \in P \times W\}$ where P is the set of public inputs available to both P, V and W is the set of private inputs to the P (the anything else part that must not be revealed by the proof). From R one can create the NP language $L_R = \{\mathsf{prms} \in P : \exists w(\mathsf{prms}, w) \in R\}$.

So, zero-knowledge proofs can be defined more formally as:

Definition 2.10: Zero knowledge proofs

A zero-knowledge proof for a relation R and a language L_R is a protocol $\langle \mathsf{P}(w), \mathsf{V}(), \mathsf{prms} \rangle$ where $(\mathsf{prms}, w) \in R$ between a prover and a verifier for which the following properties hold:

- Completeness: $\Pr[(\perp, 1) \leftarrow \langle \mathsf{P}(w), \mathsf{V}(), \mathsf{prms} \rangle] = 1, \ \forall (\mathsf{prms}, w) \in R$
- Soundness: $\Pr[(\perp, 1) \leftarrow \langle \mathsf{P}^*(w'), \mathsf{V}(), \mathsf{prms} \rangle] = \mathsf{negl}(\lambda), \ \forall \mathsf{prms} \notin L_R, \ \forall \mathsf{P}^*, w'$
- Zero-Knowledge: $\forall \mathsf{V}^*$, $\exists$PPT Sim such that the probability distributions of $\langle \mathsf{P}(w), \mathsf{V}(), \mathsf{prms} \rangle$ and $\langle \mathsf{Sim}, \mathsf{V}^*(), \mathsf{prms} \rangle$ are indistinguishable.

If P^* in the soundness condition of Definition 2.10 is computationally restricted then the protocol is called a zero-knowledge argument. If the probability distributions of $\langle \mathsf{P}(w), \mathsf{V}(), \mathsf{prms} \rangle$ and $\langle \mathsf{Sim}, \mathsf{V}^*(), \mathsf{prms} \rangle$ in the zero-knowledge condition of Definition 2.10 are only computationally indistinguishable then the protocol provides *computational* zero-knowledge. If the verifier in the same condition is honest then

the protocol provides *Honest Verifier* zero-knowledge (HVZK). Note that in Definition 2.10 the verifier is convinced for the existence of a witness. Zero-knowledge proofs that also convince the verifier that the prover knows a particular witness for the validity of the relation, are called *zero-knowledge Proofs of Knowledge* or *ZKPoK*. They are formalized using an additional algorithm, called the Knowledge extractor (see Definition 2.11)

2.4.1 Σ-protocols

In this thesis, we extensively use a particular variation of ZKPoK, called Σ-protocols, which are ZKPoK protocols with 3 rounds of interactions and an honest verifier. These 3 rounds consist of the following messages:

- *Commit:* The prover selects a random value and sends a binding commitment for it to the verifier.
- *Challenge:* The verifier selects a random challenge.
- *Response:* The prover responds with a combination of the witness, the committed value and the challenge.

After the response, the verifier executes a Verify functionality to accept the transcript of the protocol. As a result, the protocol transcript consist of triples (commit,challenge,response). More formally [Sch20]:

Definition 2.11: Σ-protocols

A Σ-protocol for relation R is a protocol between a prover P and a verifier V that consists of three messages (t, c, r) satisfying the following three properties:

- *Completeness:* If both P, V follow the protocol then V always accepts.
- *Special soundness:* There exists an efficient algorithm $\mathcal{E}$ (extractor) which given any transcript of two accepting conversations $(t, c, r), (t, c', r')$ with the same commit message, always produces a witness w such that $(\mathsf{prms}, w) \in R$.
- *Special honest-verifier zero-knowledge* There exists an efficient algorithm Sim (simulator) that $\forall \mathsf{prms} \in L_R$ and $\forall c \in \mathbb{C}$ can produce transcripts (t, c, r) with the same probability distribution as conversations $\langle \mathsf{P}(w), \mathsf{V}(), \mathsf{prms} \rangle$ between honest prover and verifier for any $w \in W$, $c \in \mathbb{C}$ where $(\mathsf{prms}, w) \in R$. Furthermore, if $\mathsf{prms} \notin L_R$, Sim can produce accepting conversations for a given $c \in \mathbb{C}$.

A Σ-protocol can be made non-interactive by replacing the random challenge of V, with the output of a random oracle [FS86]. In practice, the random oracle is instantiated with a hash function H. In this thesis, non-interactive Σ-protocols are denoted as $\mathsf{NIZK}\{(\mathsf{prms}), (w) : (\mathsf{prms}, w) \in \mathcal{R}\}$ where w is the private witness of the P that validates the relation $\mathcal{R}$. As a result, a non-interactive Σ-protocol comprises two functionalities: $\mathsf{NIZK.Prove}(\mathsf{prms}, w) = \pi$ that generates the proof π and $\mathsf{NIZK.Verify}(\mathsf{prms}, \pi) \in \{0, 1\}$ that outputs if a proof is valid.

Schnorr Σ-protocol

The prototypical Σ-protocol was introduced in [Sch89], and proves knowledge of the discrete logarithm x of a value $y = g^x$ in a group $\mathbb{G}$ of order q generated by g. Such a protocol can be used to prove knowledge of the private key, that corresponds to a public key in the ElGamal cryptosystem. The Schnorr protocol π_S is depicted in Figure 2.1.

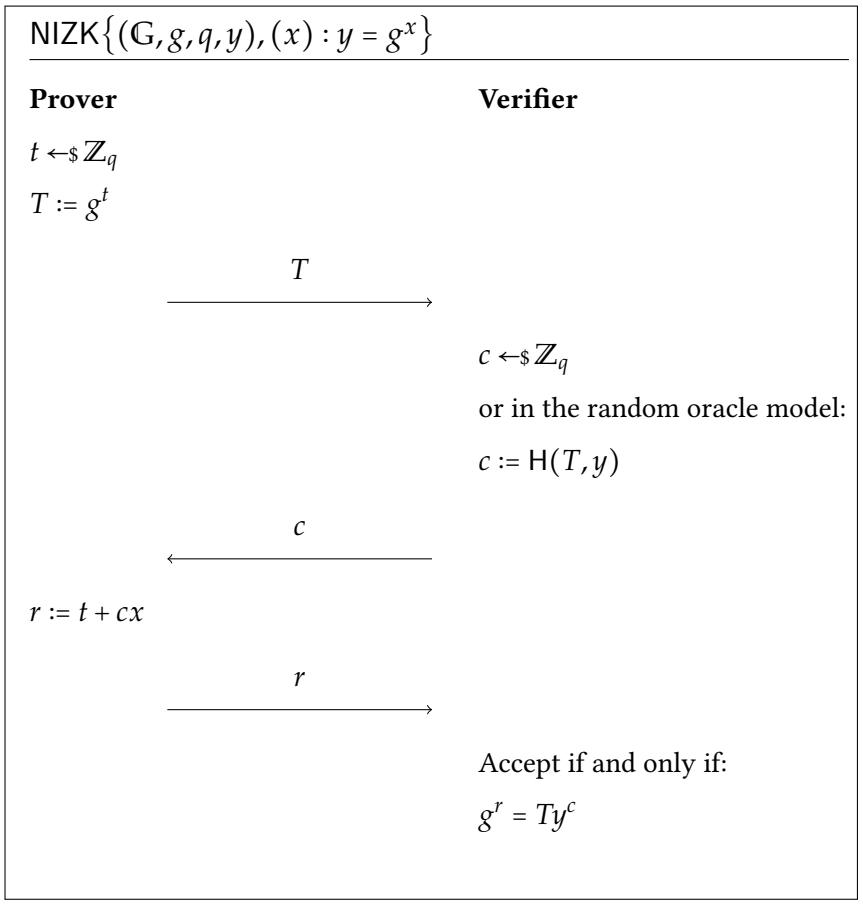


Figure 2.1: Proof of knowledge of discrete logarithm (Schnorr protocol)

The proof can be simulated by anyone by pre-selecting the commitment as $g^r y^{-c}$. It requires 1 exponentiation from P and 2 exponentiations from V.

The non-interactive version of π_S replaces the honestly generated challenge c by using a call to a random oracle H. As a result, $\mathsf{NIZK}_{\mathsf{Schnorr}}.\mathsf{Prove}((\mathbb{G}, \mathsf{H}, g, q, y), x) = (c, r)$ and $\mathsf{NIZK}_{\mathsf{Schnorr}}.\mathsf{Verify}((\mathbb{G}, \mathsf{H}, g, q, y), (c, r)) = (c = \mathsf{H}(g^r y^{-c}, y))$

Chaum-Pedersen Σ-protocol

Another Σ-protocol that is extensively used in the electronic voting literature was proposed in [CP93] and is depicted in Figure 2.2. It can be used to show discrete logarithm knowledge and equality. The Chaum-Pedersen can be equivalently formulated as a proof that the tuple (g_1, g_2, y_1, y_2) is a Diffie Hellman (DH) tuple, since if $y_1 = g_1^x$ AND $y_2 = g_2^x$, then since $g_2 = g_1^a$ for some $a \in \mathbb{Z}_q$: $(g_1, g_2, y_1, y_2) = (g_1, g_1^a, g_1^x, g_1^{ax})$.

$\mathsf{NIZK}\{(\mathbb{G}, q, g_1, g_2, y_1, y_2), (x) : y_1 = g_1^x \text{ AND } y_2 = g_2^x\}$

Prover		**Verifier**
$t \leftarrow\$ \mathbb{Z}_q$		
$T_1 := g_1^t$		
$T_2 := g_2^t$		
	$\xrightarrow{T_1, T_2}$	
		$c \leftarrow\$ \mathbb{Z}_q$
		or in the random oracle model:
		$c := \mathsf{H}(T_1, T_2, y_1, y_2)$
	$\xleftarrow{c}$	
$r := t + cx$		
	$\xrightarrow{r}$	
		Accept if and only if:
		$g_1^r = T_1 y_1^c$ AND
		$g_2^r = T_2 y_2^c$

FIGURE 2.2: Proof of knowledge of discrete logarithm equality (Chaum - Pedersen protocol)

The proof requires 2 exponentiations from P and 4 exponentiations from V. It can be simulated in a similar manner as in section 2.4.1.

The non-interactive version, π_{CP} outputs again a tuple (c, r) which must verify both relations:

$$c = \mathsf{H}(g_1^r y_1^{-c}, y_1)$$
$$c = \mathsf{H}(g_2^r y_2^{-c}, y_2)$$

Note that the Chaum-Pedersen protocol is essentially the Schnorr-protocol applied to the homomorphism $f_1 : \mathbb{G} \to (\mathbb{G} \times \mathbb{G})$. As a result, the same protocol can be used to prove that an ElGamal ciphertext is raised to a known x power, by applying the homomorphism $f_1 : (\mathbb{G} \times \mathbb{G}) \to (\mathbb{G} \times \mathbb{G}) \times (\mathbb{G} \times \mathbb{G})$.

Σ-Protocol for Pedersen commitment

The protocol in Figure 2.3 to prove knowledge of the opening of a Pedersen commitment, was proposed as an identification protocol in [Oka92].

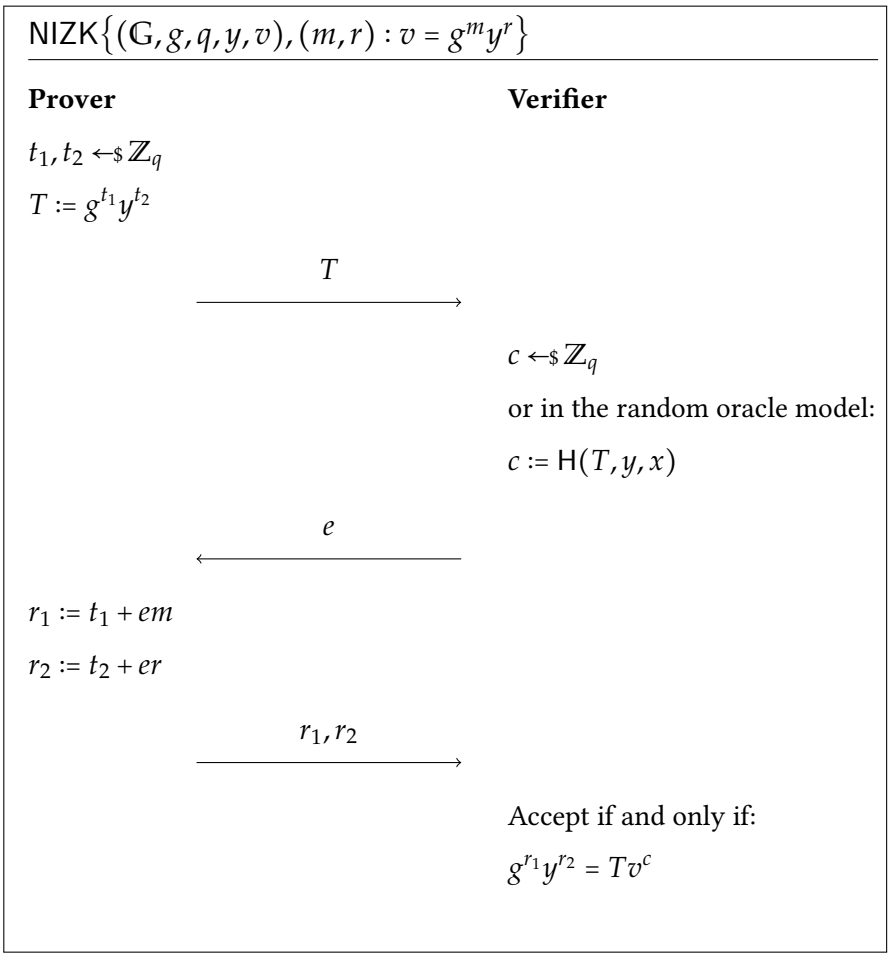


FIGURE 2.3: Proof of knowledge of Pedersen commitment openings

The proof requires 2 exponentiations from P and 3 from V. The non interactive version π_P outputs again a tuple (c, r_1, r_2) which must verify the relation: $c = \mathsf{H}(g^{r_1} y^{r_2} v^{-c}, y, c)$.

Composition of Σ-protocols

Actually the protocol in Figure 2.2 is a special case of Σ-protocol composition. In [CDS94] a complete framework is provided in order to combines Σ-protocol using disjunction, conjunction, equality and more. The disjunction case (or simply OR-PROOF) is particularly tricky since the prover must prove knowledge of any one witness in a possible set, while knowing only one. In order for the combined proof to be valid, P must combine the actual proof with 'fake' proofs for witnesses that he does not know. In order to achieve this the simulator of Definition 2.11 will be used for the fake proofs, as we described in the case of the Schnorr protocol in section 2.4.1. The disjunction of two Schnorr protocols is depicted in Figure 2.4.

$\mathsf{NIZK}\{(\mathbb{G}, g, q, y_1, y_2), (x) : y_1 = g^{x_1} \text{ OR } y_2 = g^{x_2}\}$

Prover		**Verifier**
$t_1, t_2, c_2 \leftarrow_\$ \mathbb{Z}_q$		
$T_1 := g^{t_1}$		
$T_2 := y_2^{-c_2} g^{t_2}$		
	$\xrightarrow{T_1, T_2}$	
		$c \leftarrow_\$ \mathbb{Z}_q$
		or in the random oracle model:
		$c := \mathsf{H}(T_1, T_2, y_1, y_2)$
	$\xleftarrow{c}$	
$c_1 = c - c_2$		
$r_1 := t_1 + c_1 x_1$		
$r_2 := t_2$		
	$\xrightarrow{r_1, r_2}$	
		Accept if and only if:
		$g^{r_1} = T_1 y_1^c$ AND
		$g^{r_2} = T_2 y_2^c$

FIGURE 2.4: Disjunction of Schnorr Proofs

The proof requires 3 exponentiations from P and 4 exponentiations from V.

The non-interactive version π_{ORS} is a tuple (c, r_1, r_2) that satisfies the relations:

$$c = \mathsf{H}(g^{r_1} y_1^{-c}, y_1)$$
$$c = \mathsf{H}(g^{r_2} y_2^{-c}, y_2)$$

Applications of Σ-protocols to electronic voting

Using these tools one can construct many useful Σ-protocols. In this thesis we utilize the following:

- Proof of correct ElGamal encryption π_{Enc} of a known message m in ciphertext $\boldsymbol{c}$, i.e.
  $\mathsf{NIZK}\{(\mathbb{G}, g, q, \mathsf{pk}, \boldsymbol{c}, m), (r) : \boldsymbol{c} = \mathsf{Enc}_{\mathsf{pk}}(m, r)\}$.

  If $\boldsymbol{c} = (c_1, c_2) = \mathsf{Enc}_{\mathsf{pk}}(m, r) = (g^r, m\mathsf{pk}^r)$ then $(c_1, c_2 m^{-1}) = (g^r, \mathsf{pk}^r)$. Proving correct encryption of a message reduces to proving that $(g, \mathsf{pk}, c_1, c_2 m^{-1})$ is a DH tuple, which can be done using the π_{CP} of Figure 2.2.

- Proof π_{ReEnc} that an ElGamal ciphertext $\boldsymbol{c}'$ is a reencryption of $\boldsymbol{c}$ i.e.
  $\mathsf{NIZK}\{(\mathbb{G}, g, q, \mathsf{pk}, \boldsymbol{c}, \boldsymbol{c}'), (r') : \boldsymbol{c}' = \mathsf{ReEnc}_{\mathsf{pk}}(m, r')\}$.

  If $\boldsymbol{c} = \mathsf{Enc}_{\mathsf{pk}}(m, r) = (g^r, m\mathsf{pk}^r)$ and $\boldsymbol{c}' = \mathsf{Enc}_{\mathsf{pk}}(m, r + r') = (g^{r+r'}, m\mathsf{pk}^{r+r'})$ then $\boldsymbol{c}'\boldsymbol{c}^{-1} = (g^{r'}, \mathsf{pk}^{r'})$ which reduces to π_{CP} proving that $(g, \mathsf{pk}, g^{r\prime}, \mathsf{pk}^{r\prime})$ is a DH tuple (Figure 2.2).

- Proof of correct decryption π_{Dec} of an ElGamal ciphertext $\boldsymbol{c} = (c_1, c_2)$ to a message m. In order for the decryption to be correct, the correct private key must be known and used by the prover, i.e. $g^{\mathsf{sk}} = \mathsf{pk}$ and $c_2 = m c_1^{\mathsf{sk}}$. This corresponds to the composition of two π_S or equivalently to π_{CP} (Figure 2.2): $\mathsf{NIZK}\{(\mathbb{G}, g, q, \mathsf{pk}, \boldsymbol{c}, m), (\mathsf{sk}) : g^{\mathsf{sk}} = \mathsf{pk} \text{ AND } c_2 m^{-1} = c_1^{\mathsf{sk}}\}$

- Proof of knowledge π_m of plaintext encrypted in a lifted - ElGamal ciphertext $\boldsymbol{c}$. To be more specific: $\mathsf{NIZK}\{(\mathbb{G}, g, q, \mathsf{pk}, \boldsymbol{c}), (m, r) : \boldsymbol{c}' = \mathsf{Enc}_{\mathsf{pk}}(g^m, r)\}$. This proof is a combination of the Schnorr Σ-protocol and the Σ-protocol of the Pedersen commitment openings. We can come up with the same proof by applying the general pattern of [Gro05] as described in Figure 2.5.

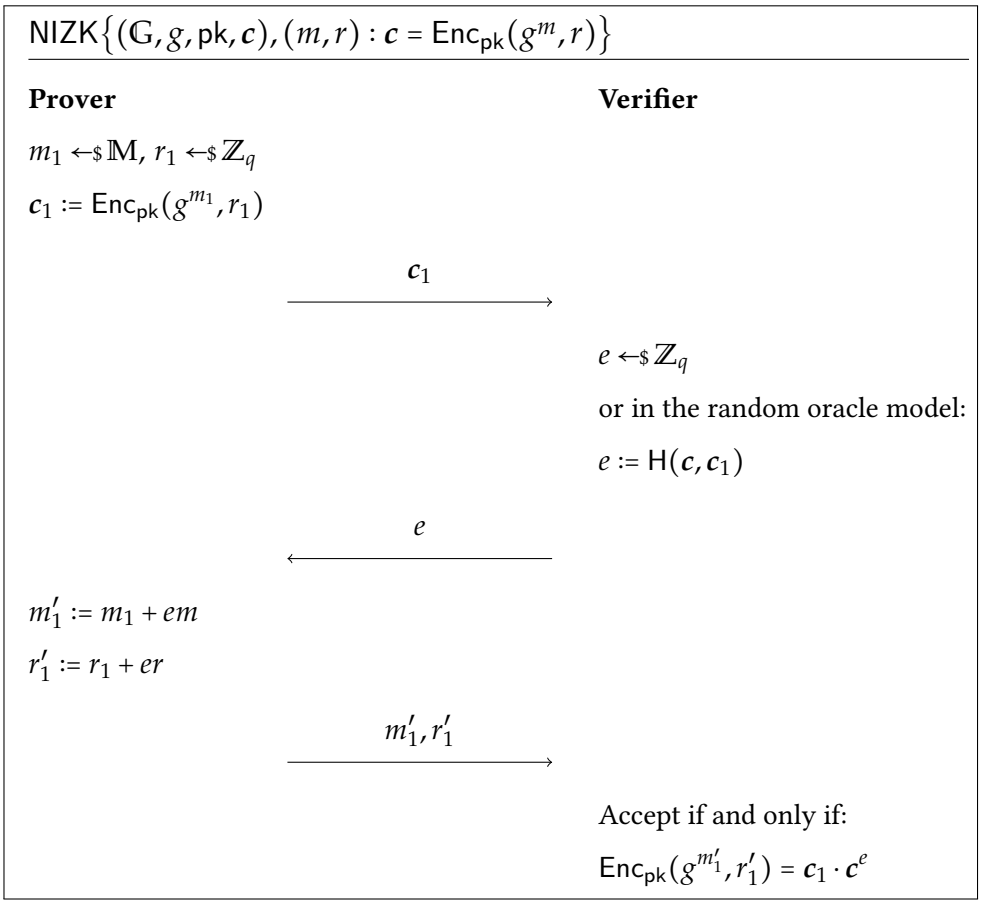


FIGURE 2.5: Proof of knowledge of plaintext in lifted ElGamal ciphertext

Σ-protocol disjunction is of particular interest in the electronic voting scenario case, since it can be used to prove vote validity, i.e that an encrypted vote belongs to a set of valid options. For instance, if the possible candidates are m_1, m_2, m_3 the voter must prove that her encrypted ballot contains an encryption of m_1 OR m_2 OR m_3. This task can be performed by using a disjunction of 3 proofs of correct ElGamal encryption π_{Enc} for m_1, m_2, m_3 respectively. Using this technique, the proof size is proportional to the number of elements in the set, which is impractical. Many techniques can improve it DBLP:conf/acns/Groth05,Camenisch2008.

Use of strong Fiat-Shamir heuristic A final note concerns non-interactive Σ-protocols utilizing the Fiat-Shamir heuristic. [BPW12] notes two variants of the heuristic, as far as the challenge step is concerned. In particular, in the *strong* variant, the challenge is computed on both the commitment and the full statement to be proved. On the other hand in the *weak* variant only the commitment is taken into account. If the prover is malicious and can adaptively changes the statement, the *weak* variant can yield unsound proofs. Recall, that use of the strong Fiat-Shamir heuristic can create NM-CPA secure schemes from IND-CPA secure schemes accompanied with NIZKPoK [BPW12].

To better illustrate the weakness, assume that a malicious prover P^* is allowed to adaptively select y_2 in $= \mathsf{NIZK}\{(\mathbb{G}, q, g_1, g_2, y_1, y_2), (x) : y_1 = g_1^x \text{ AND } , y_2 = g_2^x\}$ of Figure 2.2.

- P^* selects $a, b \leftarrow_\$ \mathbb{Z}_q$ and commits to $(T_1, T_2) = (g_1^a, g_1^b)$
- The challenge is computed only as $c = \mathsf{H}(T_1, T_2)$, without including y_1, y_2.
- The response is computed as $r = a + cx$
- The relation $g_1^r = T_1 y_1^c$ verifies correctly.
- P^* selects $y_2 = (g_2^r T_2^{-1})^{c^{-1}}$
- Note that the relation $g_2^r = T_2 y_2^c$ verifies correctly, despite the fact that with overwhelming probability $x = log_{g_1} y_1 \neq log_{g_2} y_2$.

This weakness has been actively exploited even in voting systems [BPW12; Cul+19], causing breaks in verifiability and privacy. To bypass this weakness all non-interactive versions of Σ-protocols should include the complete public input and the statement to be proved. In the exposition in the following chapters, we omit for better readability, but we note that a secure implementation must take this 'detail' into account.

2.5 Digital Signatures

Encryption schemes deal with the secrecy of a message. Digital signatures, initially proposed in [DH76] deal with the authenticity of the message, i.e. that a message was sent by the claimed sender and that it was not altered in transit. Digital signatures, can be constructed out of public key cryptosystems by using the private key for signing sk and the public key for verifying (vk), also enabling the *public verifiability* of these properties. However, as we shall see in subsection 2.5.2, the visibility of the verifiability operations can be tweaked, giving access to many interesting and useful schemes.

Definition 2.12: Digital Signature Scheme

A signature scheme $\mathcal{DS}$ is a triple of algorithms $(\mathsf{KGen}, \mathsf{Sign}, \mathsf{Verify})$ and three sets $\mathbb{K}, \mathbb{M}, \mathbb{S}$ such that:

- $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathcal{DS}.\mathsf{KGen}(1^\lambda)$, generates the verification and signing key $\mathsf{vk}, \mathsf{sk} \in \mathbb{K}$.
- $\overline{\sigma} := \mathcal{DS}.\mathsf{Sign}_{\mathsf{sk}}(m)$, signs the message $m \in \mathbb{M}$ using sk.
- $\{0,1\} := \mathcal{DS}.\mathsf{Verify}_{\mathsf{vk}}(m, \overline{\sigma})$, verifies the signature $\overline{\sigma}$ on message m, outputting 1 if and only if the verification is successful.

For the scheme to make sense, correctness must hold:

$$\forall m \in \mathbb{M}, \mathsf{vk}, \mathsf{sk} \in \mathbb{K}$$
$$\mathcal{DS}.\mathsf{Verify}_{\mathsf{vk}}(m, \mathcal{DS}.\mathsf{Sign}_{\mathsf{sk}}(m)) = 1$$

The main security property of digital signatures is *unforgeability*, meaning that only the signer S (identified by the possession of the signing key) can generate valid signatures verified by the corresponding verification key. Since the signatures are public, the aspiring forger can utilize previously signed messages. This is formalized in the game Algorithm 2.8, where the forger can also generate his own messages.

Algorithm 2.8: $\mathrm{Forge}_{\mathcal{A},\mathcal{DS}}$

Input : security parameter λ
Output: $\{0,1\}$

$((\mathsf{sk},\mathsf{vk}) \leftarrow \mathcal{DS}.\mathsf{KGen}(1^\lambda)$
$\{(m_i,\overline{\sigma}_i)\}_{i=1}^{\mathsf{poly}(\lambda)} \leftarrow \mathcal{A}^{\mathcal{DS}.\mathsf{Sign}_{\mathsf{sk}}}(\textbf{issue},\mathsf{pk})$
$(m,\overline{\sigma}) \leftarrow \mathcal{A}(\textbf{guess})$
if $\mathcal{DS}.\mathsf{Verify}(m,\overline{\sigma}) = 1$ AND $\forall i\ m \neq m_i$ **then**
  return 1
else
  return 0
end

Definition 2.13: Unforgeability

A signature scheme $\mathcal{DS}$ is (existentially) unforgeable (under a chosen message attack) if for every PPT $\mathcal{A}$ there is a negligible function of λ where:

$$\Pr\left[\mathrm{Forge}_{\mathcal{A},\mathcal{DS}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$$

Okamoto-Schnorr Signatures Signature schemes can be created from Σ-protocols, by applying the Fiat-Shamir heuristic [FS86]. Instead of supplying only the commitment to the hash function, the signer provides the message to be signed and the public key. In essence, the signature is a proof of knowledge of the private key and the message.

The Schnorr signature scheme derives directly from the Schnorr Σ-protocol [Sch91].

This thesis proposes two signature schemes that are extensions of the scheme proposed in [Oka92] also known as Okamoto-Schnorr signature scheme. The functionality is depicted in Figure 2.7.

Common input: random primes $p, q|(p-1)$, $g \in q$-order subgroup $\mathbb{G}$ of $\mathbb{Z}_p$
$\mathcal{U}$'s private input: $m \in \mathbb{M}$
S's private input : $s \in \mathbb{Z}_q$
S' public verification key: $v = g^{-s} \bmod p$
Commitment Phase. The Signer:

- Picks $r \leftarrow_\$ \mathbb{Z}_q$;
- Computes $x := g^r \bmod p$;
- Sends x to the user.

Challenge Phase. The User:

- Set $e := \mathsf{H}(m, x, v)$
- Sends e to the signer.

Signing Phase. The Signer:

- Computes $y := r + es \bmod q$;
- Outputs signature $\overline{\sigma} := (e, y)$.

(Public) Verification Phase.

- Accept if and only if $e = \mathsf{H}(m, g^y v^e, v)$

FIGURE 2.6: Schnorr Signatures

Common input: random primes $p, q|(p-1)$, g_1, g_2 elements of q-order subgroup $\mathbb{G}$ of $\mathbb{Z}_p$
$\mathcal{U}$'s private input: $m \in \mathbb{M}$
S's private input : $(s_1, s_2) \in \mathbb{Z}_q \times \mathbb{Z}_q$
S' public verification key: $v = g_1^{-s_1} g_2^{-s_2} \bmod p$
Commitment Phase. The Signer:

- Picks $r_1, r_2 \leftarrow_\$ \mathbb{Z}_q$;
- Computes $x := g_1^{r_1} g_2^{r_2} \bmod p$;
- Sends x to the user.

Challenge Phase. The User:

- Set $e := \mathsf{H}(m, x, v)$
- Sends e to the signer.

Signing Phase. The Signer:

- Computes $y_1 := r_1 + es_1 \bmod q$, $y_2 := r_2 + es_2 \bmod q$;
- Outputs signature $\overline{\sigma} := (x, e, y_1, y_2)$.

(Public) Verification Phase.

- Accept if and only if $x = g_1^{y_1} g_2^{y_2} v^e \pmod p$

FIGURE 2.7: Okamoto-Schnorr Signatures

The security of this scheme has been studied in [PS00]. Performance-wise the scheme requires 2 exponentiations for the signer and 3 for the verifier.

2.5.1 Blind Signatures

In most signature schemes the signer has access to the message to be signed. In [Cha83], David Chaum proposed a new type of digital signature, called a *blind* signature, which allows a signer to sign messages without having access to their contents, thus protecting the privacy of the message contents. The user first *blinds* the message and the signer signs it in this blinded form. The user subsequently *unblinds* the signature, and retrieves a signature for the plain message. This unblinded signature does not differ in anyway from a normal signature. Note that the signer essentially creates an intermediate representation of the final signature, who is then 'forged' by the user during unblinding to create the final signature. The motivating application, behind this primitive is anonymous centralized electronic cash. A bank blindly signs tokens created by the user to vouch for her capacity to spend a coin of particular value. Blinding protects the user, as it stops the bank from linking signing requests with signature verifications. Furthermore, the fact that the signer is impervious to the contents of the vote, motivates its use for electronic voting as well. Here the election authority authorizes the ballot submitted by the user without having access to its contents.

Blind signatures inherit the unforgeability security property from plain digital signatures. However, they must also express the signer should not be able to retrieve the signed message (in the voting scenario) or associate signatures with protocol executions (in the e-cash scenario). As a result, their security also depends on having *blindness* or *unlinkability*, formally defined in the game in Algorithm 2.9 which is adapted from [SU12]. The goal of the adversary is to learn (a function of) the message to be signed, which in blind signatures is a secret input of the user. As a result, the adversary sets up the keys to his advantage and selects two messages. The user executes the signing protocol with these messages in random order. In the end the adversary must guess which message was signed first and which second, in effect breaking the blindness property. Note that the signing is not simply an algorithm executed by the signer, but an interactive protocol executed between the signer and the user, where the output (i.e. the signature) comes from the user.

Definition 2.14

A blind signature scheme Π is *perfectly* blind if for every (unbounded) $\mathcal{A}$:

$$\Pr\left[\text{BlindExp}_{\mathcal{A},\Pi}(\lambda) = 1\right] = \frac{1}{2}$$

Algorithm 2.9: $\text{BlindExp}_{\mathcal{A},\Pi}$

Input : security parameter λ
Output: $\{0,1\}$

$(\mathsf{vk}, \mathsf{sk}, m_0, m_1) \leftarrow \mathcal{A}(\textbf{find}, 1^\lambda)$
$\mathfrak{b} \leftarrow_\$ \{0,1\}$
$\mathfrak{b}' \leftarrow_\$ \{0,1\}$
$\overline{\sigma}_\mathfrak{b} \leftarrow \mathsf{Sign}\langle \mathcal{A}(\textbf{issue}, \mathsf{sk}), \mathcal{U}(m_\mathfrak{b}), (\mathsf{pk})\rangle$
$\overline{\sigma}_{1-\mathfrak{b}} \leftarrow \mathsf{Sign}\langle \mathcal{A}(\textbf{issue}, \mathsf{sk}), \mathcal{U}(m_{1-\mathfrak{b}}), (\mathsf{pk})\rangle$
if $\mathsf{Verify}(m_\mathfrak{b}, \overline{\sigma}_\mathfrak{b}) = 1$ AND $\mathsf{Verify}(m_{1-\mathfrak{b}}, \overline{\sigma}_{1-\mathfrak{b}}) = 1$ **then**
  $\mathfrak{b}' \leftarrow \mathcal{A}(\textbf{guess}, \overline{\sigma}_0, \overline{\sigma}_1)$
end
if $\mathfrak{b} = \mathfrak{b}'$ **then**
  return 1
else
  return 0
end

The unblinding used to produce the signature has an important complication regarding unforgeability: The final signature actually comes from the user now, and not the signer. As noted in [PS96] this breaks the standard definition of forgery as described in Algorithm 2.8: The user herself creates the signature from an 'intermediate' representation provided by the signer. As a result, the standard unforgeability definitions do not make sense in this case. Instead, unforgeability is defined as the inability of a malicious user to create more signatures than the number of interactions with the signer, in direct reference to the e-cash and e-voting scenarios: the user cannot make more coins (ballots) than the bank (election authority) approved. This interpretation of unforgeability has been formalized in [PS00] with the notion of $(l, l+1)$-Forgery, where for any integer l the forger $\mathcal{A}$ must produce $l+1$ valid signatures after at most l interactions with the signer S. *One-More Forgery* is a $(l, l+1)$-Forgery, where l is polynomially bounded, while *Strong One-More Forgery* is a $(l, l+1)$-Forgery, where l is polylogarithmically bounded.

Algorithm 2.10: $\text{OneMoreForge}_{\mathcal{A},\Pi}$

Input : security parameter λ
Output: $\{0,1\}$

$(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KGen}(1^\lambda)$
$\{(m_i, \overline{\sigma}_i)\}_{i=1}^{l+1} \leftarrow \mathsf{Sign}\langle \mathsf{S}(\mathsf{sk}), \mathcal{A}(\cdot), (\mathsf{pk})\rangle_{i=1}^{\mathsf{poly}(\lambda)}$
if $(\forall i, j$ with $i \neq j \Rightarrow m_i \neq m_j)$ AND $(\forall i\ \mathsf{Verify}(\mathsf{prms}, \mathsf{pk}, \mathsf{sk}_\mathsf{V}, m_i, \overline{\sigma}_i) = 1)$ AND $k \leq l$ **then**
  return 1
else
  return 0
end

Definition 2.15

A blind signature scheme Π is one more unforgeable if for every PPT $\mathcal{A}$ there is a negligible function of λ where:

$$\Pr\left[\text{OneMoreForge}_{\mathcal{A},\Pi}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$$

The security of blind signatures has been studied in the random oracle model in [PS96]. Their results were later refined in [PS00] and revisited in [SU12]. A complexity-based approach was presented in [JLO97], however the schemes analyzed are deemed largely theoretical.

Our novel primitives CBS (section 3.1) and PACBS (section 3.1) are direct extensions of the Okamoto - Schnorr (OS) blind signature scheme [Oka92], presented in Figure 2.8.

Common input: random primes $p, q | (p-1)$, g_1, g_2 elements of q-order subgroup $\mathbb{G}$ of $\mathbb{Z}_p$
$\mathcal{U}$'s private input: $m \in \mathbb{M}$
S's private input : $(s_1, s_2) \in \mathbb{Z}_q \times \mathbb{Z}_q, \quad b \in \{0,1\}$
S' public verification key: $v = g_1^{-s_1} g_2^{-s_2} \bmod p$
Commitment Phase. The Signer:

- Picks $r_1, r_2 \leftarrow\$ \mathbb{Z}_q$;
- Computes $x := g_1^{r_1} g_2^{r_2} \bmod p$;
- Sends x to the user.

Blinding Phase. The User:

- Selects blinding factors $u_1, u_2, d \leftarrow\$ \mathbb{Z}_q$;
- Computes $x^* := x g_1^{u_1} g_2^{u_2} v^d \bmod p, e^* := \mathsf{H}(m, x^*), e := e^* - d \bmod q$;
- Sends e to the signer.

Signing Phase. The Signer:

- Computes $y_1 := r_1 + e s_1 \bmod q$, $y_2 := r_2 + e s_2 \bmod q$;
- Outputs blind signature $\beta := (x, e, y_1, y_2)$.

Unblinding Phase. The User:

- Unblinds by computing $\sigma_1 := y_1 + u_1$ and $\sigma_2 := y_2 + u_2$;
- Outputs $\overline{\sigma} := (x^*, e^*, \sigma_1, \sigma_2)$.

(Public) Verification Phase.

- Check if $x^* = g_1^{\sigma_1} g_2^{\sigma_2} v^{e^*}$

FIGURE 2.8: Okamoto-Schnorr Blind Signatures

2.5.2 Designated Verifier Signatures

Blind signatures restrict access to the message being signed in order to protect the privacy of the user. Another variation of digital signatures, *Designated Verifier Signatures (DVS)* restrict their verifiability, for a similar reason. DVS originate from designated verifier proofs, proposed in [JSI96]. These proofs are only verifiable by an entity specified by the prover during their creation. From their inception, one of their possible uses involved coercion resistance electronic voting; whether the vote is counted would not be a publicly available fact, so that a coercer could check it. Only the voter would be convinced about the validity of the vote or a credential.

More concretely, assume that the prover wants to prove the statement $\mathcal{R}(\mathsf{prms}, w)$ where w is a private input known to P. To make the proof designated by a specific verifier, the statement to be proved becomes $\mathcal{R}(\mathsf{prms}, w)$ OR I know the private key of the verifier. When the verifier receives the proof, she can be convinced of $\mathcal{R}(\mathsf{prms}, w)$, assuming that her private key has not leaked. However, a third party is not sure whether he is viewing the original proof or a proof simulated by the verifier.

More formally [LWB05]:

Definition 2.16: Designated Verifier Signatures

A designate verifier signature scheme $\mathcal{DVS}$ is a tuple of algorithms $(\mathsf{KGen}, \mathsf{Sign}, \mathsf{Sim}, \mathsf{Verify})$ and three sets $\mathbb{K}, \mathbb{M}, \mathbb{S}$ such that:

- $(\mathsf{vk_S}, \mathsf{sk_S}, \mathsf{vk_V}, \mathsf{sk_V}) \leftarrow \mathcal{DVS}.\mathsf{KGen}(1^\lambda)$, generates the verification and signing key for the signer and the designated verifier respectively.
- $\overline{\sigma} := \mathcal{DVS}.\mathsf{Sign}_{\mathsf{sk_S},\mathsf{vk_V}}(m)$, signs the message $m \in \mathbb{M}$ using $\mathsf{sk_S}$. Note that the algorithm is parameterized with the public key of the verifier.
- $\overline{\sigma} := \mathcal{DVS}.\mathsf{Sim}_{\mathsf{vk_S},\mathsf{sk_V}}(m)$, simulates a signature for the message $m \in \mathbb{M}$ using $\mathsf{sk_V}$. Note that the algorithm is parameterized with the public key of the signer.
- $\{0,1\} := \mathcal{DVS}.\mathsf{Verify}_{\mathsf{vk_S},\mathsf{vk_V}}(m, \overline{\sigma})$, verifies the signature $\overline{\sigma}$ on message m, outputting 1 if and only if the verification is successful.

Correctness implies that:

$$\begin{aligned} &\forall m \in \mathbb{M}, \mathsf{pk_S}, \mathsf{sk_S}, \mathsf{vk_V}, \mathsf{sk_V} \in \mathbb{K} \\ &\mathcal{DVS}.\mathsf{Verify}_{\mathsf{vk_S},\mathsf{vk_V}}(m, \mathcal{DVS}.\mathsf{Sign}_{\mathsf{sk_S},\mathsf{vk_V}}(m)) = 1 \text{ AND} \\ &\mathcal{DVS}.\mathsf{Verify}_{\mathsf{vk_S},\mathsf{vk_V}}(m, \mathcal{DVS}.\mathsf{Sign}_{\mathsf{vk_S},\mathsf{sk_V}}(m)) = 1 \end{aligned}$$

According to the game in Algorithm 2.8, a simulated designated verifier signature is a forgery since it is not created using the private key of the signer. However, this

forgery is useful in DVS, so the game in Algorithm 2.8 must be updated, so that the forger has access to the simulation functionality:

Algorithm 2.11: $\text{Forge}_{\mathcal{A},\mathcal{DVS}}$

Input : security parameter λ
Output: $\{0,1\}$

$(\mathsf{vk}_S, \mathsf{sk}_S, \mathsf{vk}_V, \mathsf{sk}_V) \leftarrow \mathcal{DVS}.\mathsf{KGen}(1^\lambda)$
$\{(m_i, \overline{\sigma}_i)\}_{i=1}^{\mathsf{poly}(\lambda)} \leftarrow \mathcal{A}^{\mathcal{DVS}.\mathsf{Sign}_{\mathsf{sk}_S,\mathsf{vk}_V}, \mathcal{DVS}.\mathsf{Sign}_{\mathsf{vk}_S,\mathsf{sk}_V}}(\textbf{issue}, \mathsf{vk}_S, \mathsf{vk}_V)$
$(m, \overline{\sigma}) \leftarrow \mathcal{A}(\textbf{guess})$
if $\mathcal{DVS}.\mathsf{Verify}(m, \overline{\sigma}) = 1$ AND $\forall i\ m \neq m_i$ **then**
  return 1
else
  return 0
end

Definition 2.17: Unforgeability

A signature scheme $\mathcal{DVS}$ is (existentially) unforgeable (under a chosen message attack) if for every PPT $\mathcal{A}$ there is a negligible function of λ where:

$$\Pr\left[\text{Forge}_{\mathcal{A},\mathcal{DVS}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$$

In electronic voting, designated verifier signatures have been used is schemes that provide receipt-freeness and coercion resistance. Their main use is that they provide deniability through their simulatability. All voters possess key pairs and when the EA wants to provide proof that a credential or ballot is valid, it includes the public key of the voter inside. As a result, when a coercer demands proof that the provided ballot is valid the voters simulates it using her private key.

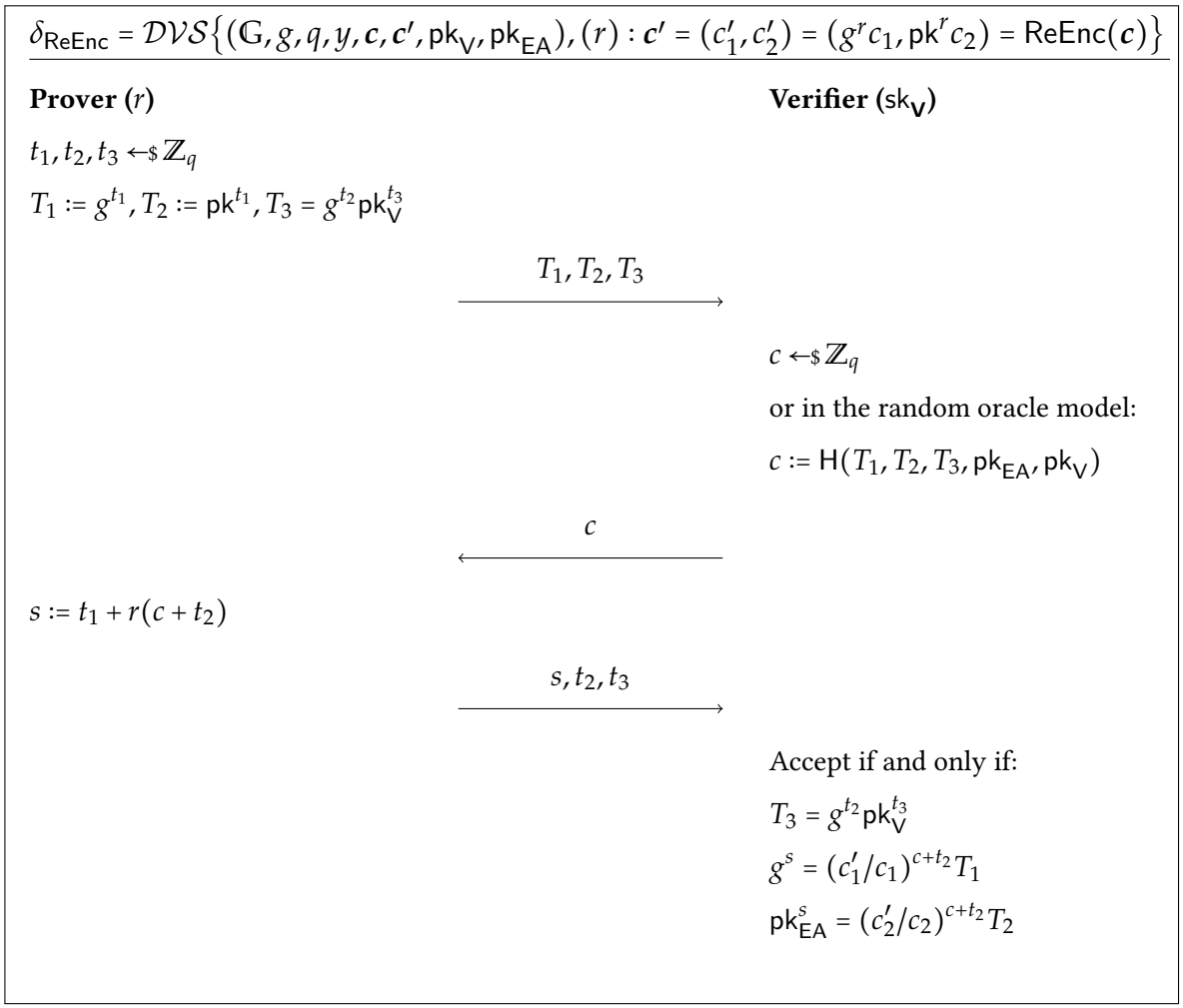


FIGURE 2.9: Designated verifier proof of correct reencryption from [HS00]

A designated verifier proof of correct reencryption δ_{ReEnc} (i.e. a variation of π_{ReEnc}), that is used in voting systems of interest [CCM08] in this thesis was proposed in [HS00] and is depicted in Figure 2.9. If the voter has kept sk_V secret, then he can fake the value T_3 by selecting t_2'', t_3': $t_2 + \mathsf{sk}_V t_3 = t_2' + \mathsf{sk}_V t_3'$ to provide a different ballot to the coercer (vote buyer) that satisfies δ_{ReEnc}.

In *strong* designated verifier signatures, also defined in [JSI96], verifiability is not public, and the private key of the verifier, must be used for verification. The simplest way to create them is to encrypt the signature with the public key of the verifier. In the proposed voting scheme, we use strong designated verifier signature in a 'reverse' manner: The voter embeds the public key of the EA in the ballot, to allow the tallier to check if a credential (signature) is valid, without the coercer being able to do so.

2.6 Threshold Secret Sharing

Knowledge/possession of the secret key in asymmetric cryptosystems yields significant power, as the holder can read encrypted messages or create signatures. In electronic voting, this can have the effect that the election authority can have access to the plain contents of an encrypted ballot. Combined with available identity information on the voters, this would mean that the election authority is aware of the voter - vote correspondence. Consequently, the EA must be trusted *not* to make use of this power. In order to reduce the required trust and the likelihood of such a scenario occurring, one could *share* the secret key between different agents so that the execution of the signing-decryption operations requires the cooperation of either all or a subset of them. In fact, by selecting agents with conflicting interests one can insert game-theoretic dynamics into the situation, thus making deviating behavior more difficult. This is achieved with the use of *secret sharing* schemes that give rise to *threshold cryptosystems*.

A secret sharing scheme, where all participants must cooperate to decrypt an ElGamal ciphertext was presented in section 2.2. Now we relax the participation requirements.

In a (t, n) threshold secret sharing scheme, an agent called the *dealer* has a secret s that wants to share between n participants called the *players* in such a way that at least t of them can reconstruct it, but no less.

The prototypical secret sharing scheme was presented in [Sha79] and is based on polynomial interpolation using Lagrange coefficients for sharing of element of s in $\mathbb{F}_p$ where p is a prime. The idea of the scheme is that a $t-1$ degree polynomial F can be uniquely reconstructed using t points $\{(x_i, p(x_i))\}_{i=1}^{t}$, as there are infinite t degree polynomials that contain these t points but a single $t-1$ degree polynomial. Let $F(x) = \sum_{i=1}^{t} y_i \prod_{j=1, j\neq i}^{t} \frac{x-x_j}{x_i-x_j}$

As a result, in order to share a secret s:

- The dealer chooses a random polynomial of degree $t-1$ so that $F(0) = s$
- Distributes n pairs $\{(x_i, F(x_i))\}_{i=1}^{n}$, $\quad x_i \neq 0$
- t players can reconstruct the polynomial (and recover s), but $t-1$ players cannot. In fact, we are not interested in reconstructing the polynomial but only in retrieving $F(0)$
- Each player i computes the Lagrange coefficients $\lambda_i(0) = \prod_{j=1, j\neq i}^{t} \frac{-x_j}{x_i-x_j} \bmod p$
- t players compute $F(0) = \sum_{i=1}^{t} F(x_i)\lambda_i(0) \bmod p$

The scheme above assumes that all the players are honest. However, this might not always be the case [Ped91]:

- The dealer might give out incorrect shares to all or part of the players. As a result, the secret will not be reconstructed. To deal with this threat the players must be able to validate them.
- The player might not present their correct shares during reconstruction. To deal with this threat the shares need to be checkable by everybody.

These issues are dealt by verifiable secret sharing which combines secret sharing with Non-Interactive zero-knowledge Proofs of Knowledge and commitment schemes [Fel87; Sch99].

2.6.1 Threshold cryptosystems

Secret sharing schemes can be combined into cryptosystems to split the decryption and signing functions in order to reduce the power of a single authority. Note that this should not be done naively, by reconstructing the secret key, since this will be useful only for a single encryption/signature, as the participants will afterwards learn the key. In fact, the sharing function needs to be embedded into the signing and decryption functions.

As an example, consider the threshold version of the ElGamal cryptosystem (section 2.2), where the key generation functionality KGen is modified, so that it outputs shares of the private key sk except the public key pk. This shares are distributed to the authorities that will have power of decryption. The encryption function takes place without change.

Decryption of a ciphertext $c = (c_1, c_2)$ proceeds in two stages:

- Each player P_i creates a decryption factor using its share of sk by computing: $s_i = {c_1}^{F(x_i)}$
- t 'decrypted' shares are combined by:

$$\prod_i s_i^{\lambda_i(0)} = \prod_i {s_i}^{F(x_i)\lambda_i(0)} =$$
$${c_1}^{\sum_i F(x_i)(i)\lambda_i(0))} = {c_1}^{F(0)} = c_1^{\mathsf{sk}}$$

- Decryption follows by computing $c_2 \cdot c_1^{-\mathsf{sk}}$

2.6.2 Plaintext Equivalence Tests

A *Plaintext Equivalence Test*, henceforth referred to as PET , is a cryptographic primitive introduced in [JJ00] which aims to convince a set of t participants that two ciphertexts indeed encrypt the same plaintext. It is meant to operate in a distributed setting, which means that the decryption key is shared among the participants. To avoid cheating [JJ00] assumes that *at least one of the participants must be honest*, i.e. truthfully follow the protocol. However, some part of the secret key must be provided to the functionality as the PET would otherwise violate the IND-CPA security property.

Definition 2.18: PET

A Plaintext Equivalence Test is a cryptographic protocol such that:

$$\mathsf{PET}\ (\mathsf{Enc}_{\mathsf{pk}}(m_1), \mathsf{Enc}_{\mathsf{pk}}(m_2)) = 1 \Leftrightarrow m_1 = m_2$$

In the case of ElGamal encrypted ciphertexts, the inputs to the protocol are two tuples $\boldsymbol{c} = (c_1, c_2)$, $\boldsymbol{c'} = (c'_1, c'_2)$ and the output is true if they encrypt the same plaintext and false otherwise. The main idea employed is that the equivalent ciphertext $\boldsymbol{c}_{\mathsf{PET}} = (\frac{c_1}{c'_1}, \frac{c_2}{c'_2})$ will be the encryption of 1 if the ciphertexts encrypt the same plaintext. Otherwise it will decrypt to a random integer. The PET functionality is depicted in Algorithm 2.12:

All players blind $\boldsymbol{c}_{\mathsf{PET}}$ and create the proof π_{i1} that they know the blinding factor. Then all players create a common blinded ciphertext and a decryption factor ψ_i along with a proof of knowledge of the private key share. Then everybody pools together the values of ψ_i and decrypt the ciphertext. If the plaintext is 1, it means that $\boldsymbol{c}, \boldsymbol{c'}$ encrypted the same message. Otherwise, decryption returns a random group element, which means that the test is unsuccessful.

Each participant in PET performs 6 exponentiations to create the values (3 for the values of the algorithms and for 3 the proofs). The verification of all the generated proofs requires $6t$ exponentiations.

[MPT20] note that *if all players collude* then they can falsely prove either that two ciphertexts do encrypt the same plaintext (when they do not) or that two ciphertexts do not encrypt the same plaintext (when in fact they do).

As a simple way to cheat in the former case consider the following simple scenario: All colluding players agree to z_i such that $\sum_i z_i = 0$. As a result, in Algorithm 2.12 $\phi = (1, 1)$, so a trivial encryption of 1 is presented, which is valid even if $m_1 \neq m_2$.

Algorithm 2.12: PET functionality for ElGamal ciphertexts

Input : $\mathbb{G}, g, q$,
pk_i such that: $\prod_i^t \mathsf{pk}_i = \mathsf{pk}$,
$\boldsymbol{c} = (c_1, c_2)$, $\boldsymbol{c'} = (c'_1, c'_2)$ encrypted under pk
Private Input: $\mathsf{sk}_i \in \mathbb{Z}_q$ such that $\sum_i^t \mathsf{sk}_i = \mathsf{sk}$
Output : $\{0, 1\}$

$\boldsymbol{c}_{\mathsf{PET}} := \frac{\boldsymbol{c}}{\boldsymbol{c'}} = (\frac{c_1}{c'_1}, \frac{c_2}{c'_2})$
$z_i \leftarrow_{\$} \mathbb{Z}_q$
$\boldsymbol{c}_{i,\mathsf{PET}} := \boldsymbol{c}_{\mathsf{PET}}^{z_i} = (c_{i1}, c_{i2}) = ((\frac{c_1}{c'_1})^{z_i}, (\frac{c_2}{c'_2})^{z_i})$
$\pi_{i1} \leftarrow \mathsf{NIZK}\{(\mathbb{G}, g, q, \mathsf{pk}, \boldsymbol{c}_{\mathsf{PET}}, \boldsymbol{c}_{i,\mathsf{PET}}), (z_i) : \boldsymbol{c}_{i,\mathsf{PET}} = \boldsymbol{c}_{\mathsf{PET}}^{z_i}\}$

Publish $(\boldsymbol{c}_{i,\mathsf{PET}}, \pi_{i1})$
Wait until all players have posted
Verify the proofs π_{i1} posted from other players

$\phi := \prod_i^t \boldsymbol{c}_{i,\mathsf{PET}} = (\prod_i c_{1i}, \prod_i c_{2i}) = (c_{1i}^{\sum_i z_i}, c_{2i}^{\sum_i z_i}) = (x, y)$
$\psi_i := x^{sk_i}$
$\pi_{i2} \leftarrow \mathsf{NIZK}\{(\mathbb{G}, g, q, \mathsf{pk}, \psi_i), (sk_i) : \psi_i = x^{sk_i}\}$

Publish (ψ_i, π_{i2})
Wait until all players have posted
Verify the proofs π_{i2} posted from other players

$\rho := y / \prod_i^t \psi_i$
if $\rho = 1$ **then**
  return 1
else
  return 0
end

The probability that this happens by accident is negligible. In order to thwart this attack, a check must be added so that if $\prod_i \boldsymbol{c}_{i,\mathsf{PET}} = (1,1)$ then the protocol aborts.

A more subtle way from [MPT20] to cheat and prove that $m_1 = m_2$ utilizes the attack against the Fiat-Shamir heuristic from [BPW12], where the full statement has not been included in the call to the random oracle (cf. section 2.4.1). This allows the ciphertexts to be selected after the proof is created, in a way that make the proof hold, regardless of whether the plaintexts are equivalent.

There are two ways to fix this vulnerability which has been found to affect most electronic voting systems that use fake credentials to achieve coercion resistance [MPT20]. The first way, makes the assumption that there is at least one honest player. However, if applied to electronic voting, it is in conflict with the universal verifiability property, where the EA is assumed to be corrupted. In order to deal with this problem, it must be ensured that the prover creates the proof *after* the ciphertexts have been selected. To do this both ciphertexts must be present in the call to the hash function implementing the Fiat-Shamir random oracle.

Whenever we use the PET primitive in this thesis, we refer to the version with the strong Fiat-Shamir transform. As a result, the distributed equivalence test can take place with all members corrupted, and provide a plaintext equivalence proof.

2.7 Verifiable Shuffles

In section 1.2 we remarked that there can be two ways to implement the ballot secrecy requirement. The one involves the secrecy of the ballot contents, implemented by an IND-CPA secure cryptosystem such as ElGamal. However, there is another possible solution: to disassociate the ballot contents from the voter identity, thus achieving vote *unlinkability*. In more detail, the users submit their votes to an anonymizing functionality and they emerge transformed without any identity information, in random order, stripping away network addresses and timing data.

However, since such a functionality directly processes ballots, there is much room for adversarial manipulation. For instance, ballots could be dropped or replaced with altered contents. As a result, an anonymizing functionality should also provide evidence that its operations followed the protocol correctly. We shall call such a functionality a *verifiable shuffle* or a *verifiable mixnet* [1].

Verifiable shuffles have been used in many politically binding elections (e.g. in Switzerland as described in [Cul+19]). Formally, they are defined in Definition 2.19 adapted from [Boy+18]:

[1] Typically a mixnet is a sequence of shuffles, but we use refer to a shuffle as a collective functionality

Definition 2.19: Verifiable shuffle

A verifiable shuffle or mix is a tuple of algorithms $\mathsf{Shuffle} = (\mathsf{Gen}, \mathsf{Submit}, \mathsf{Mix}, \mathsf{Verify})$ such that:

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Shuffle}.\mathsf{Gen}(1^\lambda)$, generates the public and secret keys.
- $c_i \leftarrow \mathsf{Shuffle}.\mathsf{Submit}_{\mathsf{pk}}(m_i)$ is an algorithm, parameterized by the public key, that allows the users to input their messages. The vector of messages is denoted $\vec{m} = (m_i)_{i=1}^n$. After all inputs are submitted the output is produced denoted $\overline{c} = (c_i)_{i=1}^n$
- $(\overline{m}, \pi_{\mathsf{Shuffle}}) \leftarrow \mathsf{Shuffle}.\mathsf{Mix}(\overline{c})$ is an algorithm that performs the actual shuffling of the n submitted messages and outputs the same underlying messages in a different form and order accompanied by a proof of correct operation.
- $\{0,1\} \leftarrow \mathsf{Shuffle}.\mathsf{Verify}(\pi_{\mathsf{Mix}})$ checks the proof of correct operation of the functionality.

Internally, $\mathsf{Shuffle}$ uses an encryption scheme and a proof of knowledge system in order to implement and generate the $\mathsf{Shuffle}.\mathsf{Mix}, \pi_{\mathsf{Mix}}$ respectively. In particular $\mathsf{Submit}_{\mathsf{pk}}(m_i) = \mathsf{Enc}_{\mathsf{pk}}(m_i, r_i)$ where the randomness r_i is selected by the user. The Verify functionality may also provide evidence that pinpoint the culprit in the case of malicious behavior, thus providing accountability as well. For instance, it can be used to prove that a particular sender submitted an invalid ciphertext (e.g by duplicating another already submitted input) or that a specific mix server did not follow the specification of the Mix functionality. The various types of mixnets are differentiated by how they perform the $\mathsf{Shuffle}.\mathsf{Mix}$ functionality and how they create the π_{Mix}.

Shuffles have a 40 year research history in the literature, during which they have been proposed as a general uses anonymity primitive with application in anonymous browsing, email, auctions, electronic voting and more. They originally appear in [Cha81]. This initial version does not include the $\mathsf{Shuffle}.\mathsf{Verify}$ functionality and π_{Mix} nor does it take advantage of encryption malleability. The $\mathsf{Shuffle}.\mathsf{Mix}$ functionality is distributed into m entities called *mix servers* each equipped with its own key pair $(\mathsf{pk}, \mathsf{sk})_{j=1}^m$. Behind the scenes the Chaumian mixnet uses the RSA cryptosystem [RSA78] where $\mathsf{Shuffle}.\mathsf{Submit}_{\mathsf{pk}}(m_i) = \{\mathsf{Enc}_{pk_j}\}_{j=m}^1(m_i)$, that is, the users encrypt their inputs with the public keys of the mix servers in reverse order. During processing each mix server, sequentially batch processes *all* ciphertexts, by removing each layer of encryption using its private key after applying the random permutation. In the end the mixnet outputs the original plaintexts in random order via the last mix server. The communication between the mix servers takes place using a BB (broadcast channel with memory), implemented in practice with a central repository. As a

result, these first generation mixnets are also called *Decryption mixnets*. Each sender has access to the BB and can identify the output of each mix server.

Second generation mixnets were proposed in [PIK93]. They are also called *Reencryption mixnets*, as they utilized the malleability properties of the underlying cryptosystem. There is one common public key, while the secret key is distributed to all the mix servers. Each of them simply permutes and reencrypts the ciphertexts. In the end the ciphertexts are threshold decrypted. As a result, reencryption mixnets are usually combined with a threshold cryptosystem.

Mixnets in these categories are secure under a threat model were the adversary is passive. In fact a single honest mix server suffices for privacy under this setting. The need for verifiable mixnets was made evident after the discovery of *tagging* attacks in [PP89; Pfi94]. An active adversary can use the malleability to his advantage by injecting a tag to the message he wants to track. For example, in the case of ElGamal encryption, in order to track the message m encrypted as $\boldsymbol{c}$, the adversary chooses $x \leftarrow_\$ \mathbb{Z}_q$ and with the help of a corrupted participant injects $\boldsymbol{c}^x$. Because of the homomorphic properties the output will contain both m, m^x which the adversary can check. This tagging attack is ubiquitous the security literature and can be used to break many properties of cryptographic voting systems. More attacks can be performed by the mix server who process all the messages from all users.

An overview of verifiability in shuffles is presented in [HM20]. In general, there are two types of verifiable mixnets: the first enable *individual* or *sender* verifiability [Wik05] where each sender can check that her *own* message was correctly shuffled.

- Message tracing: Each sender keeps the randomization and all intermediate ciphertexts used to produced the input of a decryption mixnet. Subsequently she compares the output of each mix server with its own intermediate ciphertext and posts an anonymous complain if she cannot find it.
- Verification codes: Each sender includes a random code with its message before submission to the functionality. After processing, the sender checks if the verification code appears in the output.

The first universally verifiable mixnet was proposed in [SK95], where a cut and choose protocol was proposed to check the correct operation of the mixnet. Each mix server creates a secondary shuffle (permutation and randomization values). When challenged it reveals with equal probability the second shuffle or a combination of the primary and secondary shuffle. The scheme was improved in [Abe98]. It is the basis of the Zeus fork of Helios [Tso+13].

There are two types of universally verifiable mixnets: Shuffles in the first category provide proofs for the correctness of the complete shuffle operation, with overwhelming soundness and without sacrificing message privacy. They employ what is known as proof of shuffles which apply mostly to reencryption mixnets. They utilize the malleability of the encryption scheme by reencrypting the shuffle inputs after they have been randomly permuted. That is: $\mathsf{Mix}\ (\overline{m}) := \mathsf{ReEnc}(\phi(\overline{m}))$, where ϕ is a permutation selected uniformly at random.

In general the proof of shuffle π_{Shuffle} is provided by each mix server and is the non-interactive version of a proof that the permutation and reencryption operations have been correctly executed:

$$\begin{aligned}\pi_{\mathsf{Shuffle}} := \mathsf{NIZK}\big\{(\mathsf{pk}, \overline{c}, \overline{c'}), (\phi, \overline{r}) :\\ c_{(i)} = \mathsf{ReEnc}_{\mathsf{pk}}(c_{\phi}^{-1}(i), r_i),\\ \forall i \in [n], r_i \in \overline{r}, c_i \in \overline{c}, c'_i \in \overline{c'}\big\}\end{aligned}$$

In the end, all mix servers prove correct decryption. There are many works on proofs of shuffle in the literature [FS01; Nef01; Wik09; TW10; BG12] to name a few. Their main drawback is that they are computationally demanding.

To improve the performance of universally verifiable mixnets, other methods have been proposed. The performance trade-off is the loss of soundness and in some cases some loss of privacy. For instance, in Randomized Partial Checking [JJR02] the verifier asks the prover (each mix server) to reveal the correspondence between half of the inputs and outputs of the shuffle. As a result, a mix server is caught cheating with probability $\frac{1}{2}$. Despite the fact that some portion of the output is revealed the probability that a message is traced end-to-end decreases with the number of servers. Other variations are presented in [HM20].

2.8 The road to PACBS

The main ideas of PACBS originate from blind and designated verifier signatures. They aim to protect the votes cast from the signer and to protect the verification of the signatures from the coercer. However, we also use many ideas from many different variations of digital signatures found in the literature, that fiddle with the roles and actions of the participants in the basic setting (section 2.5) to enable different usage scenarios with different security properties. For instance, we were inspired from *group* [CH91], *ring* [RST01] and *designated confirmer* signatures (DCS) [Cha94]. Moreover, ideas from cryptographic primitives such as *partially blind* signatures [AO00], *plaintext equivalence tests* (PETs) [JJ00], *designated verifier proofs* (DVP)

[JSI96] and *conditional disclosure of secrets* (CDS) [Ger+00] are utilized.

Group signatures [CH91] aim to provide *signer anonymity within a group*. This means that the signature is validated as coming from the group as a whole, without giving evidence as to which member of the group actually signed. Of course in the case of a dispute, the *traceability* property allows the group manager to specify which group member actually signed. The problem with group signatures is that they do not allow ad-hoc group creation, as the members must be predefined. This predicament is dealt with ring signatures.

The idea of a designated verifier originates from [Cha94] before being applied to [JSI96]. Its original use was to solve the problem of signer unavailability of undeniable signatures [CA89], by introducing another party to the protocol that can confirm a signature in case the signer is unavailable. The use of a group/ring signature scheme with a designated verifier signature is equivalent to the signer sending a message to the verifier through the signature. For instance, if the group members are treated as possible responses to the message to be signed, a designated group signature is equivalent to sending a particular response to the verifier. This resembles again conditional disclosure of secrets [Ger+00], which was proposed as a way for a client to obtain a secret held by a server if and only if the input of the client satisfies a certain condition. The client may hold a secret key and encrypt the input using the corresponding public key that is known to the server.

3 Publicly Auditable Conditional Blind Signatures

> Doveryay, no proveryay (Trust, but verify)
>
> Russian proverb

We are now ready to combine the primitives we discussed in chapter 2 to present one of the three main results of this thesis, *Publicly Auditable Conditional Blind Signatures - (PACBS)*, a blind signature scheme, where the validity of the generated signatures is conditional to a predicate on publicly available data. However, they are not publicly verifiable. Their validity is decided by a designated verifier [1], who is identified by a private key. To counter the actions of a corrupted signer or a corrupted verifier, that do not respect the predicate during signing or verification and produce arbitrary signatures and results, we equip the scheme with the capability to produce evidence that can be later audited, by anyone, in order to verify its security. This evidence, is intended to make up for the loss of public verifiability. Blindness provides stronger privacy guarantees towards the signer.

The main goal of PACBS is to implement the functionalities that are usually found in coercion-resistant voting protocols. The general idea is that the predicate expresses the real-world condition of whether the voter is coerced or not. Its result is embedded in the signature creation by the signer, which will be valid if and only if it evaluates to true. The election tallier then, instead of comparing all possible credentials, can simply check the validity of the signature and decide whether to count the vote or not. The application of PACBS in a voting protocol is detailed in chapter 5.

We begin the exposition by detailing a simpler version, *Conditional Blind Signatures - CBS*, from [GPZ17], that lacks the auditability properties in order to clearly illustrate the operation and security model of the primitive. We then equip CBS with auditable evidence, arriving to PACBS from [Gro+20].

[1] Based on [Gro+20]

3.1 Conditional Blind Signatures

3.1.1 Definitions

In *Conditional Blind Signatures* (CBS) the signer has a private input bit b on which it bases the validity of the signature, which is verified by a designated verifier. It is valid if and only if $b = 1$. No evidence is produced from signing or verifying in this simpler version.

Definition 3.1: Conditional Blind Signatures

A conditional blind signature (CBS) scheme is a triple (CBS.Gen, CBS.Sign, CBS.Verify) such that:

- $((\mathsf{sk}_\mathsf{S}, \mathsf{pk}_\mathsf{S}), (\mathsf{sk}_\mathsf{V}, \mathsf{pk}_\mathsf{V}), \mathsf{prms}) \leftarrow \mathsf{CBS.Gen}(1^\lambda)$
- $(\cdot, \overline{\sigma}_b) \leftarrow \mathsf{CBS.Sign}\langle \mathsf{S}(\mathsf{sk}_\mathsf{S}, b), \mathcal{U}(m), (\mathsf{prms}, \mathsf{pk})\rangle$
- $\{0,1\} \leftarrow \mathsf{CBS.Verify}(\mathsf{prms}, \mathsf{pk}, \mathsf{sk}_\mathsf{V}, m, \overline{\sigma}_b)$

CBS.Gen is an algorithm that outputs two pairs of keys, $(\mathsf{sk}_\mathsf{S}, \mathsf{pk}_\mathsf{S})$ for signing and $(\mathsf{sk}_\mathsf{V}, \mathsf{pk}_\mathsf{V})$ for verification, the message space $\mathbb{M}$ and the signature space $\mathbb{S}$, described by a set of parameters (e.g. group generators) collectively denoted as prms. For convenience both public keys are grouped together and denoted as $\mathsf{pk} = (\mathsf{pk}_\mathsf{S}, \mathsf{pk}_\mathsf{V})$.

CBS.Sign is a protocol executed between the signer and the user. The secret input of the signer is the signing key sk_S and the secret information bit b, while the secret input of the user is the message m to be signed. The public input consists of the group parameters and the public keys. The protocol output for the user is a signature $\overline{\sigma}_b$ of m, while the signer receives no output.

CBS.Verify is an algorithm which outputs a single bit representing the validity of the signature. A valid signature is one for which $\mathsf{CBS.Verify}(\cdot) = 1$. Correctness must hold, that is $\mathsf{CBS.Verify}(\cdot, m, \overline{\sigma}_b)$ outputs 1 if and only if $\overline{\sigma}_b$ is the output of the execution of the protocol CBS.Sign on message m and the secret information bit of S is $b = 1$, except with negligible probability.

3.1.2 Security Properties

The security of CBS is captured using the *Blindness*, *Unforgeability* and *Conditional Verifiability* properties. These properties are defined using the respective games for plain blind signatures [SU12] extended to accommodate for the secret conditionality bit and the separate keys of the verifier.

Blindness

The blindness property is formally defined using the game presented in Algorithm 3.1, which states that a malicious signer cannot tell which of the two messages m_0, m_1 was signed first, except with negligible probability. Note that the signatures on which the adversary is challenged are forced to be valid ($b = 1$)

Algorithm 3.1: CBS-BlindExp$_{\mathcal{A},\mathsf{CBS}}$

Input : security parameter λ
Output: $\{0,1\}$

$(\mathsf{prms}, \mathsf{pk}, \mathsf{sk}_S, \mathsf{sk}_V, m_0, m_1) \leftarrow \mathcal{A}(\textbf{find}, 1^\lambda)$
$\mathfrak{b} \leftarrow_\$ \{0,1\}$
$(\cdot, \overline{\sigma}_\mathfrak{b}) \leftarrow \mathsf{CBS.Sign}\langle \mathcal{A}(\textbf{issue}, \mathsf{sk}_S, 1), \mathcal{U}(m_\mathfrak{b}), (\mathsf{prms}, \mathsf{pk})\rangle$
$(\cdot, \overline{\sigma}_{1-\mathfrak{b}}) \leftarrow \mathsf{CBS.Sign}\langle \mathcal{A}(\textbf{issue}, \mathsf{sk}_S, 1), \mathcal{U}(m_{1-\mathfrak{b}}), (\mathsf{prms}, \mathsf{pk})\rangle$
if $\mathsf{CBS.Verify}(\mathsf{prms}, \mathsf{pk}, \mathsf{sk}_V, m_\mathfrak{b}, \overline{\sigma}_\mathfrak{b}) = 1$ AND
  $\mathsf{CBS.Verify}(\mathsf{prms}, \mathsf{pk}, \mathsf{sk}_V, m_{1-\mathfrak{b}}, \overline{\sigma}_{1-\mathfrak{b}}) = 1$ **then**
  $\mathfrak{b}' \leftarrow \mathcal{A}(\textbf{guess})$
end
if $\mathfrak{b} = \mathfrak{b}'$ **then**
  return 1
else
  return 0
end

Definition 3.2: CBS Blindness

A conditional blind signature scheme CBS is perfectly blind if for every (unbounded) $\mathcal{A}$:

$$\Pr\left[\text{CBS-BlindExp}_{\mathcal{A},\mathsf{CBS}}(\lambda) = 1\right] = \frac{1}{2}$$

Unforgeability

The unforgeability property is captured using the notion of *One More Forgery* of [PS00], which states that, if l is an integer, polynomial in the security parameter λ, an attacker can produce $l + 1$ valid signatures, after at most l successful interactions with the signer. The *Strong* One More Forgery [PS00] is a variation of the above case, where l is *polylogarithmically* bound to the security parameter. More formally, in the game CBS-OneMoreForge, $\mathcal{A}$ can obtain both valid ($b = 1$) and invalid ($b = 0$) signatures after k successful interactions.

Algorithm 3.2: CBS-OneMoreForge$_{\mathcal{A},\mathsf{CBS}}$

Input : security parameter λ
Output: $\{0,1\}$

$((\mathsf{sk_S},\mathsf{pk_S}),(\mathsf{sk_V},\mathsf{pk_V}),\mathsf{prms}) \leftarrow \mathsf{CBS.Gen}(1^\lambda)$
$\{(\cdot,(m_i,\overline{\sigma}_i))\}_{i=1}^{l+1} \leftarrow \mathsf{CBS.Sign}\langle \mathsf{S}(\mathsf{sk_S},b),\mathcal{A}(\cdot),(\mathsf{prms},\mathsf{pk})\rangle_{i=1}^{\mathsf{poly}(\lambda)}$
/* k: the number of successful protocol interactions */
if $(\forall i,j \in [l+1]$ with $i \neq j \Rightarrow m_i \neq m_j)$ AND $(\forall i \in [l+1]\ \mathsf{CBS.Verify}(\mathsf{prms},\mathsf{pk},\mathsf{sk_V},m_i,\overline{\sigma}_i)=1)$ AND $k \leq l$ **then**
  return 1
else
  return 0
end

Definition 3.3: CBS unforgeability

A conditional blind signature scheme CBS is one more unforgeable if for every PPT $\mathcal{A}$ there is a negligible function of λ where:

$$\Pr\left[\text{CBS-OneMoreForge}_{\mathcal{A},\mathsf{CBS}}(\lambda)=1\right] \leq \mathsf{negl}(\lambda)$$

In order for the verification of the signatures (and thus the checking of the forgeries) to be trustworthy, the verifier (identified by the possession of $\mathsf{sk_V}$) should be trusted. Consequently, in Algorithm 3.2, the adversary does not receive $\mathsf{sk_V}$. In effect this makes forgery a concern only against outsiders, i.e. everybody except the real signer and the designated verifier. This is consistent with the security model for designated verifier signatures [LWB05; Li+07], where signature simulations by the designated verifier are not treated as forgeries, but as aids towards the protocol's goals. In fact, in the applications of CBS, such simulations are utilized in order to make the scheme more versatile (cf. section 3.4, section 5.1).

Conditional Verifiability

For CBS, an extra property is described, called *Conditional Verifiability*, which states that an adversary cannot guess the validity of a signature without the secret verification key. The adversary is assumed to be an external entity, i.e neither the signer nor the verifier. This is justified as the signer must already know the value of b to create the signature, and the verifier learns the value of b by executing the verification functionality.

This is defined using the CBS-CondVerExp game presented in Algorithm 3.3, which intuitively resembles the IND-CPA property of public-key encryption, as it is meant to 'hide' the conditional bit of the signatures.

the adversary can adaptively obtain a polynomially restricted number of valid or invalid signatures (denoted by $\cdot$ in the input of S) by submitting messages of his choice to the signer through the CBS.Sign protocol. Then the adversary submits the challenge and is presented with a signature whose validity is decided by a random coin flipped by the challenger. The adversary can then continue to submit signing requests. In the end, he must guess the coin toss.

Algorithm 3.3: CBS-CondVerExp$_{\mathcal{A},\mathsf{CBS}}$

Input : security parameter λ
Output: $\{0,1\}$

$((\mathsf{sk}_\mathsf{S},\mathsf{pk}_\mathsf{S}),(\mathsf{sk}_\mathsf{V},\mathsf{pk}_\mathsf{V}),\mathsf{prms}) \leftarrow \mathsf{CBS.Gen}(1^\lambda)$
$\mathfrak{b} \leftarrow_\$ \{0,1\}$
$\{(\cdot,\overline{\sigma}_i)\}_{i=1}^{\mathsf{poly}(\lambda)} \leftarrow \mathsf{CBS.Sign}\langle \mathsf{S}(\mathsf{sk}_\mathsf{S},\cdot),\mathcal{A}(m_i),\mathsf{prms},\mathsf{pk}\rangle_{i=1}^{\mathsf{poly}(\lambda)}$
$m \leftarrow \mathcal{A}(\textbf{challenge})$
$(\cdot,\overline{\sigma}) \leftarrow \mathsf{CBS.Sign}\langle \mathsf{S}(\mathsf{sk}_\mathsf{S},\mathfrak{b}),\mathcal{A}(m),\mathsf{prms},\mathsf{pk}\rangle$
$\{(\cdot,\overline{\sigma}_i)\}_{i=1}^{\mathsf{poly}(\lambda)} \leftarrow \mathsf{CBS.Sign}\langle \mathsf{S}(\mathsf{sk}_\mathsf{S},\cdot),\mathcal{A}(m_i),\mathsf{prms},\mathsf{pk}\rangle_{i=1}^{\mathsf{poly}(\lambda)}$
$\mathfrak{b}' \leftarrow \mathcal{A}(\textbf{guess})$
if $\mathfrak{b} = \mathfrak{b}'$ **then**
  return 1
else
  return 0
end

Definition 3.4: CBS Conditional Verifiability

A conditional blind signature scheme CBS is conditionally verifiable if for every PPT $\mathcal{A}$ there is a negligible function of λ such that

$$\Pr\left[\text{CBS-CondVerExp}_{\mathcal{A},\mathsf{CBS}}(\lambda) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

Note that no verification oracle is provided to the adversary and he can only make signing requests. This is better suited to the intended usage of the primitive, where publicly revealing the validity of the signature would also reveal the value of the secret bit. In real-world applications, when a scheme that utilizes CBS needs to reveal the validity of a signature, it can do so by employing anonymization or obfuscation techniques, thus rendering the knowledge of the result of the predicate useless.

3.2 Okamoto-Schnorr CBS construction

A construction of the CBS primitive can be based on the Okamoto - Schnorr blind signatures [Oka92].

The parameter generation procedure, depicted in Algorithm 3.4, creates a group $\mathbb{G}$ with prime order q ($2^{\lambda-1} \leq q < 2^{\lambda}$) and generators g_1, g_2, where the DDH assumption holds. The existence of a random oracle $\mathsf{H} : \mathbb{M} \times \mathbb{G} \to \mathbb{Z}_q$ is assumed. The secret signing key comprises the values $s_1, s_2 \in \mathbb{Z}_q$ with corresponding public signing key v. The secret verification key is $s \in \mathbb{Z}_q$ with public counterpart k.

Algorithm 3.4: $\mathsf{OSCBS.Gen}(1^{\lambda})$

Input : security parameter λ

Output: $\mathsf{sk}, \mathsf{pk}, \mathsf{prms}$

$(q, \mathbb{G}) \leftarrow \mathrm{GroupGen}(1^{\lambda})$
$(g_1, g_2) \leftarrow_{\$} \mathbb{G}$
$s_1, s_2, s \leftarrow_{\$} \mathbb{Z}_q$
$v := g_1^{-s_1} g_2^{-s_2}$
$k := g_1^{s}$
$\mathsf{prms} := (\mathbb{G}, g_1, g_2, \mathsf{H})$
$\mathsf{sk_S} := (s_1, s_2)$; $\mathsf{pk_S} := v$
$\mathsf{sk_V} := s$; $\mathsf{pk_V} := k$
return $((\mathsf{sk_S}, \mathsf{pk_S}), (\mathsf{sk_V}, \mathsf{pk_V}), \mathsf{prms})$

The signing protocol is presented in Figure 3.1. It proceeds through four phases as in [Oka92]. The signer commits to an element x. The user blinds the message along with the commitment and the signer produces the blind signature. Finally, the user unblinds the signature. In summary, in the case of valid signatures, the OSCBS instantiation, is a direct adaptation of the Okamoto-Schnorr blind signatures from Figure 2.8, with the only difference being the 'lifting' of $\overline{\beta}_1$ and $\overline{\sigma}_1$ respectively. Invalid signatures, consist of randomly sampled values.

In the verification stage (Algorithm 3.5), the verifier checks the hash of the message and the commitment using the secret key $s \in \mathbb{Z}_q$. If the signer's secret bit is 1, then the signature will be valid, otherwise the verification equation will not hold. Thus the verifier, implicitly learns the secret bit of the signer.

Algorithm 3.5: $\mathsf{OSCBS.Verify}(\mathsf{prms}, \mathsf{pk}, \mathsf{sk_V}, m, \overline{\sigma})$

Input : $\mathsf{prms}, \mathsf{pk} = (v, k), \mathsf{sk_V} = s, m, \overline{\sigma} = (x^*, e^*, \sigma_1, \sigma_2)$
Output: $\{0, 1\}$

$e^* := \mathsf{H}(m, x^*)$
if $x^{*s} = \sigma_1 \cdot {g_2}^{\sigma_2 \cdot s} \cdot v^{e^* \cdot s}$ **then**
  return 1
else
  return 0
end

Common input: prms, $\mathsf{pk} = (\mathsf{pk}_\mathsf{S}, \mathsf{pk}_\mathsf{V}) = (v, k)$
$\mathcal{U}$'s private input: $m \in \mathbb{M}$
S's private input : $b \in \{0,1\}$, $\mathsf{sk}_\mathsf{S} = (s_1, s_2) \in \mathbb{Z}_q \times \mathbb{Z}_q$
Commitment Phase. The Signer:

- Picks $r_1, r_2 \leftarrow\!\!\$\, \mathbb{Z}_q$;
- Computes $x := g_1^{r_1} g_2^{r_2}$;
- Sends x to the User.

Blinding Phase. The User:

- Selects blinding factors $u_1, u_2, d \leftarrow\!\!\$\, \mathbb{Z}_q$;
- Computes $x^* := x g_1^{u_1} g_2^{u_2} v^d$, $e^* := \mathsf{H}(m, x^*)$, $e := e^* - d$;
- Sends e to the Signer.

Signing Phase. The Signer:

- Computes $y_1 := r_1 + e s_1$, $y_2 := r_2 + e s_2$;
- If $b = 1$ computes $(\beta_1, \beta_2) := (k^{y_1}, y_2)$;
- If $b = 0$ selects random $(\beta_1, \beta_2) \leftarrow\!\!\$\, \mathbb{G} \times \mathbb{Z}_q$;
- Outputs $\overline{\beta} := (x, e, \beta_1, \beta_2)$.

Unblinding Phase. The User:

- Unblinds by computing $\sigma_1 := \beta_1 \cdot k^{u_1}$ and $\sigma_2 := \beta_2 + u_2$;
- Outputs $\overline{\sigma} := (x^*, e^*, \sigma_1, \sigma_2)$.

FIGURE 3.1: The protocol $\mathsf{OSCBS.Sign}\langle \mathcal{S}((s_1, s_2), b), \mathcal{U}(m), \mathsf{prms}, \mathsf{pk} \rangle$

Performance Signing requires 3 exponentiations if the signature is valid and 2 if not. This means that a valid signature requires more processing by the signer. This can be fixed, by first selecting a random element of $\mathbb{Z}_q$ and then performing an extra exponentiation to receive a random element of $\mathbb{G}$ even if $b = 0$. The user requesting the signature must perform 4 exponentiations. Verification requires 3 exponentiations.

3.3 CBS Security Analysis

Correctness

Theorem 3.1: OS CBS Blindness

The Okamoto-Schnorr CBS scheme is correct.

Proof. Correctness follows from straightforward computations on the equation checked in Algorithm 3.5. Indeed:

$$
\begin{aligned}
&\sigma_1 \cdot {g_2}^{\sigma_2 \cdot s} \cdot v^{e^* \cdot s} = \\
&k^{y_1} k^{u_1} \cdot g_2^{(y_2+u_2)\cdot s} \cdot (g_1^{-s_1} g_2^{-s_2})^{(e+d)s} = \\
&k^{r_1+es_1+u_1} \cdot g_2^{(r_2+es_2+u_2)\cdot s} \cdot (g_1^{-s_1} g_2^{-s_2})^{(e+d)s} = \\
&g_1^{(r_1+es_1+u_1)\cdot s} \cdot g_2^{(r_2+es_2+u_2)\cdot s} \cdot (g_1^{-es_1} g_2^{-es_2})^{s} (g_1^{-ds_1} g_2^{-ds_2})^{s} = \\
&g_1^{(r_1+u_1)\cdot s} \cdot g_2^{(r_2+u_2)\cdot s} \cdot (g_1^{-ds_1} g_2^{-ds_2})^{s} = \\
&({g_1}^{r_1} {g_2}^{r_2} \cdot {g_1}^{u_1} {g_2}^{u_2} \cdot (g_1^{-ds_1} g_2^{-ds_2}))^{s} = \\
&(x \cdot {g_1}^{u_1} {g_2}^{u_2} \cdot v^{d})^{s} = \\
&x^{*s}
\end{aligned}
$$

■

Blindness

For the blindness property the arguments of the original Okamoto-Schnorr scheme in [Oka92] and [AO00] hold. More specifically, the commitment is blinded in exactly the same way in both schemes and the second parts of the signatures are identical in both cases. In addition, the message hash is hidden using the value d exactly as in [Oka92]. The first part of the signature is 'lifted', but the mapping from y_1 to k^{y_1} is one to one and onto.

Theorem 3.2: OS CBS Blindness

The Okamoto-Schnorr CBS scheme satisfies perfect blindness.

Proof. Let S^* be the unbounded adversary in the blindness game in Algorithm 3.1 and $\text{view}_i = (x_i, e_i, \beta_i)$ for $i \in \{0,1\}$ be the view [2] of S^*. There exists a unique tuple (u_1, u_2, d) that maps view_i to signature σ_j for both cases of $i, j \in \{0,1\}$.

$$
\begin{aligned}
&u_1 = log_k(\sigma_{j1} \cdot \beta_{i1}^{-1}) \Rightarrow g_1^{u_1 s} = \sigma_{j1} \cdot \beta_{i1}^{-1}, \\
&u_2 = \sigma_{j2} - \beta_{i2}, \\
&d = e_j^* - e_i
\end{aligned}
$$

This tuple causes both signatures to be valid:

$$
\begin{aligned}
x_j^{*^s} &= (x_i g_1^{u_1} g_2^{u_2} v^d)^s = g_1^{sr_1} g_2^{sr_2} g_1^{su_1} g_2^{su_2} v^{sd} \\
&= g_1^{sr_1} g_2^{sr_2} (\sigma_{j1} \beta_{i1}^{-1}) g_2^{s(\sigma_{j2} - \beta_{i2})} v^{s(e_j^* - e_i)} \\
&= (\sigma_{j1} g_2^{\sigma_{j2} s} v^{s e_j^*}) \cdot g_1^{sr_1} g_2^{sr_2} \beta_{i1}^{-1} g_2^{-s\beta_{i2}} v^{-se_i} \\
&= (\sigma_{j1} g_2^{\sigma_{j2} s} v^{s e_j^*}) \cdot g_1^{sr_1} g_2^{sr_2} g_1^{-s(r_1 + e_i s_1)} g_2^{-s(r_2 + e_i s_2)} g_1^{s_1 s e_i} g_2^{s_2 s e_i} \\
&= (\sigma_{j1} g_2^{\sigma_{j2} s} v^{s e^*})
\end{aligned}
$$

As a result, in the blindness game in Algorithm 3.1, the view of the adversary and the signatures are statistically independent for both cases of the coin flip. So the probability that an unbounded adversary succeeds in linking two protocol executions to the corresponding messages and signature pairs is exactly 1/2.

■

Strong One More Forgery

The scheme is also secure against the strong version of the One More Forgery definition [PS00]. Note that an adversary can create invalid signatures by randomly choosing $y_2 \in \mathbb{Z}_q$ and a random element of $\mathbb{G}$. As a result, in the security proof, an interaction with the signer for an invalid signature does not provide any advantage, so it can be assumed that the adversary only interacts with the signer to obtain valid signatures. Theorem 3.3 demonstrates that the scheme is secure under the strong one more forgery definition.

[2] For simplicity, both components of the blind signature as are collectively referred as β_i i.e. $\beta_i = (\beta_{i1}, \beta_{i2})$. The same applies to σ_i as well.

Theorem 3.3: OS CBS unforgeability

Suppose there exists a PPT adversary $\mathcal{A}$ that wins the OneMoreForge experiment, for l polylogarithmic in the security parameter λ, with non negligible probability. Then there exists a PPT algorithm $\mathcal{B}$ that solves the Computational Diffie Hellman problem with non negligible probability.

Proof. Let $\mathcal{A}$ be a PPT adversary that wins the game in Algorithm 3.2 with non-negligible probability. This means that it can produce $l+1$ valid signatures of the form $(x^*, e^*, \sigma_1, \sigma_2)$[3] after l interactions with the signer S and the random oracle H. The transcript of each interaction of $\mathcal{A}$ with S is the blind signature tuple (x, e, β_1, β_2). The transcript of each interaction of $\mathcal{A}$ with H is the tuple (m, x^*, e^*).

A PPT adversary $\mathcal{B}$ will be constructed, that impersonates S to make $\mathcal{A}$ produce valid signatures $\overline{\sigma} = (x^*, e^*, \sigma_1, \sigma_2)$ and $\tilde{\overline{\sigma}} = (x^*, \tilde{e^*}, \tilde{\sigma_1}, \tilde{\sigma_2})$ with the same initial message x^* and $y_2 - s_2 e \neq \tilde{y_2} - s_2 \tilde{e}$. These two valid signatures will allow $\mathcal{B}$ to break the CDH Assumption.

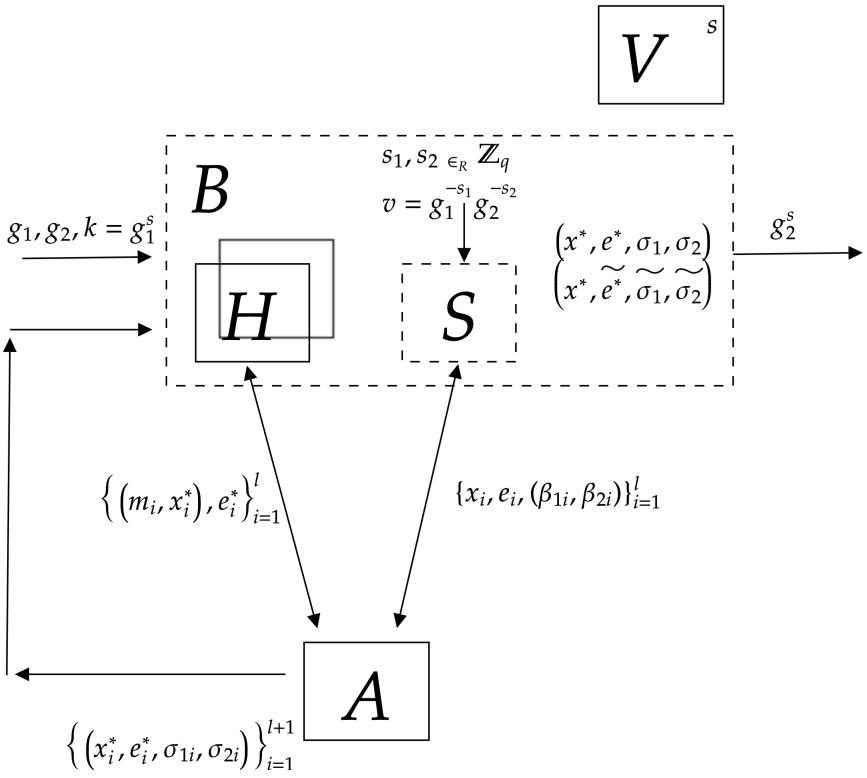


Figure 3.2: Breaking the CDH Assumption by forging OSCBS

The construction is presented at a high level in Figure 3.2. In more detail, $\mathcal{B}$ receives a triple of public group elements g_1, g_2, k where $k = g_1^s$ with unknown $s \in \mathbb{Z}_q$ and

[3] To be exact the signatures should be denoted as $\{(x^*, e^*, \sigma_1, \sigma_2)\}_{i=1}^{l+1}$, but for simplicity the indices are omitted

$g_2 = g_1^a$ for some unknown $a \in \mathbb{Z}_q$. To break the CDH Assumption $\mathcal{B}$ must compute $g_1^{as} = g_2^s$. To setup the OSCBS forgery, $\mathcal{B}$ selects $s_1, s_2 \in \mathbb{Z}_q$ and computes the public signing key $v = g_1^{-s_1} g_2^{-s_2}$ and sets the public verification key as k. $\mathcal{B}$ replays $\mathcal{A}$ until it outputs the two valid signatures $\sigma_1, \tilde{\sigma_1}$ with the same initial message x^*.

Since the verification equation in Algorithm 3.5 holds for valid signatures:

$$(x^*)^s = \sigma_1 \cdot g_2^{\sigma_2 s} \cdot v^{e^* s} \quad \text{and} \quad (x^*)^s = \tilde{\sigma_1} \cdot g_2^{\tilde{\sigma_2} s} \cdot v^{\tilde{e^*} s}$$

As x^* is the same in both cases:

$$\sigma_1 \cdot g_2^{\sigma_2 s} \cdot v^{e^* s} = \tilde{\sigma_1} \cdot g_2^{\tilde{\sigma_2} s} \cdot v^{\tilde{e^*} s} \Rightarrow \sigma_1 \cdot \tilde{\sigma_1}^{-1} = g_2^{(\tilde{\sigma_2} - \sigma_2)s} \cdot v^{(\tilde{e^*} - e^*)s}$$

All values except s are known to $\mathcal{B}$. For simplicity set:

- $\tau = \sigma_1 \cdot \tilde{\sigma_1}^{-1}$
- $\rho = \tilde{\sigma_2} - \sigma_2$
- $\phi = \tilde{e^*} - e^*$

Next, the public signing key v is analyzed:

$$\tau = g_2^{\rho s} v^{\phi s} = g_2^{\rho s} (g_1^{-s_1} g_2^{-s_2})^{\phi s} \quad = g_2^{s\rho} \cdot g_1^{-s_1 \phi s} g_2^{-s_2 \phi s} = k^{-s_1 \phi} g_2^{s(\rho - s_2 \phi)} \Rightarrow$$
$$\tau k^{s_1 \phi} = (g_2^s)^{(\rho - s_2 \phi)}$$

Now, $\mathcal{B}$ can compute g_2^s as:

$$g_2^s = (\tau k^{s_1 \phi})^{(\rho - s_2 \phi)^{-1}} \tag{3.1}$$

It remains to be proved that such valid signatures $\sigma_1, \tilde{\sigma_1}$ can be efficiently produced with non-negligible probability. This, however, is a direct consequence of the Oracle Replay Attack used to prove the unforgeability of the blind Okamoto - Schnorr signatures in [PS00]. Assume that $\mathcal{A}$ succeeds with probability at least ε in producing a $(l, l+1)$ forgery for message m. The techniques of [PS00] require that l be polylogarithmic in the security parameter.

$\mathcal{B}$ executes the signing protocol with $\mathcal{B}$ until a forgery is produced (or at most $1/\varepsilon$) times. Let $k \leq l$ be the actual number of times that $\mathcal{A}$ has interacted with the signer $\mathcal{S}$ and Q the actual number of times that $\mathcal{A}$ has interacted with H. Assume that (m, x^*) was sent to H on query j. Then each of the k signing interactions, is rerun with the same random data, except for H, which is replaced by $\tilde{\mathsf{H}}$ such that both oracles agree on the first $j-1$ answers to queries. It is proved in [PS00], that with a

polynomial overhead at most a forgery will be produced on the same (m, x^*) with non-negligible probability. Note that the data submitted to H in both OSCBS and the original Okamoto - Schnorr blind signatures follow the exact same distribution (cf. Figure 2.8). In fact, the only difference of the two protocols for valid signatures is that the first part of the *blind* signature in OSCBS is the group element k^{y_1} instead of the index $y_1 \in \mathbb{Z}_q$. Despite that the mapping between these values is one to one and onto, the difference occurs *after* the oracle call. As a result, the probabilistic analysis of the Oracle Replay Attack of [PS00] applies verbatim to OSCBS as well. ■

Conditional Verifiability

Finally, it is shown that the system is conditionally verifiable by a reduction to the DDH Assumption:

Theorem 3.4: OSCBS Conditional Verifiability

Suppose there exists a PPT adversary $\mathcal{A}$ that wins the CondVerExp with non negligible probability. Then there exists a PPT algorithm $\mathcal{B}$ that solves the Decisional Diffie Hellman problem with non-negligible probability.

Proof. $\mathcal{B}$ will be constructed (Figure 3.3).

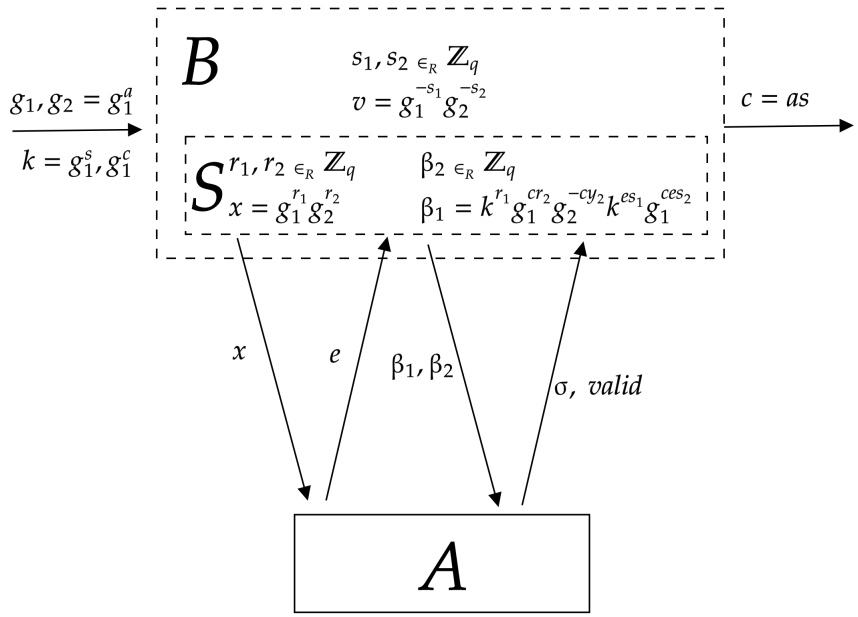


Figure 3.3: Breaking the DDH Assumption by utilizing a break in Conditional Verifiability

Its input will be the tuple g, g^a, g^s, g^c and the output will be a bit indicating whether $c = as$ or c is a random element of $\mathbb{Z}_q$. To do so, it proceeds as follows:

- $\mathcal{B}$ sets $g_1 = g$, $g_2 = g^a$ and $k = g_1^s = g^s$ and randomly chooses s_1, s_2 for $v := g_1^{-s_1} g_2^{-s_2}$. It gives g_1, g_2, k, v to $\mathcal{A}$. According to the threat model of Algorithm 3.3 the signing keys s_1, s_2 are not given to $\mathcal{A}$.
- Using the secret key (s_1, s_2) $\mathcal{B}$ can answer $\mathcal{A}$'s valid signature requests.
- When $\mathcal{B}$ gets a challenge request from $\mathcal{A}$ it randomly chooses r_1, r_2 and sends $x := g_1^{r_1} g_2^{r_2}$ to $\mathcal{A}$.
- $\mathcal{A}$ responds with e.
- $\mathcal{B}$ chooses random $\beta_2 := y_2 \in \mathbb{Z}_q$ and sets: $\beta_1 = k^{y_1} := (g^s)^{r_1}(g^c)^{r_2}(g^c)^{-y_2}(g^s)^{s_1 e}(g^c)^{s_2 e}$
- $\mathcal{B}$ sends the signature pair $(\beta_1, \beta_2) := (k^{y_1}, y_2)$ and $\mathcal{A}$ executes the unblinding phase to produce the signature $\overline{\sigma}$.
- As before, $\mathcal{B}$ responds to $\mathcal{A}$'s signing requests using the secret key (s_1, s_2).
- $\mathcal{B}$ outputs 1 (the input is a DDH tuple) if and only if $\mathcal{A}$ outputs 1 (the signature is valid).

According to Algorithm 3.5 the signature $\overline{\sigma} = (\sigma_1, \sigma_2)$ is valid if and only if: $(x^*)^s = \sigma_1 \cdot g_2^{\sigma_2 s} \cdot v^{e^* s}$

By replacing the relevant protocol transformations from Figure 3.1:

$$
\begin{aligned}
&(x g_1^{u_1} g_2^{u_2} v^d)^s = \beta_1 k^{u_1} \cdot g_2^{(\beta_2 + u_2)s} \cdot v^{(e+d)s} \Leftrightarrow \\
&x^s g_1^{s u_1} g_2^{s u_2} v^{sd} = \beta_1 g_1^{s u_1} g_2^{s \beta_2} g_2^{s u_2} v^{se} v^{sd} \Leftrightarrow \\
&x^s = \beta_1 g_2^{s \beta_2} v^{se} \Leftrightarrow \\
&\beta_1 = x^s g_2^{-s \beta_2} v^{-se} \Leftrightarrow \\
&k^{y_1} = x^s g_2^{-s y_2} v^{-se} \Leftrightarrow \\
&(g^s)^{r_1}(g^c)^{r_2}(g^c)^{-y_2}(g^s)^{s_1 e}(g^c)^{s_2 e} = x^s g_2^{-s y_2} v^{-se} \Leftrightarrow \\
&g^{s r_1} g^{c r_2} g^{-c y_2} g^{s s_1 e} g^{c s_2 e} = g^{s r_1} g_2^{s r_2} g_2^{-s y_2} g^{s s_1 e} g_2^{s s_2 e} \Leftrightarrow \\
&(g^c)^{(r_2 - y_2 + s_2 e)} = (g^{as})^{(r_2 - y_2 + s_2 e)}
\end{aligned}
$$

Provided that $r_2 - y_2 + s_2 e \neq 0$, the signature is valid if and only if $g^c = g^{as}$, which means that the input is a DDH tuple. Since y_2 is chosen randomly, $r_2 - y_2 + s_2 e = 0$ holds with negligible probability which yields the result. ■

3.4 CBS Variations

The following variations of the OSCBS construction aim to make it more versatile and, as a result, easier to be incorporated as a building block into other protocols. The design of PACBS, this work's main result, will be based on these two variations. In this section, we abstract on some key ideas that are made concrete in section 3.6.

First of all, the communication rounds of OSCBS signing protocol can be reduced, by removing the commitment phase in Figure 3.1, yielding the reduced round scheme OSCBS_r. This means that x can be replaced with a random element of $\mathbb{G}$, generated by a predetermined method. For instance, a random oracle could be used. But this is not the only option, as x could be the output of a trusted setup or a secure multi-party protocol. We only require that a common x is available to both $\mathcal{U}, \mathsf{S}$ in the beginning of the protocol. As a result, the protocol can omit the commitment message and the first round. Furthermore, this enables the verifier in possession of s to generate signatures herself, by using a random group element as v. The value of β_1 will be computed in this case by reversing the relevant part of the verification equation as $(x \cdot g_2^{-\beta_2} \cdot v^{-e})^s$, where β_2 is a random element in $\mathbb{Z}_q$.

Lemma 3.1

The reduced round OSCBS_r is l one-more unforgeable if the three round OSCBS scheme is l one-more unforgeable.

Proof. We will first describe the case where the predetermined method to compute x is a random oracle H; its programmability can be used as an advantage for the adversary. Assume that $\mathcal{A}$ is a reduced round (OSCBS_r) forger. We will construct an algorithm $\mathcal{B}$ that forges OSCBS signatures without having access to the signing key s. In order to answer $\mathcal{A}$'s requests, $\mathcal{B}$ can (by assumption) request signatures from the 3-round OSCBS signer S.

The input of $\mathcal{B}$ will be the public input of OSCBS namely $\mathbb{G}, g_1, g_2, v, k$. When $\mathcal{A}$ requests the commonly available x from H, then $\mathcal{B}$ intercepts the request and initiates a signing session with S who computes it. $\mathcal{B}$ stores x and forwards it to $\mathcal{A}$. Note that x is, by construction (Figure 3.1), a random element of $\mathbb{G}$. $\mathcal{A}$ executes the blinding phase of Figure 3.1 and sends e to $\mathcal{B}$ who in turn forwards it to S to create the blind signature $\overline{\beta} = (x, e, \beta_1, \beta_2)$. Since invalid signatures do not aid the forger we assume that the signature is always valid. Note that β_2 is a random element in $\mathbb{Z}_q$, since s_2, r_2 are sampled uniformly at random by Algorithm 3.4 and Figure 3.1 respectively. Since the signature is valid, it is easy to see that $\beta_1 = (x \cdot g_2^{-\beta_2} \cdot v^{-e})^s$. As a result, the tuple $\overline{\beta}$ received from S is indistinguishable from a valid OSCBS_r blind signature. This, by

assumption, means that $\mathcal{A}$, after an upper bound of l interactions, will generate a one-more forgery for OSCBS_r with non-negligible probability. By Figure 3.1, this forgery is valid also for OSCBS.

In order to generalize the proof, we can assume that $\mathcal{B}$ initializes signing sessions with S requesting an upper bound of $\mathsf{poly}(\lambda)$ commitment values x which are then made available for $\mathcal{A}$ to use, before the latter selects its messages. ■

Moreover, OSCBS can be easily combined with a homomorphic encryption scheme like ElGamal [Gam85]. To this end, an encryption key pair (z, h_1^z) must also be created during the parameter generation phase. The encryption secret key z is in the possession of the designated verifier. In the signing phase of Figure 3.1, S generates the first part of the blind signature as $\boldsymbol{\beta_1} := \mathsf{Enc}_{h_1^z}(k^{y_1})$. The user then unblinds it by computing $\boldsymbol{\sigma_1} := \boldsymbol{\beta_1} \cdot \mathsf{Enc}_{h_1^z}(k^{u_1})$. Due to the multiplicative homomorphic properties of the underlying cryptosystem the unblinded version of the signature is the same as CBS, albeit in encrypted form. To verify, V follows Algorithm 3.5 after decrypting $\boldsymbol{\sigma_1}$ with z.

Finally, these two variations can be combined with the signature becoming $\mathsf{Enc}_{h_1^z}((x \cdot g_2^{-\beta_2} \cdot v^{-e})^s)$ where β_2 is a random index again.

The details and security of these variations depend on the actual protocol instantiating the predetermined method to generate the first round message. As a result, their presentation in the current section should only be viewed as a stepping stone for the PACBS primitive, where a complete analysis will be presented (cf. section 3.6).

3.5 Publicly Auditable Conditional Blind Signatures

In the $\mathsf{CBS.Sign}$ protocol the conditional bit b is a private input of the signer. As a result, a malicious signer can disregard it and provide an arbitrary signature. Moreover, since the verification of a CBS signature is performed by a designated verifier, the user cannot check the validity of the signature herself. This, while counter-intuitive, is one of the design goals of the primitive, justified by its initial application to coercion resistant electronic voting. However, such a goal must not come at the expense of the signature's verifiability. In particular, the user must be protected both against a malicious signer that outputs an arbitrary signature, without taking b into account and against a verifier that does not consider $\overline{\sigma}_b$ and outputs a validity result of his liking. As a result, CBS must be augmented with a mechanism that will allow the user to verify that she was not cheated by the signer and the verifier.

This mechanism is introduced in *Publicly Auditable Conditional Blind Signatures*, a form of CBS that provides auditable evidence for the signing and verification functionalities. In particular, the Sign protocol is augmented with audit information to ensure that the signing operations were carried out correctly and Verify is augmented with evidence that verification operations conform to their specification. In order to check this evidence, two extra functionalities called PACBS.AuditSign and PACBS.AuditVrfy are proposed, that use this audit information and output if the corresponding operations are compatible with a correct protocol execution. Moreover, in order to implement the auditability requirement in a more realistic way, it is assumed that the secret input b used in CBS is replaced by a predicate function, the input of which is provided externally - e.g. $b = \mathsf{pred}(\boldsymbol{C}_1, \boldsymbol{C}_2)$ where $\boldsymbol{C}_1, \boldsymbol{C}_2$ are credentials and pred is a function that checks their equality. The signer S, accompanies the signature with evidence that he correctly followed the protocol, which means that the signature validity depends only on the result of pred on the given input. This in turn, allows an honest user, in possession of some extra secret information (for example her own credential), the definition of pred and the evidence to verify protocol compliance and be sure that the signature she holds is a valid one. On the other hand, the public (or an adversary) lacking the secret information can only check that the protocol was followed faithfully, but cannot extract the signature validity.

A useful (but not exact) intuition for the distinction between the purpose of the predicate and the PACBS.AuditSign and PACBS.AuditVrfy is the between semantics and syntax. The predicate determines the semantics of the signature, while the generated proofs and the corresponding functionalities PACBS.AuditSign and PACBS.AuditVrfy concern the syntax of the signature. Everybody can verify syntax, however, the semantics are only unlocked by the holder of the secret information.

3.5.1 Definition

Definition 3.5: PACBS Definition

A publicly auditable conditional blind signature scheme is a tuple (PACBS.Gen, PACBS.Sign, PACBS.PACBS.AuditSign, PACBS.Verify, PACBS.AuditVrfy) where:

- $((\mathsf{sk}_\mathsf{S}, \mathsf{pk}_\mathsf{S}), (\mathsf{sk}_\mathsf{V}, \mathsf{pk}_\mathsf{V}), \mathsf{prms}) \leftarrow \mathsf{PACBS.Gen}(1^\lambda)$
- $(\cdot, (\overline{\sigma}_b, \pi_{\mathsf{Sign}})) \leftarrow \mathsf{PACBS.Sign}\langle \mathsf{S}(\mathsf{sk}_\mathsf{S}), \mathcal{U}(m), (\mathsf{prms}, \mathsf{pk}, d)\rangle$
- $\{0,1\} \leftarrow \mathsf{PACBS.AuditSign}(\pi_{\mathsf{Sign}}, (\mathsf{prms}, \mathsf{pk}, d))$
- $(\{0,1\}, \pi_{\mathsf{Verify}}) \leftarrow \mathsf{PACBS.Verify}(\mathsf{sk}_\mathsf{V}, m, \overline{\sigma}_b, \mathsf{prms}, \mathsf{pk})$
- $\{0,1\} \leftarrow \mathsf{PACBS.AuditVrfy}(m, \overline{\sigma}_b, \mathsf{result}, \pi_{\mathsf{Verify}}, \mathsf{prms}, \mathsf{pk})$

PACBS.Gen is an algorithm that takes as input the security parameter 1^λ and outputs two pairs of keys $(\mathsf{sk}_\mathsf{S}, \mathsf{pk}_\mathsf{S})$ for signing and $(\mathsf{sk}_\mathsf{V}, \mathsf{pk}_\mathsf{V})$ for verification, denoted as $\mathsf{pk} = (\mathsf{pk}_\mathsf{S}, \mathsf{pk}_\mathsf{V})$ and $\mathsf{sk} = (\mathsf{sk}_\mathsf{S}, \mathsf{sk}_\mathsf{V})$. Moreover, PACBS.Gen outputs the message space $\mathbb{M}$, the signature space $\mathbb{S}$ and the public input space $\mathbb{D}$ which defines the inputs that determine the validity of the signature. These sets are described by some parameters (e.g. group generators) collectively denoted as prms. Finally, the PACBS.Gen algorithm produces a predicate function $\mathsf{pred} : \mathbb{D} \to \{0, 1\}$ that will extract the conditional part of the signature with the help of some public input.

PACBS.Sign is a protocol executed between the signer and the user. The public input consists of the parameters and the public keys as well as d from $\mathbb{D}$. The secret input of the signer is the signing key sk_S. The signer takes into account the output of the $\mathcal{U}$ algorithm on m and outputs a signature $\overline{\sigma}_b$ [4] that is conditional to some public data d, along with evidence π_{Sign} that the signer operated correctly. This evidence contains the transcript of the protocol along with proof that the internal operations were carried out correctly.

PACBS.AuditSign is an algorithm, which receives the transcript of the signature creation protocol π_{Sign} to output a bit indicating if the signing operations were carried out correctly.

PACBS.Verify is an algorithm which outputs a single bit representing the validity of the signature along with proof π_{Verify} that the verifier followed the protocol.

PACBS.AuditVrfy is an algorithm which receives the signature $\overline{\sigma}_b$, the result of the verification, result, and the evidence π_{Verify} produced during verification and outputs a bit indicating if the algorithm operations are correct with respect to the signature and the result.

Correctness must hold, which means that $\mathsf{PACBS.AuditSign}(\overline{\sigma}_b, \cdot, d)$, $\mathsf{PACBS.Verify}(\cdot, \overline{\sigma}_b, \cdot)$, $\mathsf{PACBS.AuditVrfy}(\cdot, \overline{\sigma}_b, \cdot)$ output 1 if and only if $\overline{\sigma}_b$ is the output of the execution of the protocol $\mathsf{PACBS.Sign}(\cdot, d)$ on message m with public input $d \in \mathbb{D}$ such that $\mathsf{pred}(d) = 1$ except with negligible probability.

3.5.2 Security Properties

PACBS extends the blindness, unforgeability, and conditional verifiability properties of CBS to take into account the predicate function. Furthermore, since the signing and verification operations output evidence, they are also available to the adversary, who might take advantage of them to break the security of the scheme.

[4] The notation $\overline{\sigma}_b$ is maintained, despite that there is no explicit b, in order to stress the fact that the signature is conditional

Blindness

The experiment in Algorithm 3.6 is identical to the CBS one, except for the involvement of the predicate and the audit functionalities to check the correct operations. Furthermore, the adversary is choosing the values d_0, d_1 to be used during the PACBS.Sign protocols with the restriction that $\mathsf{pred}(d_0) = \mathsf{pred}(d_1) = 1$ as imposed by the requirements of CBS that the two outputted signatures are valid. Note that the signing transcript and proofs in π_{Sign} are either generated by, or are available to the adversary (signer) and as a result, provide no advantage during the guessing stage. On the other hand, $\mathcal{A}$ can use the verification proofs along with the signatures to guess. However, as π_{Verify} will be a function of the signature, they will provide $\mathcal{A}$ with no more information than what can be obtained from the signature itself. For completeness, however, they are handed to $\mathcal{A}$ in Algorithm 3.6.

Algorithm 3.6: PACBS-BlindExp$_{\mathcal{A},\Pi}$

Input : security parameter λ
Output: $\{0,1\}$

$(\mathsf{prms}, \mathsf{pk}, \mathsf{sk}, m_0, m_1, d_0, d_1) \leftarrow \mathcal{A}(\textbf{find}, 1^\lambda)$
$\mathfrak{b} \leftarrow_\$ \{0,1\}$
$(\cdot, (\overline{\sigma}_\mathfrak{b}, \pi_{\mathsf{Sign},\mathfrak{b}})) \leftarrow \mathsf{PACBS.Sign}\langle \mathcal{A}(\textbf{issue}, \mathsf{sk}_\mathsf{S}), \mathcal{U}(m_\mathfrak{b}), (\mathsf{prms}, \mathsf{pk}, d_\mathfrak{b})) \rangle$
$(\cdot, (\overline{\sigma}_{1-\mathfrak{b}}, \pi_{\mathsf{Sign},1-\mathfrak{b}})) \leftarrow \mathsf{PACBS.Sign}\langle \mathcal{A}(\textbf{issue}, \mathsf{sk}_\mathsf{S}), \mathcal{U}(m_{1-\mathfrak{b}}), (\mathsf{prms}, \mathsf{pk}, d_{1-\mathfrak{b}})) \rangle$
$(\mathsf{result}_{\mathsf{Verify},\mathfrak{b}}, \pi_{\mathsf{Verify},\mathfrak{b}}) \leftarrow \mathsf{PACBS.Verify}(\mathsf{prms}, \mathsf{pk}, \mathsf{sk}_\mathsf{V}, m_\mathfrak{b}, \overline{\sigma}_\mathfrak{b})$
$(\mathsf{result}_{\mathsf{Verify},1-\mathfrak{b}}, \pi_{\mathsf{Verify},1-\mathfrak{b}}) \leftarrow \mathsf{PACBS.Verify}(\mathsf{prms}, \mathsf{pk}, \mathsf{sk}_\mathsf{V}, m_{1-\mathfrak{b}}, \overline{\sigma}_{1-\mathfrak{b}})$
if $\mathsf{result}_{\mathsf{Verify},\mathfrak{b}} = \mathsf{result}_{\mathsf{Verify},1-\mathfrak{b}} = 1$ **then**
  $\mathfrak{b}' \leftarrow \mathcal{A}(\textbf{guess})$
end
if $\mathfrak{b} = \mathfrak{b}'$ **then**
  return 1
else
  return 0
end

Definition 3.6: PACBS Blindness

A publicly auditable conditional blind signature scheme Π is perfectly blind if for every (unbounded) $\mathcal{A}$:

$$\Pr\left[\text{PACBS-BlindExp}_{\mathcal{A},\Pi}(\lambda) = 1\right] = \frac{1}{2}$$

Unforgeability

To capture unforgeability, the corresponding game for CBS (Algorithm 3.2) is slightly modified.

Algorithm 3.7: PACBS-OneMoreForge$_{\mathcal{A},\Pi}$

Input : security parameter λ
Output: $\{0,1\}$

$(\mathsf{sk},\mathsf{pk},\mathsf{prms}) \leftarrow \mathsf{PACBS.Gen}(1^\lambda)$
$\{(\cdot,(m_i,\overline{\sigma}_i,\pi_{\mathsf{Sign},i}))\}_{i=1}^{l+1} \leftarrow \mathsf{PACBS.Sign}\langle \mathsf{S}(\mathsf{sk}_\mathsf{S},\cdot),\mathcal{A}(\cdot)\rangle(\mathsf{prms},\mathsf{pk},d))_{i=1}^{\mathsf{poly}(\lambda)}$
/* k: the number of successful protocol interactions */
if $(\forall i,j \in [l+1]$ *with* $i \neq j \Rightarrow m_i \neq m_j)$ AND $k \leq l$ AND
$(\forall i \in [k] : \mathsf{PACBS.AuditSign}(\mathsf{prms},\mathsf{pk},\pi_{\mathsf{Sign},i}) = 1)$ AND
$(\forall i \in [l+1] : (\mathsf{result}_i,\pi_{\mathsf{Verify},i}) \leftarrow \mathsf{PACBS.Verify}(\mathsf{prms},\mathsf{pk},\mathsf{sk}_\mathsf{V},m_i,\overline{\sigma}_i);$
$\mathsf{result}_i = 1$ AND $\mathsf{PACBS.AuditVrfy}(\mathsf{prms},\mathsf{pk},m_i,\overline{\sigma}_i,\pi_{\mathsf{Verify},i}) = 1)$ **then**
  return 1
else
  return 0
end

In the game the aspiring forger chooses input d in each oracle request. If the adversary can find d for which he knows $\mathsf{pred}(d)$ he can get valid and invalid signatures at will. Using this oracle, the adversary tries to obtain more than k valid signatures where k is the number of interactions resulting in valid signature output.

Definition 3.7: PACBS Unforgeability

A publicly auditable conditional blind signature scheme Π is one more unforgeable if for every PPT $\mathcal{A}$ there is a negligible function of λ where:
$\Pr\left[\text{PACBS-OneMoreForge}_{\mathcal{A},\Pi}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$.

Conditional Verifiability

Slight modifications are also needed to capture conditional verifiability. In particular, b needs to be replaced with the value of the predicate function.

In the game, the adversary has access to a signing oracle of his choice and can ask for signatures with auxiliary values that he chooses. $\mathcal{A}$ also selects the messages that will be signed. He is challenged on a signature which is either valid or invalid depending on a coin flip on random auxiliary values. His goal is to determine the result of the coin flip or equivalently the value of the predicate. Note that, since S is assumed honest, it follows the protocol in both cases. As a result, while π_{Sign}, will be valid in all interactions, the collected proofs might leak information about the predicate, so they are handed to $\mathcal{A}$ in the guessing stage.

Algorithm 3.8: PACBS-CondVerExp$_{\mathcal{A},\Pi}$

Input : security parameter λ
Output: $\{0,1\}$

$(\mathsf{sk},\mathsf{pk},\mathsf{prms}) \leftarrow \mathsf{PACBS.Gen}(1^\lambda)$
$\mathfrak{b} \leftarrow_\$ \{0,1\}$
Let $R_0 = \{d \mid \text{s.t. } \mathsf{pred}(d) = 0\}$ and $R_1 = \{d' \mid \text{s.t. } \mathsf{pred}(d') = 1\}$
$d^* \leftarrow_\$ R_\mathfrak{b}$
$\{(\cdot,(\overline{\sigma}_i,\pi_{\mathsf{Sign},i}))\}_{i=1}^{\mathsf{poly}(\lambda)} \leftarrow \mathsf{PACBS.Sign}\langle \mathsf{S}(\mathsf{sk}_\mathsf{S}), \mathcal{A}(m_i), (\mathsf{prms},\mathsf{pk},d_i)\rangle_{i=1}^{\mathsf{poly}(\lambda)}$
$m \leftarrow \mathcal{A}(\textbf{challenge})$
$(\cdot,(\overline{\sigma}_\mathfrak{b},\pi_{\mathsf{Sign},\mathfrak{b}})) \leftarrow \mathsf{PACBS.Sign}\langle \mathsf{S}(\mathsf{sk}_\mathsf{S}), \mathcal{A}(m), (\mathsf{prms},\mathsf{pk},d^*)\rangle$
$\{(\cdot,(\overline{\sigma}_i,\pi_{\mathsf{Sign},i}))\}_{i=1}^{\mathsf{poly}(\lambda)} \leftarrow \mathsf{PACBS.Sign}\langle \mathsf{S}(\mathsf{sk}_\mathsf{S}), \mathcal{A}(m_i), (\mathsf{prms},\mathsf{pk},d_i)\rangle_{i=1}^{\mathsf{poly}(\lambda)}$
$\mathfrak{b}' \leftarrow \mathcal{A}(\textbf{guess}, \overline{\sigma}_c, \pi_{\mathsf{Sign},\mathfrak{b}}, \{\overline{\sigma}_i\}_{i=1}^{\mathsf{poly}(\lambda)}, \{\pi_{\mathsf{Sign},i}\}_{i=1}^{\mathsf{poly}(\lambda)})$
if $\mathfrak{b} = \mathfrak{b}'$ **then**
  return 1
else
  return 0
end

Definition 3.8: PACBS Conditional Verifiability

A publicly auditable conditional blind signature scheme Π is conditionally verifiable if for every PPT $\mathcal{A}$ there is a negligible function of λ such that $\Pr\left[\text{PACBS-CondVerExp}_{\mathcal{A},\Pi}(\lambda) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$.

Note that for the definition of a PACBS construction to be meaningful, the predicate pred should be infeasible to compute on random values of its domain.

Public Auditability

Public auditability for PACBS is defined with respect to the signing and verification functionalities.

In the case of PACBS.AuditSign, the desideratum is the property that the user's output of the protocol respects the value of $\mathsf{pred}(d)$ even when executed against a malicious signer. This means that if $\overline{\sigma}_b$ is the output of PACBS.Sign on secret input m, then it is valid if and only if $\mathsf{pred}(d) = 1$. In other words, if $(\text{result}, \pi_{\mathsf{Verify}}) = \mathsf{PACBS.Verify}(\mathsf{prms},\mathsf{pk},\mathsf{sk}_\mathsf{V},m,\overline{\sigma}_b)$ then $\text{result} = \mathsf{pred}(d)$. For this the PASignExp experiment is used, which is defined in Algorithm 3.9.

In this experiment, the adversary generates all the parameters and the secret keys of the PACBS scheme and he chooses the values which he wishes to be challenged on. A PACBS.Sign protocol is executed with these values and the goal of the adversary

Algorithm 3.9: $\text{PASignExp}_{\mathcal{A},\Pi}$

Input : security parameter λ
Output: $\{0,1\}$

$(\mathsf{prms},\mathsf{pk},\mathsf{sk}) \leftarrow \mathcal{A}(\textbf{find}, 1^\lambda)$
$(d, m) \leftarrow \mathcal{A}(\mathsf{prms},\mathsf{pk},\mathsf{sk})$
$(\overline{\sigma}_b, \pi_{\mathsf{Sign}}) \leftarrow \mathsf{PACBS.Sign}\langle\mathcal{A}(\mathsf{sk}_\mathsf{S}),\mathcal{U}(m),(\mathsf{prms},\mathsf{pk},d)\rangle$
$(\text{result}, \pi_{\mathsf{Verify}}) \leftarrow \mathsf{PACBS.Verify}(\mathsf{prms},\mathsf{pk},\mathsf{sk}_\mathsf{V}, m, \overline{\sigma}_b)$
if $\mathsf{PACBS.AuditSign}(\mathsf{prms},\mathsf{pk},\pi_{\mathsf{Sign}}) = 1$ AND $\text{result} \neq \mathsf{pred}(d)$ **then**
  return 1
else
  return 0
end

is to output a signature and evidence, such that PACBS.AuditSign accepts and the signature validity is different from the first output of the algorithm PACBS.Verify.

In the case of PACBS.AuditVrfy the aim is to ensure that when the designated verifier reveals the validity of a signature the result is accurate with respect to PACBS.Verify. This is necessary since the recipient of the signature does not know the value of the predicate when the inputs are randomly chosen. For this, the PAVrfyExp experiment is used, defined in Algorithm 3.10.

Algorithm 3.10: $\text{PAVrfyExp}_{\mathcal{A},\Pi}$

Input : security parameter λ
Output: $\{0,1\}$

$(\mathsf{prms},\mathsf{pk},\mathsf{sk}) \leftarrow \mathcal{A}(\textbf{find}, 1^\lambda)$
$(m, \overline{\sigma}_b, \pi_{\mathsf{Verify}}, b) \leftarrow \mathcal{A}(\mathsf{prms},\mathsf{pk},\mathsf{sk}_\mathsf{V})$
$(\text{result}, \cdot) \leftarrow \mathcal{A}^{\mathsf{PACBS.Verify}}(\mathsf{prms},\mathsf{pk},\mathsf{sk}_\mathsf{V}, m, \overline{\sigma}_b)$
if $\mathsf{PACBS.AuditVrfy}(\mathsf{prms},\mathsf{pk}, b, \overline{\sigma}_b, \pi_{\mathsf{Verify}}) = 1$ AND $\text{result} \neq b$ **then**
  return 1
else
  return 0
end

In PAVrfyEXP the adversary is given all the parameters and the secret keys of the PACBS scheme and his goal is to output a message, a signature and evidence such that PACBS.AuditVrfy accepts b as the validity of the signature while PACBS.Verify outputs $1 - b$.

Definition 3.9: PACBS Public Auditability

A publicly auditable conditional blind signature scheme Π is publicly auditable if for every PPT $\mathcal{A}$ there is a negligible function of λ such that

$$\Pr\left[\text{PASignExp}_{\mathcal{A},\Pi}(\lambda) = 1\right] + \Pr\left[\text{PAVrfyExp}_{\mathcal{A},\Pi}(\lambda) = 1\right] \leq \text{negl}(\lambda)$$

3.6 Okamoto-Schnorr PACBS construction

In this section, a construction for a PACBS scheme is presented. It extends the Okamoto-Schnorr CBS scheme (section 3.2), using the variations presented in section 3.4. In particular, since this construction is meant to be used as a building block in a coercion resistant electronic voting scheme, there is a benefit in reducing the rounds of interaction between the user (voter) and the election authorities. As a result, the presented construction is built on the reduced round OSCBS_r, where the first part of the issued signature is encrypted using ElGamal [Gam85] encryption and the verifier can issue signatures. However, this is not necessary. A PACBS construction could be built on any of the variations of CBS presented in section 3.2 or in section 3.4. As a proof of concept, another PACBS construction is presented in section 3.8, which is a direct extension of CBS and where the signer and the verifier do not share a key.

The PACBS scheme works in a group $\mathbb{G}$ of prime order q ($2^{\lambda-1} \leq q < 2^{\lambda}$), where the DDH assumption holds. During the parameter generation phase random group elements (g_1, g_2, v, h_1) are selected. The signature message space consists of pairs of group elements (ElGamal ciphertexts).

In order to make the signature conditional to public data, a function embed is used that implicitly inserts a value that acts as the 'secret bit':

$$\text{embed} : (\mathbb{G}^2 \times \mathbb{G}^2) \to \mathbb{G}^2$$

where:

$$\text{embed}(\boldsymbol{C}_1, \boldsymbol{C}_2) := (\boldsymbol{C}_2/\boldsymbol{C}_1)^{\alpha} \cdot \text{Enc}_h(1, \gamma) \tag{3.2}$$

The values $\alpha, \gamma \in \mathbb{Z}_q^*$ are blinding factors selected by the signer. The predicate pred is defined as:

$$\text{pred} : \mathbb{G}^2 \times \mathbb{G}^2 \to \{0, 1\}$$

where:

$$\text{pred}(\boldsymbol{C}_1, \boldsymbol{C}_2) := \begin{cases} 1, \text{ if } \text{Dec}_z(\boldsymbol{C}_1) = \text{Dec}_z(\boldsymbol{C}_2) \\ 0, \text{ otherwise} \end{cases} \tag{3.3}$$

The predicate function receives some group elements and some auxiliary information and checks that they are equal to the values embedded inside the signature, which means that the ciphertexts are equal. Note that in both cases there is no restriction in the amount of public and auxiliary information to be used. For simplicity and to correspond with the usage scenario in section 5.1 two pairs of group elements were chosen for the exposition.

3.6.1 OSPACBS parameter generation

The parameter generation algorithm selects the appropriate group generators and instantiates the predicate functions, as described in the previous section. A signing key $s \in \mathbb{Z}_q$ and a decryption key $z \in \mathbb{Z}_q$ are also selected. These secret keys are collectively denoted as sk. Note that s plays the role of the secret signing key sk_{S}, while the tuple (s, z) plays the role of sk_{V} in Definition 3.5. The corresponding public keys are $k := g_1^s$ and $h := h_1^z$, denoted as pk. Two random oracles $\mathsf{H}_1 : \mathbb{G}^4 \times \mathbb{G} \to \mathbb{G}$, $\mathsf{H}_2 : m \times \mathbb{G} \to \mathbb{Z}_q$ are assumed.

Algorithm 3.11: OSPACBS.Gen Algorithm

Input : security parameter λ
Output: sk, pk, prms

$(g_1, g_2, v, h_1) \leftarrow_\$ \mathbb{G}$
$s \leftarrow_\$ \mathbb{Z}_q$
$z \leftarrow_\$ \mathbb{Z}_q$
$k := g_1^s$
$h := h_1^z$
$\mathsf{pred}(\boldsymbol{C}_1, \boldsymbol{C}_2) := \begin{cases} 1, \text{ if } \mathsf{Dec}_z(\boldsymbol{C}_1) = \mathsf{Dec}_z(\boldsymbol{C}_2) \\ 0, \text{ otherwise} \end{cases}$
$\mathsf{prms} := (q, \mathbb{G}, g_1, g_2, v, \mathsf{pred}, \mathsf{H}_1, \mathsf{H}_2)$
$\mathsf{sk} := (s, z)$
$\mathsf{pk} := (k, h)$
return $(\mathsf{prms}, \mathsf{sk}, \mathsf{pk})$

Note that the validity of the signature is based on the value of the predicate, regardless of how it was constructed and embedded.

3.6.2 OSPACBS signing

The PACBS signing protocol is shown in Figure 3.4. Note that each algorithm in the protocol is explicitly named for easier reference.

NIZK proofs for signing The proof π_1 is generated as a standard Chaum-Pedersen [CP92] proof for valid encryption. The proof π_3 is generated as a composition (cf.

Common input: prms, pk, $\boldsymbol{C}_1, \boldsymbol{C}_2 \in \mathbb{G}^2$
$\mathcal{U}$'s private input: $m \in \mathbb{M}$
S's private input : $\mathsf{sk}_\mathsf{S} = s \in \mathbb{Z}_q$ s.t. $k = g_1^s$

$\mathcal{U}$ executes the OSPACBS.Blind Algorithm on input $(\mathsf{prms}, \boldsymbol{C}_1, \boldsymbol{C}_2, m)$:

- Compute $x := \mathsf{H}_1(\boldsymbol{C}_1, \boldsymbol{C}_2)$;
- Pick $u_1, u_2, d \leftarrow_\$ \mathbb{Z}_q$ and compute $x^* := x g_1^{u_1} g_2^{u_2} v^d$, $e^* := \mathsf{H}_2(m, x^*)$ and $e := e^* - d$;
- Send e to S.

S executes the OSPACBS.BlindSign Algorithm on input $(\mathsf{prms}, \mathsf{sk}_\mathsf{S}, \boldsymbol{C}_1, \boldsymbol{C}_2, e)$:

- Compute $x := \mathsf{H}_1(\boldsymbol{C}_1, \boldsymbol{C}_2)$;
- Pick $y_2 \leftarrow_\$ \mathbb{Z}_q$ and compute $n := x g_2^{-y_2} v^{-e}$;
- Pick $t \leftarrow_\$ \mathbb{Z}_q$ and compute $\boldsymbol{N} := \mathsf{Enc}_h(n; t)$;
- Pick blinding factors $\alpha, \gamma \in \mathbb{Z}_q^*$ and compute $\boldsymbol{W} := \mathsf{embed}(\boldsymbol{C}_1, \boldsymbol{C}_2) = (\boldsymbol{C}_2/\boldsymbol{C}_1)^\alpha \cdot \mathsf{Enc}_h(1, \gamma)$ and apply signing key to compute $\boldsymbol{B} := (\boldsymbol{N} \cdot \boldsymbol{W})^s$ with:

$$\begin{aligned} \pi_1 &\leftarrow \mathsf{NIZK}\{(h_1, h, n, \boldsymbol{N}), (t) : \boldsymbol{N} = \mathsf{Enc}_h(n; t)\} \\ \pi_2 &\leftarrow \mathsf{NIZK}\{(\boldsymbol{C}_1, \boldsymbol{C}_2, \boldsymbol{W}), (\alpha, \gamma) : \boldsymbol{W} = (\boldsymbol{C}_2/\boldsymbol{C}_1)^\alpha \cdot \mathsf{Enc}_h(1; \gamma)\} \\ \pi_3 &\leftarrow \mathsf{NIZK}\{(h, k, \boldsymbol{N}, \boldsymbol{W}, \boldsymbol{B}), (s) : \boldsymbol{B} = (\boldsymbol{N} \cdot \boldsymbol{W})^s \text{ AND } k = g_1^s\} \end{aligned}$$

- Set $\overline{\beta} := (((n, \boldsymbol{N}, \boldsymbol{W}, \boldsymbol{B}), y_2), \pi_1, \pi_2, \pi_3)$ and send $\overline{\beta}$ to the $\mathcal{U}$.

$\mathcal{U}$ executes the OSPACBS.Unblind Algorithm on input $(\mathsf{prms}, \mathsf{pk}_\mathsf{S}, m, \overline{\beta}, u_1, u_2, x^*, e^*)$:

- Verify π_1, π_2, π_3;
- Unblind by computing $\overline{\sigma}_\mathbf{1} := \boldsymbol{B} \cdot \mathsf{Enc}_h(k^{u_1})$ and $\sigma_2 := y_2 + u_2$.
- Set $\overline{\sigma} := (x^*, e^*, \overline{\sigma}_\mathbf{1}, \sigma_2)$ and output $(m, \overline{\sigma})$.

FIGURE 3.4: OSPACBS.Sign Protocol

$\pi_2 = \mathsf{NIZK}\{(\boldsymbol{C}_1, \boldsymbol{C}_2, \boldsymbol{W}), (\alpha, \gamma) : \boldsymbol{W} = (\boldsymbol{C}_2/\boldsymbol{C}_1)^{\alpha} \cdot \mathsf{Enc}_h(1;\gamma)\}$

Prover		**Verifier**
$\alpha', \gamma' \leftarrow_\$ \mathbb{Z}_q$		
$\boldsymbol{W}' := (\boldsymbol{C}_2/\boldsymbol{C}_1)^{\alpha'} \mathsf{Enc}_h(1, \gamma')$		
	$\boldsymbol{W}' \longrightarrow$	
		$c \leftarrow_\$ \mathbb{Z}_q$
		or in the random oracle model
		$c := \mathcal{H}(\boldsymbol{C}_1, \boldsymbol{C}_2, \boldsymbol{W}, \boldsymbol{W}')$
	$\longleftarrow c$	
$\alpha'' := \alpha c + \alpha'$		
$\gamma'' := \gamma c + \gamma'$		
	$\alpha'', \gamma'' \longrightarrow$	
		Accept if:
		$\alpha' \neq \alpha''$ AND
		$\gamma' \neq \gamma''$ AND
		$\boldsymbol{W}^c \boldsymbol{W}' = (\boldsymbol{C}_2/\boldsymbol{C}_1)^{\alpha''} \mathsf{Enc}_h(1, \gamma'')$

Figure 3.5: The proof π_2 in OSPACBS.Sign

[CDS94]) of a Schnorr proof [Sch91] for the relation $k = g_1^s$ and Chaum-Pedersen proof for knowledge and equality of discrete log s for the relation $\boldsymbol{B} = (\boldsymbol{N} \cdot \boldsymbol{W})^s$. The techniques to construct them are detailed in section 2.4.1.

The proof π_2 is generated as detailed in Figure 3.5 (cf. [Gro05]).

Theorem 3.5: Properties of π_2

The protocol described for proof of knowledge π_2 is a Σ-protocol.

Proof. Completeness is straightforward.

For special soundness, assume two valid interactions $(\boldsymbol{W}', c, \alpha'', \gamma'')$ and $(\boldsymbol{W}', \tilde{c}, \tilde{\alpha''}, \tilde{\gamma''})$, with $c \neq \tilde{c}$:

The witness (α, γ) can be extracted by setting: $\alpha = (\alpha'' - \tilde{\alpha''})(c - \tilde{c})^{-1}$ and $\gamma = (\gamma'' - \tilde{\gamma''})(c - \tilde{c})^{-1}$.

Indeed:

$$(\boldsymbol{C}_2/\boldsymbol{C}_1)^{(\alpha''-\tilde{\alpha''})(c-\tilde{c})^{-1}} \cdot \mathsf{Enc}_h(1;(\gamma''-\tilde{\gamma''})(c-\tilde{c})^{-1}) =$$
$$(\boldsymbol{C}_2/\boldsymbol{C}_1)^{(\alpha c+\alpha'-\alpha\tilde{c}-\alpha')(c-\tilde{c})^{-1}} \cdot \mathsf{Enc}_h(1;(\gamma c+\gamma'-\gamma\tilde{c}-\gamma'))(c-\tilde{c})^{-1}) =$$
$$(\boldsymbol{C}_2/\boldsymbol{C}_1)^{\alpha} \cdot \mathsf{Enc}_h(1;\gamma) = \boldsymbol{W}$$

For honest verifier zero-knowledge it is easy to see that the distributions $(\boldsymbol{W}', c, \alpha'', \gamma'')$ where $\boldsymbol{W}' = (\boldsymbol{C}_2/\boldsymbol{C}_1)^{\alpha'}\mathsf{Enc}_h(1;\gamma')$ for uniformly distributed α', γ', c from $\mathbb{Z}_q$ and $\alpha'' = \alpha c + \alpha'$, $\gamma'' = \gamma c + \gamma'$ and $((\boldsymbol{C}_2/\boldsymbol{C}_1)^{\alpha''}\mathsf{Enc}_h(1;\gamma'')\boldsymbol{W}^{-c}, c, \alpha'', \gamma'')$ for uniformly distributed α'', γ'', c from $\mathbb{Z}_q^*$ are identical. ■

The proofs π_1, π_2, π_3 can be AND combined into a single proof, detailed in section 3.6.3.

OSPACBS auditing for signing The PACBS.AuditSign process (Algorithm 3.12) is straightforward for the Okamoto-Schnorr instantiation. The auditor needs to verify the proofs issued by the signer.

Algorithm 3.12: OSPACBS.AuditSign Algorithm

Input : prms, pk, Trans = $((\boldsymbol{C}_1, \boldsymbol{C}_2), n, \boldsymbol{B}, \boldsymbol{N}, \boldsymbol{W}, y_2, \pi_1, \pi_2, \pi_3)$
Output: $d \in \{0,1\}$

if $n \neq \mathsf{H}_1(\boldsymbol{C}_1, \boldsymbol{C}_2) g_2^{-y_2} v^{-e}$ **then**
  return 0
end
if π_1, π_2, π_3 are valid **then**
  return 1
else
  return 0
end

3.6.3 OSPACBS verification

The OSPACBS verification procedure is given in Algorithm 3.13. The verifier V, given a message, a signature, and a secret key, outputs whether the signature is valid or not and provides NIZK proofs that the verification operations were done correctly. In more detail, the verifier computes the verification equation and checks if it matches the first part of the signature, by blindingly dividing them. If the signature is valid, the result will decrypt to 1. In any other case the result will be random.

The proofs $\pi_1, \pi_2, \pi_3, \pi_4$ standard Chaum - Pedersen [CP92]. Their construction is detailed in section 2.4.1. Efficiency improvements can also be achieved from their AND combination detailed in section 3.6.3.

Algorithm 3.13: OSPACBS.VerifyAlgorithm

Input : $\mathsf{prms}, \mathsf{pk}, \mathsf{sk}, m, \overline{\sigma} = (x^*, e^*, \overline{\sigma_1}, \sigma_2)$
Output: $d \in \{0,1\}, \pi_{\mathsf{Verify}}$

if $\mathsf{H}_2(m, x^*) \neq e^*$ **then**
  return $\perp$
end
$\gamma \leftarrow_{\$} \mathbb{Z}_q$
validity $:= x^* \cdot g_2^{-\sigma_2} \cdot v^{-e^*}$
$\boldsymbol{M} := \mathsf{Enc}_h(\text{validity}; r_1)$
$\boldsymbol{V} := \boldsymbol{M}^s$
$\boldsymbol{R} := \left(\frac{\boldsymbol{V}}{\overline{\sigma_1}}\right)^{\gamma}$
result $:= \mathsf{Dec}_z(\boldsymbol{R})$

$\pi_1 \leftarrow \mathsf{NIZK}\left\{(h_1, h, \boldsymbol{M}, \text{validity}), (r_1) : \boldsymbol{M} = \mathsf{Enc}_h(\text{validity}; r_1)\right\}$
$\pi_2 \leftarrow \mathsf{NIZK}\left\{(\boldsymbol{V}, \boldsymbol{M}), (s) : \boldsymbol{V} = \boldsymbol{M}^s\right\}$
$\pi_3 \leftarrow \mathsf{NIZK}\left\{(\boldsymbol{V}, \overline{\boldsymbol{\sigma}_1}, \boldsymbol{R}), (\gamma) : \boldsymbol{R} = \left(\frac{\boldsymbol{V}}{\overline{\sigma_1}}\right)^{\gamma}\right\}$
$\pi_4 \leftarrow \mathsf{NIZK}\left\{(h_1, h, \text{result}, \boldsymbol{R}), (z) : \text{result} = \mathsf{Dec}_z(\boldsymbol{R})\right\}$

$d := (\text{result} = 1)$
$\pi_{\mathsf{Verify}} := (\text{validity}, \boldsymbol{M}, \boldsymbol{V}, \boldsymbol{R}, \text{result}, \pi_1, \pi_2, \pi_3, \pi_4)$
return $(d, \pi_{\mathsf{Verify}})$

OSPACBS auditing for verification Finally the PACBS.AuditVrfy procedure is presented in Algorithm 3.14.

Algorithm 3.14: OSPACBS.AuditVrfy Algorithm

Input : $\mathsf{prms}, \mathsf{pk}, m, \overline{\sigma} = (x^*, e^*, \overline{\sigma_1}, \sigma_2)$,
$\pi_{\mathsf{Verify}} = (\text{validity}, \boldsymbol{M}, \boldsymbol{V}, \boldsymbol{R}, \text{result}, \pi_1, \pi_2, \pi_3, \pi_4)$
Output: $d \in \{0,1\}$

if $\mathsf{H}_2(m, x^*) \neq e^*$ OR validity $\neq x^* \cdot g_2^{-\sigma_2} \cdot v^{-e^*}$ **then**
  return 0
end
if $\pi_1, \pi_2, \pi_3, \pi_4$ are valid **then**
  return 1
else
  return 0
end

Note that the scheme is auditable by everyone, meaning that everyone can check the actions of the verifier. However, if the auditor has knowledge of the conditional information, then she can also check that the predicate was correctly computed and checked. This is the key property that will utilized in the design of protocols around this primitive.

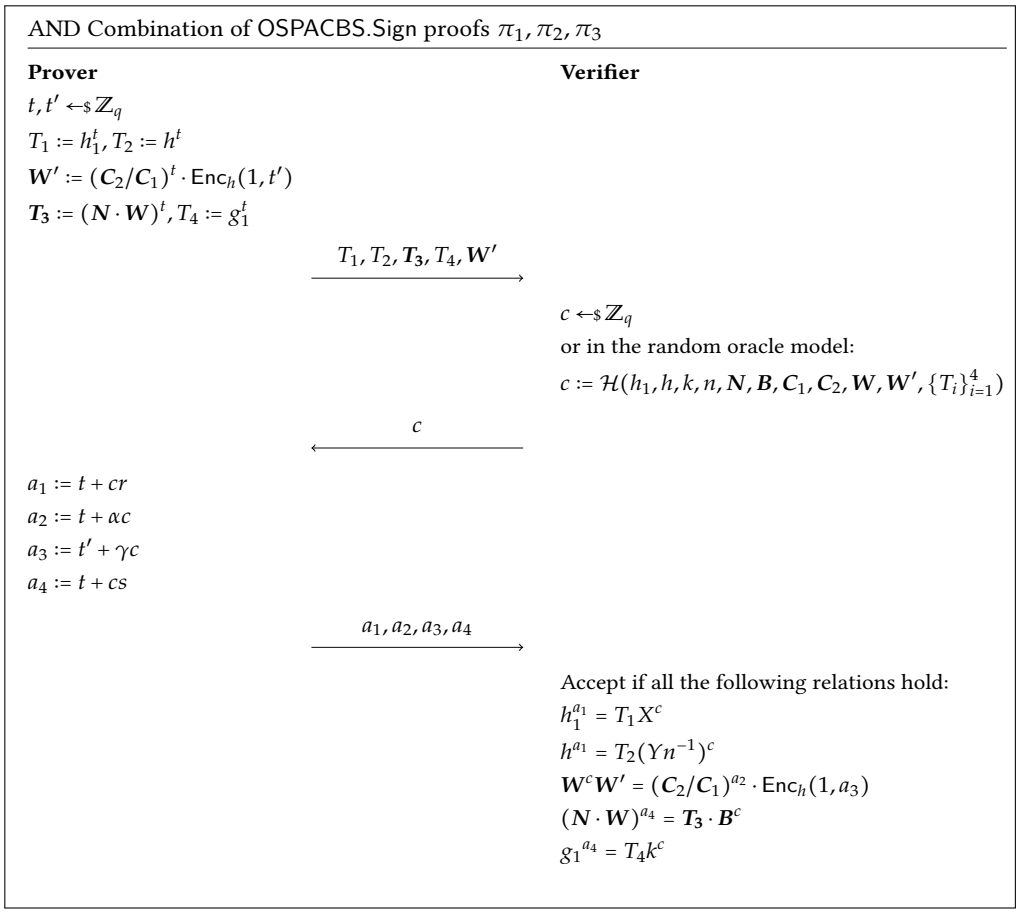


FIGURE 3.6: AND(π_1, π_2, π_3) in OSPACBS.Sign

Performance The performance requirements of PACBS are more extensive due to the increased security guarantees it provides, both on the back end as well as on the front end (user). Signing (OSPACBS.Sign) requires in total 15 exponentiations - 8 for the main operation and 7 for the generation of π_1, π_2, π_3. Verification(OSPACBS.Verify) requires in total 15 exponentiations - 7 for the main operation and 8 for the generation of $\pi_1, \pi_2, \pi_3, \pi_4$. The auditing of the signature generation costs 12 exponentiations, while the auditing of the signature verification costs 18 exponentiations. Minor performance improvements can be achieved from the AND compositions of the proofs.

AND composition for proofs in OSPACBS.Sign and OSPACBS.Verify

The AND composition of π_1, π_2, π_3 from Figure 3.4 is described in Figure 3.6 where:

$$\pi_1 = \mathsf{NIZK}\left\{(h_1, h, n, \boldsymbol{N}), (r) : \boldsymbol{N} = \mathsf{Enc}_h(n; r) = (X, Y) = (h_1^r, nh^r)\right\}$$
$$\pi_2 = \mathsf{NIZK}\left\{(\boldsymbol{C}_1, \boldsymbol{C}_2, \boldsymbol{W}), (\alpha, \gamma) : \boldsymbol{W} = (\boldsymbol{C}_2/\boldsymbol{C}_1)^\alpha \cdot \mathsf{Enc}_h(1; \gamma)\right\}$$
$$\pi_3 = \mathsf{NIZK}\left\{(h, k, \boldsymbol{N}, \boldsymbol{W}, \boldsymbol{B}), (s) : \boldsymbol{B} = (\boldsymbol{N} \cdot \boldsymbol{W})^s \ \text{ AND } \ k = g_1^s\right\}$$

The AND combination of $\pi_1, \pi_2, \pi_3, \pi_4$ from Algorithm 3.13 is straightforward (Figure 3.7). Efficiency improvements can be gained by reusing T_1 from π_1 in π_4. Recall that:

$$\pi_1 = \mathsf{NIZK}\Big\{(h_1, h, \boldsymbol{M}, \text{validity}), (r_1) : \boldsymbol{M} = \mathsf{Enc}_h(\text{validity}; r_1) = (A_1, B_1)\Big\}$$
$$\pi_2 = \mathsf{NIZK}\Big\{(\boldsymbol{V}, \boldsymbol{M}), (s) : \boldsymbol{V} = \boldsymbol{M}^s\Big\}$$
$$\pi_3 = \mathsf{NIZK}\Big\{(\boldsymbol{V}, \overline{\boldsymbol{\sigma}}_{\boldsymbol{1}}, \boldsymbol{R}), (\gamma) : \boldsymbol{R} = \Big(\frac{\boldsymbol{V}}{\overline{\boldsymbol{\sigma}}_{\boldsymbol{1}}}\Big)^{\gamma}\Big\}$$
$$\pi_4 = \mathsf{NIZK}\Big\{(h_1, h, \text{result}, \boldsymbol{R} = (A_4, B_4)), (z) : \text{result} = \mathsf{Dec}_z(\boldsymbol{R})\Big\}$$

AND Combination of OSPACBS.Verify proofs $\pi_1, \pi_2, \pi_3, \pi_4$

Prover		**Verifier**
$t \leftarrow\$ \mathbb{Z}_q$		
$T_1 := h_1^t, T_2 := h^t$		
$\boldsymbol{T_3} := \boldsymbol{M}^t$		
$\boldsymbol{T_4} := (\frac{\boldsymbol{V}}{\overline{\boldsymbol{\sigma}}_{\boldsymbol{1}}})^t$		
$T_5 := A_4^t$		
	$T_1, T_2, \boldsymbol{T_3}, \boldsymbol{T_4}, T_5 \longrightarrow$	
		$c \leftarrow\$ \mathbb{Z}_q$
		or in the random oracle model:
		$c := \mathcal{H}(h_1, h, \boldsymbol{M}, \text{validity}, \boldsymbol{V}, \overline{\boldsymbol{\sigma}}_{\boldsymbol{1}}, \boldsymbol{R}, \text{result}, \{T_i\}_{i=1}^5)$
	$\longleftarrow c$	
$a_1 := t + cr_1$		
$a_2 := t + cs$		
$a_3 := t + c\gamma$		
$a_4 := t + cz$		
	$a_1, a_2, a_3, a_4 \longrightarrow$	
		Accept if all the following relations hold:
		$h_1^{a_1} = T_1 A_1^c$
		$h^{a_1} = T_2(B_1 \cdot \text{validity}^{-1})^c$
		$\boldsymbol{M}^{a_2} = \boldsymbol{T_3}\boldsymbol{V}^c$
		$(\frac{\boldsymbol{V}}{\overline{\boldsymbol{\sigma}}_{\boldsymbol{1}}})^{a_3} = \boldsymbol{T_4} \cdot \boldsymbol{R}^c$
		$h_1^{a_4} = T_1 h^c$
		$A_4^{a_4} = T_5(B_4 \cdot \text{result}^{-1})^c$

FIGURE 3.7: $\text{AND}(\pi_1, \pi_2, \pi_3, \pi_4)$ in OSPACBS.Verify

3.7 PACBS Security analysis

3.7.1 Correctness

The predicate is invariant in the algorithms that comprise the PACBS scheme.

Lemma 3.2

Let $\overline{\beta} = (((n, \boldsymbol{B}, \boldsymbol{N}, \boldsymbol{W}), y_2), \pi_1, \pi_2, \pi_3)$ and $x = \mathsf{H}_1(\boldsymbol{C}_1, \boldsymbol{C}_2)$ be the output of OSPACBS.BlindSign algorithm executed by S in the OSPACBS.Sign protocol. Then:

$$\mathsf{Dec}_z(\boldsymbol{B}) = x^s g_2^{-y_2 s} v^{-es} \Leftrightarrow \mathsf{pred}(\boldsymbol{C}_1, \boldsymbol{C}_2) = 1$$

Proof. The result follows from straightforward computations and the homomorphic properties of the underlying encryption scheme:

$$\begin{aligned}
\mathsf{Dec}_z(\boldsymbol{B}) &= x^s g_2^{-y_2 s} v^{-es} \Leftrightarrow \\
\mathsf{Dec}_z((\boldsymbol{NW})^s) &= x^s g_2^{-y_2 s} v^{-es} \Leftrightarrow \\
\mathsf{Dec}_z(\boldsymbol{N}^s)\mathsf{Dec}_z(\boldsymbol{W})^s &= x^s g_2^{-y_2 s} v^{-es} \Leftrightarrow \\
n^s \mathsf{Dec}_z(\boldsymbol{W})^s &= x^s g_2^{-y_2 s} v^{-es} \Leftrightarrow \\
x^s g_2^{-y_2 s} v^{-es} \mathsf{Dec}_z(\boldsymbol{W})^s &= x^s g_2^{-y_2 s} v^{-es} \Leftrightarrow \\
\mathsf{Dec}_z(\boldsymbol{W})^s &= 1 \Leftrightarrow \\
\mathsf{Dec}_z(\boldsymbol{C}_2/\boldsymbol{C}_1)^{as} &= 1
\end{aligned}$$

Now since $a \neq 0$, $\mathsf{Dec}_z(\boldsymbol{C}_2/\boldsymbol{C}_1)^{as} = 1 \Leftrightarrow \mathsf{Dec}_z(\boldsymbol{C}_2/\boldsymbol{C}_1) = 1 \Leftrightarrow \mathsf{Dec}_z(\boldsymbol{C}_2) = \mathsf{Dec}_z(\boldsymbol{C}_1)$ which gives the result. ∎

Lemma 3.3

Let $(x^*, e^*, \overline{\sigma}_{\boldsymbol{1}}, \sigma_2)$ be $\mathcal{U}$'s output of the OSPACBS.Sign protocol on message m and on predicate input $(\boldsymbol{C}_1, \boldsymbol{C}_2)$. Then it holds that

$$\mathsf{Dec}_z(\overline{\sigma}_{\boldsymbol{1}}) = x^{*s} g_2^{-\overline{\sigma}_2 s} v^{-e^* s} \Leftrightarrow \mathsf{pred}(\boldsymbol{C}_1, \boldsymbol{C}_2) = 1$$

Proof. Following the protocol description in Figure 3.4:

$$\begin{aligned}
\mathsf{Dec}_z(\overline{\sigma}_{\boldsymbol{1}}) &= x^{*s} g_2^{-\overline{\sigma}_2 s} v^{-e^* s} \Leftrightarrow \\
\mathsf{Dec}_z(\boldsymbol{B}\mathsf{Enc}_h(k_1^u)) &= x^s g_1^{u_1 s} g_2^{u_2 s} v^{ds} g_2^{-y_2 s - u_2 s} v^{-es-ds} \Leftrightarrow \\
\mathsf{Dec}_z(\boldsymbol{B}) k_1^u &= x^s k^{u_1} g_2^{-y_2 s} v^{-es} \Leftrightarrow \\
\mathsf{Dec}_z(\boldsymbol{B}) &= x^s g_2^{-y_2 s} v^{-es}
\end{aligned}$$

and from Lemma 3.2 it holds that $\mathsf{Dec}_z(\boldsymbol{B}) = x^s g_2^{-y_2 s} v^{-es} \Leftrightarrow \mathsf{pred}(\boldsymbol{C}_1, \boldsymbol{C}_2) = 1$. ∎

Using Lemma 3.3 correctness can be proved.

Theorem 3.6: OSPACBS correctness

The Okamoto-Schnorr PACBS scheme has correctness.

Proof. It always holds that $\mathsf{H}_2(m, x^*) = e^*$. Following Lemma 3.3 the PACBS.Verify algorithm outputs 1 if and only if $\mathsf{Dec}_z(\boldsymbol{R}) = 1$ and since the blinding factor $\gamma \neq 0$:

$$\begin{aligned}
\mathsf{Dec}_z(\boldsymbol{R}) = 1 &\Leftrightarrow \mathsf{Dec}_z(\boldsymbol{V}/\overline{\sigma}_{\boldsymbol{1}}) = 1 \\
&\Leftrightarrow \mathsf{Dec}_z(\boldsymbol{M}^s) = \mathsf{Dec}_z(\overline{\sigma}_{\boldsymbol{1}}) \\
&\Leftrightarrow \mathsf{Dec}_z(\mathsf{Enc}_h(\text{validity})^s) = \mathsf{Dec}_z(\overline{\sigma}_{\boldsymbol{1}}) \\
&\Leftrightarrow \text{validity}^s = \mathsf{Dec}_z(\overline{\sigma}_{\boldsymbol{1}}) \\
&\Leftrightarrow x^{*s} g_2^{-\overline{\sigma}_2 s} v^{-e^* s} = \mathsf{Dec}_z(\overline{\sigma}_{\boldsymbol{1}}) \\
&\Leftrightarrow \mathsf{pred}(\boldsymbol{C}_1, \boldsymbol{C}_2) = 1
\end{aligned}$$

which concludes the proof. ■

3.7.2 Blindness

The proof follows [Sch01]. Note that π_{Verify} in Algorithm 3.13 depends solely on the signature.

Lemma 3.4

Let $\overline{\beta} = (((n, \boldsymbol{B}, \boldsymbol{N}, \boldsymbol{W}), \mathrm{y}_2), \pi_1, \pi_2, \pi_3)$ be an output of the OSPACBS.Sign protocol with public transcript $(e, \overline{\beta}, \overline{\sigma})$ and public input $x = \mathsf{H}_1(C_1, C_2)$ where $\overline{\sigma} = (x^*, e^*, \overline{\boldsymbol{\sigma}}_{\boldsymbol{1}}, \sigma_2)$ is a valid signature on message m. Then there exist a unique tuple (u_1, u_2, d, r) such that $\mathsf{Unblind}(\overline{\beta}, u_1, u_2, d, r) = \overline{\sigma}$ where Unblind is the algorithm issued as the last step of the OSPACBS.Sign protocol, u_1, u_2, d are the blinding factors and r is the randomness used to encrypt k^{u_1}.

Proof. Since $\overline{\sigma}$ is valid, Figure 3.4 yields about $\boldsymbol{B}$:

$$\boldsymbol{B} = \mathsf{Enc}_h(x^s g_2^{-y_2 s} v^{-es} \cdot 1) \Rightarrow \mathsf{Dec}_z(\boldsymbol{B}) = x^s g_2^{-y_2 s} v^{-es} \tag{3.4}$$

Furthermore, the validity of $\overline{\sigma}$ means that in OSPACBS.Verify (Algorithm 3.13):

$$\overline{\sigma}_{\boldsymbol{1}} = \boldsymbol{M}^s = \mathsf{Enc}_h(x^{*s} g_2^{-\overline{\sigma}_2 s} v^{-e^* s}) \tag{3.5}$$

It is immediate from the Blind and Unblind algorithms in Figure 3.4 that the only possible tuple (u_1, u_2, d, r) for a valid signature must satisfy:

$$u_1 = \log_k(\mathsf{Dec}_z(\overline{\sigma}_{\mathbf{1}}) \cdot \mathsf{Dec}_z(\boldsymbol{B}^{-1})) \tag{3.6}$$

and $u_2 = \sigma_2 - y_2$, $d = e^* - e$, $r = r_{\overline{\sigma}_{\mathbf{1}}} - r_{\boldsymbol{B}}$, where $r_{\boldsymbol{B}}$ is the encryption randomness of $\boldsymbol{B}$ and $r_{\overline{\sigma}_{\mathbf{1}}}$ the randomness in $\overline{\sigma}_{\mathbf{1}}$.

The value $\tilde{x^*}$ computed when unblinding with these values equals x^*. From Equation 3.4 and Equation 3.6:

$$g_1^{u_1} = k^{u_1 s^{-1}} = \mathsf{Dec}_z(\overline{\sigma}_{\mathbf{1}})^{s^{-1}} \mathsf{Dec}_z(\boldsymbol{B}^{-1})^{s^{-1}} = \mathsf{Dec}_z(\overline{\sigma}_{\mathbf{1}})^{s^{-1}} x^{-1} g_2^{y_2} v^e$$

From Equation 3.5:
$\tilde{x}^* = x g_1^{u_1} g_2^{u_2} v^d = x(\mathsf{Dec}_z(\overline{\sigma}_{\mathbf{1}})^{s^{-1}} x^{-1} g_2^{y_2} v^e) g_2^{\sigma_2 - y_2} v^{e^* - e} = \mathsf{Dec}_z(\overline{\sigma}_{\mathbf{1}})^{s^{-1}} g_2^{\sigma_2} v^{e^*} = x^*$ ■

As a result:

Theorem 3.7: OSPACBS Blindness

The Okamoto-Schnorr PACBS scheme satisfies perfect blindness.

Proof. Let S^* be the unbounded adversary in the blindness game in Algorithm 3.6 and $\mathsf{view}_i = (e_i, \overline{\beta}_i)$ for $i \in \{0,1\}$ be his view in each case. From Lemma 3.4 it follows that there exists a unique tuple (u_1, u_2, d, r) that maps view_i to $\overline{\sigma}_j$ for both cases of $i, j \in \{0,1\}$ and the unbounded adversary can always compute it. This means that the view of the adversary and the produced signatures are statistically independent. As a result, in the blindness game (Algorithm 3.6) both signatures $\overline{\sigma}_b, \overline{\sigma}_{1-b}$ are perfectly indistinguishable for S^* and his advantage in the PACBS-BlindExp is zero. ■

3.7.3 Unforgeability

Theorem 3.8: OSPACBS Unforgeability

If the OSCBS scheme is l one-more unforgeable then the OSPACBS scheme is l one-more unforgeable under the assumption that no signatures with the same input $\boldsymbol{C}$ to the predicate are requested.

Proof. It will be shown that if there exists an $\mathcal{A}$ that wins the PACBS-OneMoreForge game with non-negligible probability, an algorithm $\mathcal{B}$ can be constructed that wins the CBS-OneMoreForge game with non-negligible probability.

The role of $\mathcal{B}$ will be to simulate a PACBS signer for $\mathcal{A}$, by responding to his requests for signatures without having the signing key s.

If the conversation between $\mathcal{A}$ and $\mathcal{B}$ is indistinguishable from a conversation between $\mathcal{A}$ and a real OSPACBS signer then $\mathcal{A}$ will issue a forgery with non-negligible probability (by assumption). Then $\mathcal{B}$ can utilize it to issue an OSCBS forgery when interacting with the OSCBS signer $\mathcal{S}$.

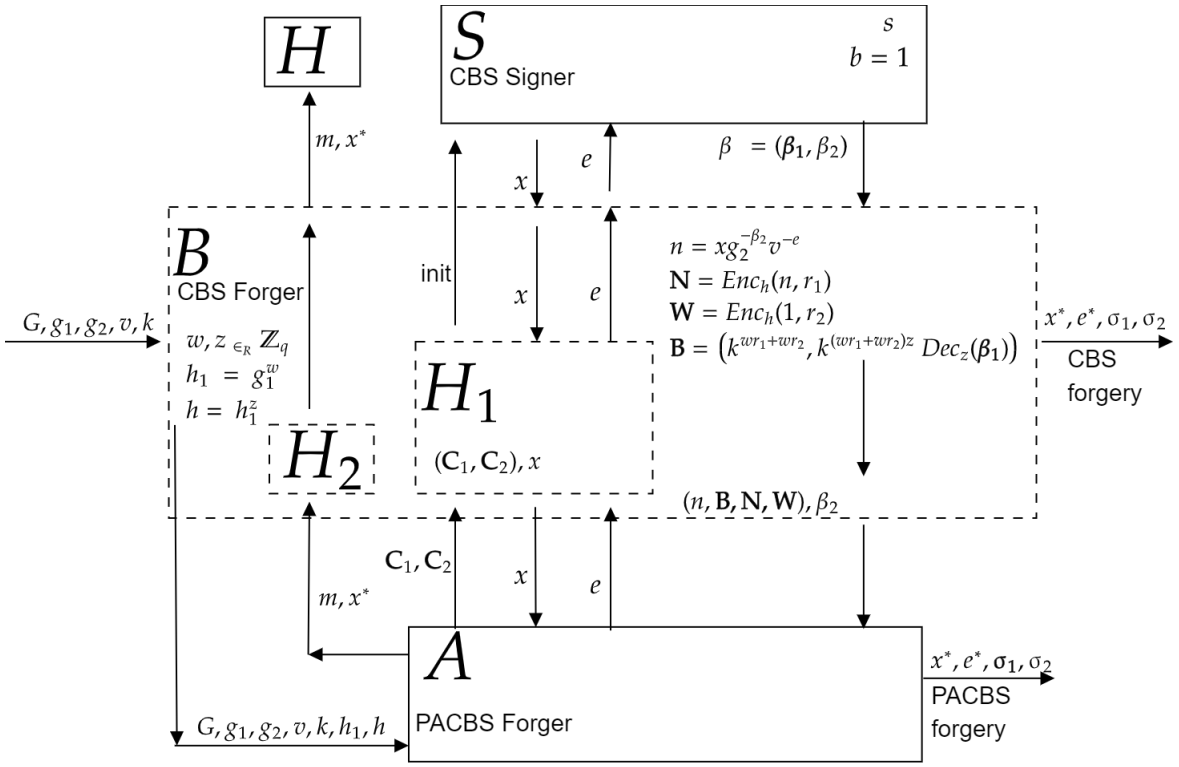


FIGURE 3.8: Forging OSPACBS by using OSCBS

An overview of the reduction is presented in Figure 3.8.

The input of $\mathcal{B}$ will be the public input of OSCBS namely $\mathbb{G}, g_1, g_2, v, k$. The CBS signing oracle S is initialized with s by the challenger in order to be able to answer signing requests. In order to simulate a PACBS Signer, $\mathcal{B}$ generates a random $w \in \mathbb{Z}_q$ and computes $h_1 = g_1^w \in \mathbb{G}$. Moreover, he selects $z \in \mathbb{Z}_q$ and computes $h = h_1^z \in \mathbb{G}$. $\mathcal{B}$ must know the decryption key z in order to be able to check the value of the predicate and properly derive the validity of the signature generated by $\mathcal{A}$. $\mathcal{B}$ initializes $\mathcal{A}$ with $\mathbb{G}, g_1, g_2, v, k, h_1, h$. $\mathcal{A}$ can query the two random oracles $\mathsf{H}_1, \mathsf{H}_2$ and $\mathcal{B}$ with $\boldsymbol{C} = (\boldsymbol{C}_1, \boldsymbol{C}_2)$ of his choice. As a PACBS-signer, $\mathcal{B}$ should be able to answer these requests indistinguishably from a real execution of the experiment. To do so, $\mathcal{B}$ utilizes the random oracle H and the signing interactions with $\mathcal{S}$.

For the H_2 queries that $\mathcal{A}$ makes, $\mathcal{B}$ queries the random oracle H on the same message and forwards the response.

For the H_1 queries that $\mathcal{A}$ makes on input $\boldsymbol{C}$, if $\mathsf{H}_1(\boldsymbol{C})$ is not yet defined, $\mathcal{B}$ can check the validity of the predicate since he can decrypt $\boldsymbol{C} = (\boldsymbol{C}_1, \boldsymbol{C}_2)$ by utilizing z and check if $pred(\boldsymbol{C}) = 1$ or equivalently if $\mathsf{Dec}_z(\boldsymbol{C}_1) = \mathsf{Dec}_z(\boldsymbol{C}_2)$. After the decryption $\mathcal{B}$ proceeds as follows:

- If $\mathsf{pred}(\boldsymbol{C}) = 1$, $\mathcal{B}$ begins an interaction (which he might continue or abort later) with the valid OSCBS signer $\mathcal{S}$ and receives the first message x. He sets $\mathsf{H}_1(\boldsymbol{C}) = x$ and responds to $\mathcal{A}$ with x.
- If $\mathsf{pred}(\boldsymbol{C}) = 0$, $\mathcal{B}$ responds with a uniformly selected element of $\mathbb{G}$.

Without loss of generality, it can be assumed that whenever $\mathcal{A}$ queries $\mathcal{B}$ with $\boldsymbol{C}$ he has queried $\mathsf{H}_1(\boldsymbol{C})$.

To answer the signing queries of $\mathcal{A}$, $\mathcal{B}$ proceeds as follows:

When receiving $\boldsymbol{C}, e$ from $\mathcal{A}$, if $\mathsf{pred}(\boldsymbol{C}) = 1$, $\mathcal{B}$ continues the interaction started with $\mathcal{S}$ when answering $\mathsf{H}_1(\boldsymbol{C})$ by forwarding e.

$\mathcal{B}$ receives an OSCBS signature $\overline{\beta} = (\overline{\boldsymbol{\beta}}_{\mathbf{1}}, \overline{\beta}_2)$ from $\mathcal{S}$ where $\overline{\beta}_2 \leftarrow_{\$} \mathbb{Z}_q$ and $\overline{\boldsymbol{\beta}}_{\mathbf{1}} = \mathsf{Enc}_h((xg_2^{-\overline{\beta}_2}v^{-e})^s)$. Then $\mathcal{B}$ must compute $n, \boldsymbol{N}, \boldsymbol{W}, B$ as specified in Figure 3.4. $\mathcal{B}$ proceeds as follows:

- n is computed as specified by the protocol: $n = xg_2^{-\overline{\beta}_2}v^{-e}$. This is straightforward since all values are known. Indeed, v was provided by the challenger, $x, \overline{\beta}_2$ were provided by S and e by $\mathcal{A}$.
- $\mathcal{B}$ can also compute $n^s = \mathsf{Dec}_z(\overline{\boldsymbol{\beta}}_{\mathbf{1}})$. Note that $\overline{\boldsymbol{\beta}}_{\mathbf{1}} = \mathsf{Enc}_h(n^s)$ (cf. section 3.4 and Figure 3.4). Since $\mathcal{B}$ knows the decryption key, decryption is straightforward.
- $\boldsymbol{N}$ is computed normally as $\boldsymbol{N} = \mathsf{Enc}_h(n, r_1) = (h_1^{r_1}, nh^{r_1}) = (h_1^{r_1}, nh_1^{zr_1})$. $\mathcal{B}$ also generates the proof π_1 in the normal manner, as he knows the randomness used in the encryption.
- $\boldsymbol{W}$ is computed as an encryption of 1 since the signature will be valid. $\mathcal{B}$ selects $r, (\alpha, \gamma)$ and computes $r_2 = \alpha r + \gamma$ and $\boldsymbol{W} = \mathsf{Enc}_h(1, r_2) = (h_1^{r_2}, h^{r_2}) = (h_1^{r_2}, h_1^{zr_2})$ which is a valid reencryption of $\frac{\boldsymbol{C}_2}{\boldsymbol{C}_1} = \mathsf{Enc}_h(1)$. $\mathcal{B}$ simulates the proof π_2.

– To compute $\boldsymbol{B}$ as $(\boldsymbol{NW})^s$, $\mathcal{B}$ must use s, which he does not possess. But he can compute $\boldsymbol{B}$ using k, n^s, r_1 and r_2 as follows:

$$
\begin{aligned}
&(k^{wr_1+wr_2}, k^{(wr_1+wr_2)z} n^s) = \\
&(g_1^{s(wr_1+wr_2)}, g_1^{s(wr_1+wr_2)z} n^s) = \\
&(g_1^{s(wr_1)}, g_1^{s(wr_1)z} n^s)(g_1^{s(wr_2)}, g_1^{s(wr_2)z}) = \\
&(h_1^{sr_1}, h_1^{sr_1 z} n^s)(h_1^{sr_2}, h_1^{sr_2} z) = \\
&(h_1^{sr_1}, h^{sr_1} n^s)(h_1^{sr_2}, h^{sr_2}) = \\
&\mathsf{Enc}_h(n; r_1)^s \mathsf{Enc}_h(1; r_2)^s = (\boldsymbol{NW})^s
\end{aligned}
$$

The proof π_3 is simulated as $\mathcal{B}$ does not possess s, but the equation holds.

If $\mathsf{pred}(\boldsymbol{C}) = 0$, $\mathcal{B}$ does not use $\mathcal{S}$ and must construct an invalid signature for $\mathcal{A}$ on its own. This can be done in the following manner:

– $\mathcal{B}$ randomly selects $\overline{\beta}_2 \in \mathbb{Z}_q$.

– $\mathcal{B}$ computes $n = x g_2^{-\overline{\beta}_2} v^{-e}$.

– $\boldsymbol{N}$ is computed normally as $\boldsymbol{N} = \mathsf{Enc}_h(n, r_1) = (h_1^{r_1}, n h^{r_1}) = (h_1^{r_1}, n h_1^{z r_1})$. π_1 is generated as before.

– $\boldsymbol{W}$ must contain the encryption of a random group element. $\mathcal{B}$ chooses $r_3 \in \mathbb{Z}_q$ and computes $g_1^{r_3} n^{-1}$ for this reason and sets $\boldsymbol{W} = \mathsf{Enc}_h(g_1^{r_3} n^{-1}, r_2) = (h_1^{r_2}, g_1^{r_3} n^{-1} h^{r_2})$.

– $\boldsymbol{B}$ is computed using k as: $(k^{wr_1+wr_2}, k^{(wr_1+wr_2)z} k^{r_3})$. It is easy to see that $\boldsymbol{B} = (\boldsymbol{NW})^s$.

– Proofs π_2, π_3 are simulated by $\mathcal{B}$.

Assuming that no signing requests with the same predicate input $\boldsymbol{C}$ are issued by $\mathcal{A}$, all the interactions are indistinguishable from interactions with a real OSPACBS signer. Using Lemma 3.3 it follows that every valid message-signature outputted by $\mathcal{A}$ is also a valid signature for the OSCBS scheme. Furthermore, if $\mathcal{A}$ queries for l valid signatures then $\mathcal{B}$ completes exactly l interactions with $\mathcal{S}$. So:

$$
\Pr\left[\text{PACBS-OneMoreForge}_{\mathcal{A},\text{OSPACBS}} = 1\right] = \Pr\left[\text{CBS-OneMoreForge}_{\mathcal{B},\text{OSCBS}} = 1\right]
$$

which concludes the proof. ∎

Note that the security guarantees for this instantiation hold against adversaries who cannot ask for a signature with the same challenge more than once. A larger protocol

that utilizes this scheme should make sure this restriction holds and deny issuing signatures on challenges that are already used.

3.7.4 Conditional Verifiability

A malicious user, without any access to either the verification key or the encryption key, not knowing the decryptions of C_1, C_2 cannot decide the value of the predicate to determine whether a received signature is valid or not. This can be proved by a reduction to the indistinguishability of the underlying encryption scheme.

Theorem 3.9: OSPACBS Conditional Verifiability

The OSPACBS scheme has conditional verifiability.

Proof. Suppose there exists a PPT algorithm $\mathcal{A}$ that breaks the conditional verifiability of the OSPACBS scheme, by winning the game in Algorithm 3.8. Then, there exists a PPT algorithm $\mathcal{B}$ that breaks the indistinguishability of the underlying encryption scheme.

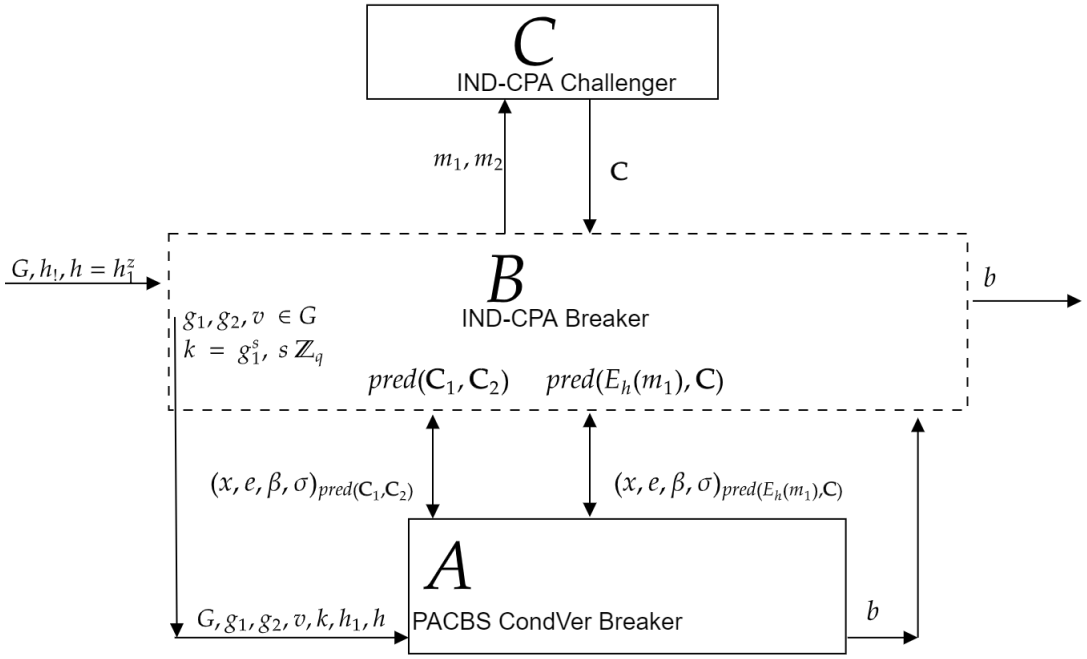


FIGURE 3.9: Breaking the IND-CPA by utilizing a break in Conditional Verifiability

The interactions of $\mathcal{B}$ are:

- $\mathcal{B}$ gets as input the parameters and public key of the underlying encryption scheme $\mathbb{G}, h_1, h$ where $h = h_1^z$ for some z.
- $\mathcal{B}$ creates keys and parameters for the PACBS scheme. In particular $\mathcal{B}$ chooses random $g_1, g_2, v \in \mathbb{G}$ and $s \in \mathbb{Z}_q$ and sets $k = g_1^s$. He hands the parameters $\mathbb{G}, g_1, g_2, v, k, h_1, h$ to $\mathcal{A}$.
- $\mathcal{B}$ computes the signatures requested by $\mathcal{A}$ by using the OSPACBS.Sign protocol from Figure 3.4. Note that for signing only the private key s is required. So $\mathcal{B}$ can create signatures without knowing the secret encryption key z.
- When $\mathcal{A}$ asks to be challenged on a signature, $\mathcal{B}$ selects two group elements m_1, m_2 and hands $(\mathsf{Enc}_h(m_1), \mathsf{Enc}_h(m_2))$ to the challenger of the IND-CPA property of the encryption scheme as his own challenge. He receives $\boldsymbol{C}$ as a response. $\mathcal{B}$ hands the pair $(\boldsymbol{C}, \mathsf{Enc}_h(m_1))$ as the public input for the challenge predicate $\mathsf{pred}(\boldsymbol{C}, \mathsf{Enc}_h(m_1))$ and receives e as a response. He computes a signature $(((n, \boldsymbol{B}, \boldsymbol{N}, \boldsymbol{W}), y_2), \pi_1, \pi_2, \pi_3)$ with public input $(\boldsymbol{C}, \mathsf{Enc}_h(m_1))$ and hands it to $\mathcal{A}$.
- $\mathcal{A}$ responds with 0 (valid) or 1 (invalid). In the first case $\mathcal{B}$ outputs 1 and in the second 0.

First, the signatures $\mathcal{A}$ receives are identically distributed as real ones since they are computed in the exact same way. In the case $\boldsymbol{C} = \mathsf{Enc}_h(m_1)$ the view of $\mathcal{A}$ is identical to a real interaction for a valid signature request and in the case $\boldsymbol{C} = \mathsf{Enc}_h(m_2)$ it is identical to an invalid request. It is clear that the advantage of $\mathcal{A}$ in distinguishing between the two cases is identical to the advantage of $\mathcal{B}$ against the indistinguishability of the underlying encryption scheme. ■

3.7.5 Public Auditability for signing and verifying

Using the correctness of the protocol, it can be seen that if the statements of the proofs presented hold, a signature is valid if and only if pred = 1. This means that for a Signer/Verifier to win one of the two Public Auditability experiments one of the proofs presented must not hold and he must convince the auditor that it does. Thus, at least for one proof soundness must not hold, which is a contradiction.

3.7.6 Performance

We now calculate the performance of our scheme. During the signing phase, the user performs 3 exponentiations for signing and 3 exponentiations for unblinding.

The signer performs 10 exponentiations to compute the functionality and 8 exponentiations for the proofs. For the auditing of π_{Sign} 18 exponentiations or $12t+6$ are required.

To verify the signature, if the verifier consists of a single entity 9 proofs are required for the functionality and 8 to generate the proofs. If the verifier consists of t members, $5t+4$ exponentiations are required to implement the functionality and $4t+1$ for the proofs. To audit the proof 16 exponentiations are required for 1 member and $8(t+1)$ for t members.

The results are summarized in Table 3.1. PACBS is quite performance-intensive, but this justified from the increased security guarantees.

Functionality	Entity	Exponentiations
PACBS.Sign	$\mathcal{U}$	6
PACBS.Sign 1 member	S	18
PACBS.Sign t members	S	$12t+6$
PACBS.AuditSign	$\mathcal{U}$	18
PACBS.AuditSign t members	$\mathcal{U}$	$12t+6$
PACBS.Verify 1 member	V	17
PACBS.Verify t members	V	$9t+8$
PACBS.AuditVrfy 1 member	$\mathcal{U}$	16
PACBS.AuditVrfy t members	$\mathcal{U}$	$8(t+1)$

Table 3.1: Performance of PACBS in exponentiations

3.8 Alternative OSPACBS instantiation

An alternative PACBS instantiation is provided, OSPACBS_2, where the signer and the verifier *do not* share a key. The key generation algorithm is the same as in Algorithm 3.4 along with h, h_1 from Algorithm 3.11. The function embed is defined as in Equation 3.2. The predicate pred is defined as in Equation 3.3.

The signing protocol is presented in Figure 3.10. The proofs π_1, π_2, π_4 are standard Okamoto [Oka92] proofs. The proof π_3 is similar to the one in Figure 3.5.

The verification algorithm is presented in Algorithm 3.15. The proofs π_1, π_2, π_3 are similar to the ones from Algorithm 3.13.

The algorithms for OSPACBS.AuditSign, OSPACBS.AuditVrfy are similar to Algorithm 3.12 and Algorithm 3.14 respectively.

Theorem 3.10: OSPACBS Correctness

The protocol OSPACBS_2 is correct.

Common input: $\mathsf{prms}, (\boldsymbol{C}_1, \boldsymbol{C}_2), \mathsf{pk}$
$\mathcal{U}$'s private input: $m \in \mathbb{M}$
S's private input : $\mathsf{sk}_\mathsf{S} = (s_1, s_2) \in \mathbb{Z}_q \times \mathbb{Z}_q$ such that $v = g_1^{-s_1} g_2^{-s_2}$
Commitment Phase. The Signer:

- Picks $r_1, r_2 \leftarrow\$ \mathbb{Z}_q$;
- Computes $x := g_1^{r_1} g_2^{r_2}$;
- Sends x to the user.

Blinding Phase. The User:

- Selects blinding factors $u_1, u_2, d \leftarrow\$ \mathbb{Z}_q$;
- Computes $x^* := x g_1^{u_1} g_2^{u_2} v^d, e^* := \mathsf{H}(m, x^*), e := e^* - d$;
- Sends $e, \boldsymbol{C}_1, \boldsymbol{C}_2$ to the signer.

Signing Phase. The Signer:

- Computes $y_1 := r_1 + e s_1$, $y_2 := r_2 + e s_2$;
- Computes $\boldsymbol{\beta_1} := \mathsf{embed}(\boldsymbol{C}_2, \boldsymbol{C}_1) \cdot \mathsf{Enc}_h(k^{y_1})$ and $\beta_2 := y_2$;
- Computes:

$$
\begin{aligned}
\pi_1 &\leftarrow \mathsf{NIZK}\{(g_1, g_2, x), (r_1, r_2) : x = g_1^{r_1} g_2^{r_2}\} \\
\pi_2 &\leftarrow \mathsf{NIZK}\{(g_1, g_2, v), (s_1, s_2) : v = g_1^{-s_1} g_2^{-s_2}\} \\
\pi_3 &\leftarrow \mathsf{NIZK}\{(\boldsymbol{C}_1, \boldsymbol{C}_2, k, \boldsymbol{\beta_1}), (y_1, \alpha, \gamma) : \boldsymbol{\beta_1} = \mathsf{embed}(\boldsymbol{C}_2, \boldsymbol{C}_1)\mathsf{Enc}_h(k^{y_1})\} \\
\pi_4 &\leftarrow \mathsf{NIZK}\{(g_1, g_2, x, v, e), (y_1, y_2) : g_1^{y_1} g_2^{y_2} = x v^{-e}\}
\end{aligned}
$$

- Outputs $\overline{\beta} := (x, e, \boldsymbol{\beta_1}, \beta_2, (\pi_1, \pi_2, \pi_3, \pi_4))$.

Unblinding Phase. The User:

- Verifies $\pi_1, \pi_2, \pi_3, \pi_4$;
- Unblinds by computing $\overline{\sigma}_\mathbf{1} := \boldsymbol{\beta_1} \cdot \mathsf{Enc}_h(k^{u_1})$ and $\sigma_2 := \beta_2 + u_2$;
- Outputs $\overline{\sigma} := (x^*, e^*, \overline{\sigma}_\mathbf{1}, \sigma_2)$.

FIGURE 3.10: The protocol $\mathsf{OSPACBS.Sign}_2\langle \mathcal{S}(s_1, s_2), \mathcal{U}(m), \mathsf{prms}, (\boldsymbol{C}_1, \boldsymbol{C}_2), \mathsf{pk} \rangle$

Algorithm 3.15: $\mathsf{OSPACBS.Verify}_2(\mathsf{prms}, \mathsf{pk}, \mathsf{sk}_\mathsf{V}, m, \overline{\sigma})$

Input : $\mathsf{prms}, \mathsf{pk} = (v, k), \mathsf{sk}_\mathsf{V} = (s, z), m, \overline{\sigma} = (x^*, e^*, \overline{\sigma_\mathbf{1}}, \sigma_2)$
Output: $\{0, 1\}$

$\gamma \leftarrow_\$ \mathbb{Z}_q$
$\boldsymbol{M} := \mathsf{Enc}_h(x^* \cdot (g_2^{\sigma_2} v^{e^*})^{-1})$
$\boldsymbol{V} := \boldsymbol{M}^s$
$\boldsymbol{R} := \left(\frac{\overline{\sigma}_\mathbf{1}}{\boldsymbol{V}}\right)^\gamma$
$\text{result} := \mathsf{Dec}_z(\boldsymbol{R})$
$\pi_1 \leftarrow \mathsf{NIZK}\{(\boldsymbol{V}, \boldsymbol{M}), (s) : \boldsymbol{V} = \boldsymbol{M}^s\}$
$\pi_2 \leftarrow \mathsf{NIZK}\{(\boldsymbol{V}, \overline{\sigma}_\mathbf{1}, \boldsymbol{R}), (\gamma) : \boldsymbol{R} = \left(\frac{\overline{\sigma}_\mathbf{1}}{\boldsymbol{V}}\right)^\gamma\}$
$\pi_3 \leftarrow \mathsf{NIZK}\{(h_1, h, \text{result}, \boldsymbol{R}), (z) : \text{result} = \mathsf{Dec}_z(\boldsymbol{R})\}$
$d := (\text{result} = 1)$
$\pi_{\mathsf{Verify}} := (\boldsymbol{M}, \boldsymbol{V}, \boldsymbol{R}, \text{result}, \pi_1, \pi_2, \pi_3)$
return $(d, \pi_{\mathsf{Verify}})$

Proof. Starting from $\mathsf{Dec}_z(\boldsymbol{V})$ straightforward calculations yield:

$$x^{*s} \cdot (g_2^{\sigma_2} v^{e^*})^{-s} = (x g_1^{u_1} g_2^{u_2} v^d)^s \cdot (g_2^{y_2} \cdot g_2^{u_2})^{-s} \cdot v^{-s(e+d)} =$$
$$x^s \cdot k^{u_1} \cdot g_2^{-sy_2} \cdot v^{-se} = g_1^{sr_1} g_2^{sr_2} \cdot k^{u_1} \cdot g_2^{-s(r_2+es_2)} \cdot (g_1^{-s_1} g_2^{-s_2})^{-se} =$$
$$g_1^{sr_1} \cdot k^{u_1} \cdot (g_1^{-s_1})^{-se} = g_1^{s(r_1+es_1)} \cdot k^{u_1} = k^{y_1+u_1}$$

Then: $\mathsf{Dec}_z(\boldsymbol{R}) = (k^{-(y_1+u_1)} \cdot (\mathsf{embed}(\boldsymbol{C}_2, \boldsymbol{C}_1) \cdot k^{y_1} \cdot k^{u_1}))^\gamma = (\mathsf{embed}(\boldsymbol{C}_2, \boldsymbol{C}_1))^\gamma$.

From there it is evident that $\mathsf{Dec}_z(\boldsymbol{R}) = 1 \Leftrightarrow \mathsf{Dec}_z(\boldsymbol{C}_1) = \mathsf{Dec}_z(\boldsymbol{C}_2)$. ∎

The security analysis of this alternative instantiation is similar to section 3.7.

3.9 A note on the ROS attack

In [Sch01], Schnorr proposed a new attack against interactive blind signature schemes ([Sch91; Oka92]) and a new problem to characterize their security. Recently, [FPS20; Ben+20], this attack became practical. Since our scheme indirectly builds on Okamoto - Schnorr blind signatures, we analyze the effects of this attack on our scheme.

The variation of this attack on the Okamoto - Schnorr blind signatures (cf. Figure 2.8) with $\mathsf{sk} = (s_1, s_2)$ and $\mathsf{pk} = v = g_1^{-s_1} g_2^{-s_2}$ is as following:

- $\mathcal{A}$ begins l *parallel* signing sessions with S.
- S selects $\{r_{i1}, r_{i2}\}_{i=1}^l \in \mathbb{Z}_q$ and computes $\{x_i := g_1^{r_{i1}} g_2^{r_{i2}}\}_{i=1}^l$.
- $\mathcal{A}$ selects $t \gg l$ and messages $\{m_j\}_{j=1}^t$.

- $\mathcal{A}$ selects $\{a_{ji} \in \mathbb{Z}_q\}_{j=1,i=1}^{t,l}$ and computes $\{x_j^* := \prod_{i=1}^{l} x_i^{a_{ji}}\}_{j=1}^{t}$ and $\{e_j^* := \mathsf{H}(m_j, x_j^*)\}_{j=1}^{t}$.
- $\mathcal{A}$ selects $l+1$ out of t equations from the system $\sum_{i=1}^{l} a_{ji} e_i = e_j^*$ and produces l solutions $\{e_i\}_{i=1}^{l}$.
- $\mathcal{A}$ sends the solutions $\{e_i\}_{i=1}^{l}$ as challenges to the S.
- S sends the responses $y_{i1} := r_{i1} + e_i s_1$, $y_{i2} := r_{i2} + e_i s_2$.
- From the solutions $\mathcal{A}$ computes a new signature:

$$\overline{\sigma}_j = (m_j, e_j^* = \sum_{i=1}^{l} a_{ji} e_i, \overline{\sigma}_{k1} = \sum_{i=1}^{l} a_{ji} y_{i1}, \overline{\sigma}_{k2} = \sum_{i=1}^{l} a_{ji} y_{i2})\}$$

- The forgery is valid since:

$$\begin{aligned} g_1^{\overline{\sigma}_{k1}} g_2^{\overline{\sigma}_{k2}} v^{e_j^*} &= g_1^{\sum_{i=1}^{l} a_{ji} y_{i1}} g_2^{\sum_{i=1}^{l} a_{ji} y_{i2}} v^{\sum_{i=1}^{l} a_{ji} e_i} \\ &= g_1^{\sum_{i=1}^{l} a_{ji}(r_{i1}+e_i s_1)} g_2^{\sum_{i=1}^{l} a_{ji}(r_{i2}+e_i s_2)} g_1^{-s_1 \sum_{i=1}^{l} a_{ji} e_i} g_1^{-s_2 \sum_{i=1}^{l} a_{ji} e_i} \\ &= g_1^{\sum_{i=1}^{l} a_{ji} r_{i1}} g_2^{\sum_{i=1}^{l} a_{ji} r_{i2}} = \prod_{i=1}^{l} x_i^{a_{ji}} = x_j^* \end{aligned}$$

The difficulty of the attack is abstracted in the intractability of the following *Random inhomogenities Overdetermined Solvable system of linear equations modulo q - ROS* problem proposed by Schnorr:

Definition 3.10: ROS$_l$ problem from [Sch01]

Given an oracle random function $\mathsf{H} : \mathbb{Z}_q^l \to \mathbb{Z}_q$ find coefficients $a_{kl} \in \mathbb{Z}_q$ and a solvable system of $l+1$ distinct equations with unknowns $e_1, ..., e_l \in \mathbb{Z}_q$:

$$\sum_{i=1}^{l} a_{ji} e_i = \mathsf{H}(a_{j1}, \cdots, a_{jl}), \qquad j \in [t], t \gg l$$

The ROS problem does not reduce the unforgeability of a blind signature scheme to one of the assumptions of subsection 2.1.1, but only on the length of the (prime) group order.

In [FPS20] it is proved that the blind Schnorr scheme is unforgeable if the One-More Discrete Log problem [Bel+03] is hard assumption holds and the ROS problem is hard. However, the second assumption, was found to be weak by [Wag02] and a practical attack was given while this thesis was being written [Ben+20]. This attack is polynomial in time when $l > log_2 q$ and subexponential when l is $\mathcal{O}(log_2 q)$.

This attack applies to all the schemes we presented in chapter 3. First, in OSCBS since $\beta_1 = k^{y_1}$, the forger must select:

$$\overline{\sigma}_{j1} = \prod_{i=1}^{l} k^{a_{ji} y_{ji}} \text{ and } \overline{\sigma}_{j2} = \sum_{i=1}^{l} a_{ji}\beta_{i2} \text{ and } e_j^* = \sum_{i=1}^{l} a_{ji} e_i \tag{3.7}$$

to produce a forgery for a valid signature ($b = 1$) (Figure 3.1) for a specified verifier with public verification key k. In the reduced round OSCBS, where $\beta_1 = (x \cdot g_2^{-\beta_2} \cdot v^{-e})^s$ the forger computes again $\prod_{i=1}^{l} \beta_{i1}^{a_{ji}}$ and $\overline{\sigma}_{j2}, e_j^*$ as in Equation 3.7. In the encrypted reduced round OSCBS, where the signer computes $\boldsymbol{\beta_1} = (g^r, \beta_1 h^r)$ the forger must employ the homomorphic properties of the ElGamal cryptosystem and produce:

$$\overline{\sigma}_{j1} = (\prod_{i=1}^{l} g^{r_i a_{ji}}, \prod_{i=1}^{l} \beta_{i1}^{a_{ji}} h^{r_i a_{ji}}) = g^{\sum_{i=1}^{l} r_i a_{ji}}, \prod_{i=1}^{l} \beta_{i1}^{a_{ji}} \cdot h^{\sum_{i=1}^{l} r_i a_{ji}} \tag{3.8}$$

In both previous cases, the forger must also select random group elements $\{x_i\}_{i=1}^{l}$ and make the signer apply them to the protocol.

Finally, in OSPACBS, the forger must issue a signing request with predicate input $(\boldsymbol{C}_1, \mathsf{ReEnc}(\boldsymbol{C}_1))$ so as to force the signer to compute a valid signature. As a result, $\boldsymbol{W}$ will now contain an encryption of 1. Then $\overline{\sigma}_{j1}$ is computed exactly as Equation 3.8.

While this attack is applicable to our primitives it *does not* contradict our unforgeability analysis of Theorem 3.3 and Theorem 3.8, where we proved that our proposals are unforgeable for $\mathcal{O}(polylog(\lambda))$ parallel sessions, while the attack of [Ben+20] uses $\Omega(\lambda)$ such sessions to solve the ROS problem. [5]

Additionally, since our primitives are part of larger protocols we can employ mechanisms at a higher level to compensate for the lack of efficiency that the bound on l entails. For instance, in the case of forgery, the number of blind signature requests, will be less than the number of signatures, which can be detected. This check can be accompanied by further ones to detect duplicates. Finally, in order not to allow the adversary to take advantage of existing requests, a proof of knowledge of the plaintext encrypted by $\boldsymbol{C}_1, \boldsymbol{C}_2$ must be also provided.

Finally, note that this attack does not apply to *Clause* blind Schnorr signatures proposed in [FPS20]. For each requested signature by the user, the signer creates two parallel sessions with commitments x_0, x_1. The user computes two challenges e_0, e_1. But at the last step, the signer flips a coin $b \leftarrow_\$ \{0,1\}$ and if $b = 0$ aborts the signing session, while if $b = 1$ generates the signature and concludes the session. Neither the attack of [Wag02] nor the attack of [Ben+20] are practical now since $\mathcal{A}$ must guess

[5] To be exact, for our schemes to be secure we require $l < log_Q q < log_2 q$ parallel sessions where Q is the number of queries to the random oracle (according to the analysis of [PS00]).

which session will lead to a valid signature. This is impractical since for l sessions there are 2^l possible selections. The coin used, resembles the conditionality bit of our signatures. As such, the latter could be used to create a more realistic version of Clause blind Schnorr signatures, where the signer produces a single commitment and the user computes a single challenge. Instead of aborting, though, the signer could generate both a valid and an invalid signature and return them both, in random order, to the forger. The conditional verifiability property, would hide which one is valid and which is not, so the forger would have to select one at random in order to produce the forgery. However, this change would entail modifications to the rest of the protocols as the user would have to submit both signatures to the signer. We leave this for future work.

4 Electronic Voting Systems and Models

> All models are wrong, but some are useful
>
> George Box

In section 1.2 we informally described the most important security requirements of electronic voting systems. In this chapter, we present a more rigorous analysis of the sought properties, by using game-based formal models. To do so, we investigate the relevant literature. We accompany these definitions of security by example voting systems that are of interest in our thesis such as Helios [Adi08], the FOO [FOO92] and [JCJ05] along with their many variations. In the end, we propose our novel game-based definitions of everlasting privacy and explore the relations between the security requirements as made evident by the formal models.

4.1 Voting System Syntax

We begin by describing the components of an abstract voting scheme VS that incorporates functionalities from many proposals in the literature. The aim is for it to be as generic as possible. The formalization is built, by having in mind, election schemes that are initialized once and reused in many elections. As a result, many functionalities and parameters involved are not present in the analyzed voting schemes. In our syntax we denote optionality with ?.

VS is associated with three parameters, the security parameter λ, the number of voters n and the number of possible candidates m. The voters are collectively denoted by V and express their preferences by possibly using *voter supporting devices (VSD)*, i.e. software-hardware combinations that allow them to interact with the election system. The scheme is controlled by an Election Authority EA, which is stateful and its state is updated in every step of the protocols. In all algorithms we omit the state update for simplicity. The EA consists of 3 sub-entities the registration authorities RA, the tallying authority TA and the bulletin board BB. The latter two are always

part of VS, while the former appears only in schemes that explicitly deal with internal registration options. The alternative is that a scheme uses an external registration service. The BB denotes the public transcript of all executed protocols. It contains all the election-related data (ballots, parameters, proofs etc.) As we described in subsection 2.1.2 it is append-only. Therefore, whenever it is used, it contains all the data already written to it. Thus, the BB would suffice as the public input in the definitions of the functionalities of VS. However, when we wish to emphasize the use of such parameters, we also include specific public data. When we would like to refer to the bulletin board as functionality and not as a data store we use a method invocation-like syntax and we write $\mathsf{BB}()$.

Definition 4.1: Voting scheme

A voting scheme

$$\begin{aligned}\mathsf{VS} =(&\mathsf{Setup}, \mathsf{Register}, \mathsf{SetupElection}, \mathsf{Authorize},\\ &\mathsf{Vote}, \ \mathsf{Valid}, \mathsf{VerifyBallot}, \mathsf{Tally}, \mathsf{Verify})\end{aligned}$$

is a tuple of algorithms and protocols executed by the election authority $\mathsf{EA} = (\mathsf{RA}, \mathsf{TA})$, the bulletin board BB and the set of voters $V = \{\mathsf{V}_1, \ldots, \mathsf{V}_n\}$ parameterized by $\lambda, n, m \in \mathbb{N}$ such that:

- $(\mathsf{prms}, \mathsf{sk}_{\mathsf{EA}}, \mathsf{pk}_{\mathsf{EA}}, \pi_{\mathsf{Setup}}) := \mathsf{VS.Setup}(1^\lambda)$
- $(\mathsf{pk}_{i?}, (\mathsf{sk}_i, \mathsf{pk}_i)_?) := \mathsf{VS.Register}_?\langle \mathsf{RA}(\mathsf{sk}_{\mathsf{RA}?}), \mathsf{V}_i() \rangle$
- $(\mathsf{V}_{\mathtt{El}}, \mathsf{CS}) := \mathsf{VS.SetupElection}_?(sk_{\mathsf{EA}}, n, m, \mathsf{prms}, \mathsf{L})$
- $(\perp, (\mathsf{b}_i, \pi_{\mathsf{b}_i}, \mathsf{r}_{i?})) := \mathsf{VS.Vote}\langle \mathsf{EA}_?(\mathsf{sk}_{\mathsf{EA}?}), \mathsf{V}_i(\mathtt{vt}_i, \mathsf{sk}_{i?}), \mathsf{prms}, \mathsf{pk}_{\mathsf{TA}}, \mathsf{pk}_{i?}, \mathsf{V}_{\mathtt{El}}, \mathsf{CS} \rangle$
- $\mathsf{BB} \Leftarrow \mathsf{VS.Cast}_?\langle \mathsf{BB}(), \mathsf{V}_i(\mathsf{b}_i, \pi_{\mathsf{b}_i}) \rangle$
- $\{0,1\} = \mathsf{VS.Valid}(\mathsf{BB}, \mathsf{b})$
- $\{0,1\} = \mathsf{VS.VerifyBallot}(\mathsf{r}_i, \mathsf{b}_i, \mathsf{BB}, \mathsf{prms}, \mathsf{pk}_{\mathsf{EA}}, \mathsf{L})$
- $(\mathtt{T}, \pi_{\mathtt{T}}) := \mathsf{VS.Tally}(\mathsf{sk}_{\mathsf{TA}?}, \mathsf{CS}, \mathsf{BB})$
- $\{0,1\} = \mathsf{VS.Verify}(\mathtt{T}, \pi_{\mathtt{T}}, \mathsf{prms}, \mathsf{pk}_{\mathsf{EA}}, \mathsf{BB}, \mathsf{CS}, \mathsf{V}_{\mathtt{El}})$

We now detail the various functionalities found in Definition 4.1:

- $(\mathsf{prms}, \mathsf{sk}_{\mathsf{EA}}, \mathsf{pk}_{\mathsf{EA}}, \pi_{\mathsf{Setup}}) := \mathsf{VS.Setup}(1^\lambda)$

  Setup is an algorithm executed by the EA which on input 1^λ outputs public parameters of the VS and a key pair of the EA $(\mathsf{sk}_{\mathsf{EA}}, \mathsf{pk}_{\mathsf{EA}})$. The bulletin board transcript BB is appended with $(\mathsf{prms}, \mathsf{pk}_{\mathsf{EA}})$. Note, that if the scheme consists of both RA, TA then distinct key pairs $(\mathsf{sk}_{\mathsf{TA}}, \mathsf{pk}_{\mathsf{TA}}), (\mathsf{sk}_{\mathsf{RA}}, \mathsf{pk}_{\mathsf{RA}})$ are generated.

- $(\mathsf{pk}_{i?}, (\mathsf{sk}_i, \mathsf{pk}_i)_?) := \mathsf{VS.Register}_?\langle \mathsf{RA}(\mathsf{sk}_{\mathsf{RA}?}), \mathsf{V}_i() \rangle$

Register is a protocol executed between a voter V_i and the RA. We assume that the voter id, is public $i \in [n]$. The output is a voter public key pk_i (available to both parties) and a secret key sk_i as a private output of the voter, which takes the role of a voter credential. The values (i, pk_i) are appended to the BB. We must stress here, that Register is optional, as the voters could be identified by an external service or by non-electronic means (in-person). Furthermore, even if it is used, it is not obligatory for voters to have a key pair. The RA might use its secret key as a signing key and sign each voter credential, or it might use it to ensure the authenticity of the election roll when it posts the public list in the BB. Registration is an important part of remote voting schemes, however, and as such it is included in our model. In such protocols the registration phase is meant to be executed once and used for multiple elections.

– $(\mathsf{V}_{\mathtt{El}}, \mathsf{CS}) := \mathsf{VS.SetupElection}_?(\mathsf{sk}_{\mathsf{EA}?}, n, m, \mathsf{prms}, \mathsf{L})$

The EA creates a new election using as input its secret key $\mathsf{sk}_{\mathsf{EA}}$, the number of voters n, the number of candidates m and additional election information (e.g. start and end times). The SetupElection functionality outputs the set of identities of eligible voters for the particular election $\mathsf{V}_{\mathtt{El}} \subseteq [n]$, along with their public keys and the candidate slate CS which contains encodings of the choices. The tuple of lists $(\mathsf{V}_{\mathtt{El}}, \mathsf{CS})$ is posted to the BB. Note that if there is no need for the EA to sign the output of this functionality, its secret key is not required.

– $(\perp, (\mathsf{b}_i, \pi_{\mathsf{b}_i}, \mathsf{r}_{i?})) := \mathsf{VS.Vote}\langle \mathsf{EA}_?(\mathsf{sk}_{\mathsf{EA}?}), \mathsf{V}_i(\mathtt{vt}_i, i, \mathsf{sk}_{i?}), \mathsf{prms}, \mathsf{pk}_{\mathsf{TA}}, \mathsf{pk}_{i?}, \mathsf{V}_{\mathtt{El}}, \mathsf{CS}\rangle$

Vote is a protocol executed between the EA and a voter V_i which aims to create and authorize the ballot. The voter's private input is her choice of candidate $c_i \in \mathsf{CS}$ and the public voter identity i which could be the legal name of a voter or an email address. Optionally, in systems like [JCJ05] that require private credentials for voter authentication, her secret key sk_i. The EA can play an active role in the protocol, by authorizing ballots created by the voter. This is typically done by signing ballots as in [FOO92]. The EA checks voter identification information and creates a signature. Everybody can verify it for the scheme to provide eligibility verifiability. In that case, the EA requires a private input like a secret key $\mathsf{sk}_{\mathsf{EA}}$. The public input consists of the system parameters, the corresponding public keys $\mathsf{pk}_{\mathsf{TA}}, \mathsf{pk}_i$, the set of eligible voters $\mathsf{V}_{\mathtt{El}}$ and the candidate slate CS. The protocol outputs the ballot b_i, which is a transformation (encryption or commitment) of $\mathtt{vt}_i$ and a proof π_{b_i} of the correctness of this transformation, usually a NIZK (cf. section 2.4.1). Optionally, it outputs a receipt, so that the voter can check if her vote will be later counted or not. In voting systems, where the ballot is created by encryption of the voter choice, the receipt could

be the randomness used to encrypt the vote. The election authority receives no output from this functionality. We again assume that the protocol transcript is appended to the BB. Coercion resistant voting schemes, also use an extra functionality that allows the voter to evade coercion. In some systems, this functionality is a simple repetition of VS.Vote, while in others it is combined with fakekey - a credential generation mechanism. Finally in vote-and-go schemes the VS.Vote protocol can be replaced by an algorithm $\mathsf{VS.Vote}(\mathtt{vt}_i, i, \mathsf{sk}_{i?})$.

– $\mathsf{BB} \Leftarrow \mathsf{VS.Cast}_?\langle \mathsf{BB}(), \mathsf{V}_i(\mathsf{b}_i, \pi_{\mathsf{b}_i})\rangle$

Cast is a protocol executed between the voter V_i and the bulletin board BB. The voter V_i essentially appends a transformation of the authorized ballot b_i to the election transcript. The transformation must not render the authorization information obsolete. One such possible transformation is the unblinding of a signature as in [FOO92], but other might be applicable too. In most voting schemes the Vote and Cast functionalities are merged into a single functionality.

– $\{0,1\} = \mathsf{VS.Valid}(\mathsf{BB}, \mathsf{b})$

Valid is an algorithm executed by the BB when the ballot b is to be appended. It performs various checks in order to make sure that the ballot conforms to the specifications set by the EA. For instance, it verifies the proofs of correct ballot formation. Additionally, to avoid some attacks, it might check that there are no exact copies of the encrypted contents of the ballot inside the BB. This functionality is sometimes executed by the EA, but it can also be embedded into the BB.

– $\{0,1\} = \mathsf{VS.VerifyBallot}(\mathsf{r}_i, \mathsf{b}_i, \mathsf{BB}, \mathsf{prms}, \mathsf{pk}_{\mathsf{EA}}, \mathsf{L})$

VerifyBallot is an algorithm executed by the voter with input the receipt r_i received during voting, her ballot b_i, the contents of the BB. It is meant to support individual verifiability, where a voter verifies that her ballot will be counted.

– $(\mathtt{T}, \pi_{\mathtt{T}}) := \mathsf{VS.Tally}(\mathsf{sk}_{\mathsf{TA}?}, \mathsf{BB})$

Tally is an algorithm executed by the election authority with input the parameters of the scheme prms and the transcript BB of the bulletin board which contains the ballot and outputs the election tally T and a proof $\pi_{\mathtt{T}}$. The output is appended to the bulletin board BB. In case, the ballots are decrypted the TA provides proof of correct decryption. If the ballots are not decrypted, then everyone can perform this function.

– $\{0,1\} = \mathsf{VS.Verify}(\mathtt{T}, \pi_{\mathtt{T}}, \mathsf{prms}, \mathsf{pk}_{\mathsf{EA}}, \mathsf{BB}, \mathsf{CS}, \mathsf{V}_{\mathtt{El}})$

Verify is an algorithm executed by any interested party (voters or public interest organizations) with input the election tally $\mathtt{T}$, the proof of correct computation $\pi_{\mathtt{b}_i}$ the parameters of the scheme prms, the public key of the $\mathsf{pk}_{\mathsf{TA}}$, the contents of the bulletin board BB, the candidate slate CS and the set of eligible voters for the election $\mathsf{V}_{\mathtt{E1}}$. The output is a bit representing the result of the election verification. Verify can be executed by any interested party using all the ballots, for universal verifiability purposes, since all inputs can be found in the BB.

Every voting protocol is associated with a function result that computes the tally based on the plaintext of the ballots, i.e. $\mathsf{result} : \mathsf{CS} \to \mathtt{R}$ where $\mathtt{R}$ is the set of all possible results. The purpose of this function is to present the 'correct' tally of the election in order to compare it with the output of the Tally algorithm. In some schemes, the voter identity might play some part in the result function. For instance, if it represents a credential, only votes with correct credentials will be counted. Consequently, a more general representation of the result function would be $\mathsf{result} : V \times \mathsf{CS} \to \mathtt{R}$.

A voting scheme is intrinsically correct if $\mathsf{result}(\{\mathtt{vt}_i\}_{i=1}^{n}) = \mathsf{Tally}(\cdot, \mathsf{BB})$ where $\mathsf{BB} = \{\mathtt{b}_i\}_{i=1}^{n}$.

4.2 Helios Case Study

Helios [Adi08] can be considered a reference voting system. Much of its workflow is used in other election systems. It also serves as a model for the property of verifiability, as it provides this property without the need to trust the members of the TA. Its initial version is very closely based on a well-known voting protocol from [CGS97]. The main addition concerns mechanisms to capture voter intent. It works in the unsupervised setting allowing remote voting through the internet. However, as it is designed for low coercion environments the only way to defeat a coercer is by re-voting. Initially it supported both tallying using mixnets and homomorphic encryption. However, currently only the homomorphic tallying is maintained. A fork of Helios, Zeus [Tso+13] enables tallying using mixnets. It has been used in many binding elections as described in [MPQ09] and it has been widely analyzed both for verifiability and privacy leading to many variations [BPW12; Cor+14; CGG19].

Main workflow

The entities comprising the system consist of voters, election administrators EA and tellers TA. The BB is a centralized database controlled by the EA. Both the EA and the BB do not participate in the cryptographic protocol but perform helper functions.

Helios does not support all functionalities from section 4.1. In fact, in the basic version:

$$\mathsf{VS}_{\mathsf{HELIOS}} = (\mathsf{Setup}, \mathsf{Vote}, \mathsf{VerifyBallot}, \mathsf{Tally}, \mathsf{Verify})$$

Newer variations have added a registration authority RA and the relevant functionalities.

Setup. The EA selects the members of the TA and generates the cryptographic parameters $\mathsf{prms} = (\mathbb{G}, g, q)$ of the election as well as the keys of the TA $(\mathsf{pk}_{\mathsf{TA}}, \mathsf{sk}_{\mathsf{TA}})$. The $\mathsf{sk}_{\mathsf{TA}}$ is shared among the tellers, by using the scheme of section 2.2. Each teller must also post a proof π_{Setup} of correct share construction, i.e. a proof $\mathsf{NIZK}\left\{(\mathbb{G}, g, \mathsf{pk}_i), \mathsf{sk}_i : \mathsf{pk}_i = g^{\mathsf{sk}_i}\right\}_{i \in \mathsf{TA}}$ which is a Schnorr proof π_S (cf. section 2.4.1).

The cryptographic parameters, the election public key and its shares are posted in the BB joined with π_{Setup}. The EA creates the candidate list CS. This setup is repeated for each election. As a result, there is no SetupElection functionality. A list of eligible voters $V_{\mathtt{El}}$ is selected by the authorities. Similarly, in the most used version, there is no registration authority. It is assumed that the eligible voters are authenticated using external services. The EA inputs all these parameters to a hash function to create the election fingerprint and post it to the BB.

Vote. Voting takes place through a web browser, which connects to the BB and downloads the election public data and recomputes the fingerprint for validation. All computations are performed locally on the client browser, which can be considered the voter VSD. When the voter wants to cast her vote, the Helios client software simulates a voting booth, by disabling all network connectivity. The voter creates her ballot by encrypting her candidate choice using exponential ElGamal (section 2.2). When there are multiple candidate choices each voter must input either 0 or 1 to indicate that she prefers the particular voter. If homomorphic tallying is to be used, then the voter must prove that her vote is valid, i.e. that each candidate received a preference that consisted either of 0 or of 1, and that each voter voted for only one candidate (that is the sum of the preferences equals to 1). This is performed through a disjunction of Σ-protocols from [CGS97] that a voter has produced a correct encryption of a message from a known set (cf. section 2.4.1).

Then the voter decides whether she will cast or audit the vote. In the latter case, the randomness used to encrypt the vote is revealed and the voter can re-compute the ballot using a tool of her choice to check if it matches with the encryption performed by the system. An audited vote cannot be cast. After all voters have cast their ballots, they are posted to the BB together with the voter identities, by the Helios server, in order for the voter to check if the ballot will be counted. Note that the

auditing protocol does not use the voter identity. As a result, the ballot consists of: $\mathsf{b} = (i, \mathtt{vt}, \pi_{\mathsf{Enc}}, \pi_{\mathsf{b}})$, where $\mathtt{vt}$ contains an encryption of $m \in \{0,1\}$ for each candidate, π_{Enc} contains the non interactive PoK for the correctness of each encryption and π_{b} is the proof of correctness for the entire ballot.

Alternatively, in [MPQ09] voter aliases are provided by a registration authority RA. One of the reasons for this variation, is that if a ballot is accompanied with the voter ID, then it leaks which voters abstained, which might be illegal in some jurisdictions. A further variation is to post the ballots to the BB without names or aliases. As a minor coercion countermeasure, the voters can re-vote. Only the last ballot is counted per voter.

VerifyBallot. The EA provides as a receipt for voters to check their votes, the randomness used to create the encryptions posted on the BB. For individual verifiability the voter can create encryptions of their choices using third-party tools and pinpoint the exact bits on the BB (under their names or by using a search algorithm).

Tally. In the original mixnet-based version, when the voting phase has concluded, the ballots are run through a verifiable shuffle and then jointly decrypted by the members of the TA, who then produces the result. In the homomorphic version, all the ballots are multiplied, and the result is decrypted, are partially decrypted. All decryptions are accompanied with a proof of correct decryption π_{Dec} from [CGS97] (cf. section 2.4.1). The EA combines the partial decryptions to create the decryption of the complete tally.

Verify. In order to verify the election every interested participant can check the proofs generated by the various authorities $\pi_{\mathsf{Setup}}, \{(\pi_{\mathsf{Enc}}, \pi_{\mathsf{b}})\}_{\mathsf{VR}}, \pi_{\mathsf{Dec}}$.

This main workflow described in this section is included in Helios 2.0. The currently deployed Helios 3.0 expands this functionality with practical additions that make registration of the system easier.

Attacks and Variations

There have been numerous attacks on Helios' verifiability, both concerning the implementation details and the general security model. The former can take advantage of programming errors and oversights as well as the difficulty of implementing secure functionality over an insecure medium. Furthermore, they can be caused by the selection of cryptographic parameters, such as groups and candidate encodings. They have been studied in detail in [ED10; CE16]. While they are by no means of negligible importance, here we focus on attacks on the model of the Helios voting scheme.

Clash attacks Clash attacks were discovered by [KTV12b]. The corrupted EA provides voters with identical ballots. When they try to verify their receipts the verification is successful, however they are verifying the same ballot and not their individual ones. As a result, the BB is free to modify the rest of the ballots, to candidates of its liking. In the original Helios variant, this attack cannot be mounted, because of the unique ciphertext produced by the ElGamal encryption. So, if two distinct voters discover next to their names an identical random string, it means that the EA has performed the clash attack.

In the Helios variant with aliases [MPQ09] this attack can be performed if the corrupted VSD colludes with the RA issuing the pseudonyms. The RA issues the same alias to some voters that will choose the same candidate with very high probability (e.g. members of the same party). The VSD always uses the same random coins for these voters (in all audits). As a result, when the voter verifies their vote, all the voters with the same alias will successfully pass the verification. The BB can now replace the identical ballots with the ones of its liking. According to [KTV12b], the clash attack will not be detected in the audits, as the audit procedure checks that the encryption of the candidate using the specified randomness always produces the audited ciphertext. This attack will not work, if the random coins used in successive audits are revealed to the voter.

To perform the clash attack on the variation of Helios where only the ballot contents are posted on the BB, exactly the same sequence of random coins must be used by the VSD. This means that all the voters that perform a single audit will be provided with randomness r_1, all the voters that perform two audits will be provided with randomness r_2 etc., resulting in a series of identical ballots. The malicious BB can intercept the identical ballots in constant time, using a hash table, and inject ballots of its own.

Both attacks of [KTV12b] can be deterred if the ballot is posted on the BB immediately after voting and not when the voting period has expired. Another solution is to let voter contribute their own randomness.

Weak Fiat-Shamir transform (Helios-BPW) A more serious attack was noted in [BPW12] and was briefly mentioned in section 2.4.1. The [CGS97] protocol on which homomorphic Helios is based, employs several Σ-protocols for the voter to prove that she cast a correct vote, for the members of the TA to prove that the public key shares have been correctly computed and for the TA to prove that the result is correctly decrypted for the homomorphic product of the votes. These Σ-protocols have been turned non-interactive, by using the weak version of the Fiat-Shamir heuristic [FS86], where the random oracle is applied to the commitment message only.

The voters and the TA can exploit this weakness and adaptively change the NIZK of correct vote encryption, by altering the public key to construct a proof that their vote encodes $\mathtt{vt} \in \{0,1\}$, when in fact they encode a vote $\mathtt{vt} \notin \{0,1\}$. This will mean that an arbitrary number of votes will be added to the tally, in an undetected manner. The proof will verify correctly, as the real public key is not an input to the random oracle call.

The same technique can be used by a malicious member of the TA to create a proof of correctness of its public key share, even though that it does not correspond to the correctly assigned private key. As a result, when the shares are combined from the decryption of the result, the malicious teller will contribute a random private key share, which will make the result decrypt to a random element of $\mathbb{Z}_q$. If left unchecked (e.g. against the maximum number of voters) this will result in a brute force search for a large discrete logarithm, causing a Denial of Service attack.

A simple fix for these attacks, is to use the strong Fiat-Shamir transform as proposed by [BPW12] and include the complete statement to be proved into the random oracle call. Similar attacks have been made against the privacy of the Helios system, and will be discussed in the next section. The variation of Helios with the strong Fiat-Shamir heuristic is referred to the bibliography as Helios-BPW.

Helios with credentials A takeaway from the clash attack is that a corrupted BB can alter votes. In the original variant of Helios, this was not applicable as the corrupted BB should indicate to which (real-world) voter identities these votes correspond. As a result, if a voter saw a vote under her name, when in fact she was absent during the election, she would probably complain. However, as mentioned in [Cor+14], posting the real-world identities of voters is not always legal and many voters that abstain have no interest in the election anyway (although many of their acquaintances might). In order to control the BB ability to stuff ballots [Cor+14] propose to use a registration authority RA, that issues credentials, i.e. voter pseudonyms that allow voter authentication and are disassociated from real-world voter identities. This variation of Helios is called Helios with credentials or Helios-C. The credentials for Helios-C consist of a public and private counterpart. The voters receive their private counterpart (through an offline phase) and use it to sign their ballots (encrypted votes) in the Vote functionality. The public keys are made available in a list and can be used to validate the signature ballots. This functionality can be executed independently from voters for individual verifiability, during tallying from the members of the TA and during the auditing for universal verifiability. The existence of an independent RA changes the trust assumptions and leads to two variations of verifiability

which will be explored in the following sections. A system that bears many similarities to Helios with credentials is Belenios [CGG19]. Another variation of Helios to add eligibility verifiability is proposed in [Sri+14].

Lack of ballot independence and ballot weeding Ballot independence is a property that does not allow a voter to replay another voter's interaction with the voting system either exactly or in a related manner. For instance, a violation of ballot independence would be for a voter to cast an exact replica of a ballot on the BB. Helios does not provide ballot independence and this fact has been employed in [CS13] to break ballot secrecy.

For instance, in an election with 3 voters, assume that V_1 and V_2 have already cast their ballots. The BB will contain the following tuples:

$$(ID_1, \mathsf{b}_1) = (\mathtt{vt}_1, \pi_{\mathsf{Enc},1}, \pi_{\mathsf{b},1})$$
$$(ID_2, \mathsf{b}_2) = (\mathtt{vt}_2, \pi_{\mathsf{Enc},2}, \pi_{\mathsf{b},2})$$

Assume that $\mathcal{A}$ controls V_3 and wants to learn how V_1 voted. He can replay an exact copy of b_1, which is a valid ballot. When the votes are decrypted then whoever candidate receives 2 votes, will be revealed to be the option preferred by V_1. While this attack, is artificial in nature as it is targeted only to elections with 3 candidates, [CS13], show how it can be used to break privacy in precinct-based elections where the adversary can learn with the help of a few corrupted voter and with great confidence how a particular voter voted in small precincts, that publish their partial tallies.

Other variations of this attack, exploit the malleability of the cryptosystem and the NIZK proofs in order not to post an exact copy of a ballot contained in the BB [CS13]:

- Adding multiples of the group order to the response s of the NIZK.
- Permuting the elements of the vectors
- Reencrypting (homomorphically changing) all elements of the ballots and the respective proofs

In order to thwart these attacks, [CS13] propose that the Helios BB must be changed in order to accept unique ciphertexts, make the NIZK proofs non-malleable and only allow ciphertexts that encrypt proper elements of $\mathbb{G}$. Additionally, the random oracle call in the Fiat-Shamir heuristic must include the voter identity in order to bind each NIZK to a unique voter and prevent ballot copying. These changes are collectively called ballot weeding.

As a result, the secure version of Helios that attends to all the attacks in the literature is Helios-C-BPW with ballot weeding.

KTV-Helios An extension of Helios-C, that uses many techniques inspired from coercion resistant schemes, that are of interest to our case was proposed in [KTV15; BKV17] and will be referred to as Helios-KTV. The target property that this variation aims to satisfy is participation privacy, that is to hide who voted in order to protect the identities of those who abstained. Systems with credentials violate it, as a simple search for a vote corresponding to a public credential, can reveal if the relevant voter participated in the election.

The basic method to achieve participation privacy is to add dummy votes for all voters. These dummy votes are null, i.e. they do not affect the result, as they are encryptions of 0. They can be added by the EA and other interested parties. This idea was first proposed in [JCJ05] and is also used in our scheme in chapter 5. The IND-CPA property of the underlying cryptosystem prevents anyone from distinguishing real from dummy ballots. In particular, Helios-C is extended with a new functionality $\mathsf{VoteDummy}(i)$ that invokes VS.Vote with voting option $\mathtt{vt} = 0$ for the voter. The resulting encrypted ballot is added to the BB for the voter and is accompanied by π_{Enc} of correct encryption. Before tallying begins, the TA multiplies all the entries for a particular voter to receive the final ballot and anonymizes. Subsequently, a PET is performed between a deterministic encryption of each vote and the final output, to check vote validity.

Another interesting aspect of KTV-Helios is that it provides a form of receipt-freeness, by using a form of deniable vote updating, detailed in section 4.4.1.

4.3 Election Verifiability

A short informal introduction to verifiability was given in section 1.2. Election verifiability is the property that allows the voters to regain the trust endangered by the volatile nature of computer systems that implement e-voting functionalities. This lack of trust is made worse when combined with the motivation for malice inherent in all types of elections due to the enormous gains of the winners. In this section, we will focus on formal definitions of verifiability, that can be used to model our voting scheme presented in chapter 5.

Recall that verifiability is not a monolithic concept. It comprises many sub notions that capture specific parts of voting systems and processes. Definitions given for these sub notions are often incomplete and tailored to specific systems. There are two encompassing notions that holistically embody them.

End-To-End verifiability, now a folklore term, was initially proposed in a series of works by Chaum, Adida and Neff [Cha04; Nef04; AN06; Ben06]. It is an umbrella

term for the properties we mentioned in section 1.2 cast as intended, recorded as cast and tallied as recorded. Its main emphasis is on accurately and securely conveying voter intent to the election system. The predominant solution to achieve it, aka the cast-or-audit mechanism (Benaloh challenge) [Ben06], especially in the manner used in the most successful remote voting scheme Helios [Adi08], can be made compatible with any remote voting scheme, as the one presented in this thesis. Consequently, we do not deal at all with cast-as-intended verifiability, despite its huge importance. In what follows, we will use the alternative notion of *Election verifiability* as it is proposed by [SFC15] formalized in the computational setting. It comprises 3 notions: *Individual*, *Universal* and *Eligibility* verifiability. The first two correspond to recorded as cast and tallied as recorded verifiability, respectively. Eligibility verifiability is often considered as contained within universal verifiability in some definitions, while other treat it as a separate concept. We discuss both possibilities.

Trust assumptions Since the essence of verifiability is to protect against systemic errors or malice, it makes sense to consider the election authorities that control the system as adversarial. As a result, in formal models, the EA either as a whole or in part (RA, TA, BB) is considered to be completely corrupt by $\mathcal{A}$. This means, that even if they consist of many members (with conflicting interests), we must assume that they collude in order to attack the system. Consequently, schemes that rely on at least one honest participant, are not verifiable [MPT20].

In systems, where the components of the EA handle different functionalities, there are nuances of verifiability that differ in their trust assumptions. For instance, for universal verifiability the TA must be completely corrupt as it handles vote tallying. For eligibility verifiability the same applies to the RA. An open question is whether the BB is considered corrupt or not. In many implementations, it is completely controlled by the EA and as a result, it this question makes no sense. Theoretically however is an independent component subsection 2.1.2, and if treated as such then some interesting attacks can come up, leading to variations that fix them.

Regarding voters, the $\mathcal{A}$ can either statically corrupt some of them at the beginning of the protocol, or dynamically during its execution. These are denoted by $V_{\texttt{Corr}}$. The rest of the voters are assumed to be honest and denoted by $V_{\texttt{Hon}}$.

A real-world problem with honest voters is that they do not always perform the verification procedure. This has various effects in the verifiability guarantees, but surprisingly even in privacy guarantees [CL18] as we will see in section 4.5. Some definitions of verifiability [Cor+14; KZZ15b; Cor+16] take this into account concluding that for a voting system to be verifiable, the following guarantees can be provided:

– All the votes of the voters who check are included in the tally.

– Some of the votes of the voters who do not check are included in the tally.

– There is no ballot stuffing or equivalently the number of adversary-cast votes do not exceed the number of corrupted voters.

4.3.1 Individual verifiability

Individual verifiability, also called *traceability* according to [JMP13], was first used in the context of mixnet-based anonymous channels, where it indicated the capacity of senders to verify that their message reached the intended recipients [SK95]. In the context of electronic voting, it refers to voters verifying that their vote was included in the tally. The architecture of most voting systems proposed in the literature makes this equivalent to votes being present in the BB, assuming that all such ballots will be counted. As a result, for a voting system to be individually verifiable, the voter must be able to locate her ballot in the BB.

This does not apply to voting systems, mostly aiming for coercion resistance, where the vote might indeed reside in the BB, but this does not automatically mean that it will be counted, since this depends on the validity of credentials. This is the case with our voting system, as well(cf.section 5.2). When we refer to the ballot we mean the version of it that will be counted, and not some intermediary version.

A necessary condition for individual verifiability is that the ballots are unique. This is formally defined using the game Algorithm 4.1, first proposed in [SFC15]. There are two variations for this game. The first applies to systems without any registration phase, that use some external mechanism to authenticate the voters, like the original version Helios. The adversary generates the public parameters of the election system and selects two different choices from the adversary generated CS to dictate to the voter. The adversary wins the game if it can manage to create a clash, i.e. two identical ballots.

Algorithm 4.1: $\mathsf{IndVer}^{\mathsf{Ext}}_{\mathcal{A},\mathsf{VS}}$ from [SFC15]

Input : security parameter λ
Output: $\{0,1\}$

$(\mathsf{prms}, \mathsf{pk}_{\mathsf{TA}}, \mathsf{CS}) \leftarrow \mathcal{A}(1^{\lambda})$
$(\mathtt{vt}_0, \mathtt{vt}_1) \leftarrow \mathcal{A}()$
$\mathsf{b}_0 := \mathsf{VS.Vote}\langle \mathsf{EA}(), \mathsf{V}_i(\mathtt{vt}_0), \mathsf{prms}, \mathsf{pk}_{\mathsf{TA}}, \mathsf{V}_{\mathtt{El}}, \mathsf{CS}, \mathsf{BB}\rangle$
$\mathsf{b}_1 := \mathsf{VS.Vote}\langle \mathsf{EA}(), \mathsf{V}_i(\mathtt{vt}_1), \mathsf{prms}, \mathsf{pk}_{\mathsf{TA}}, \mathsf{V}_{\mathtt{El}}, \mathsf{CS}, \mathsf{BB}\rangle$
if $\mathsf{b}_0 = \mathsf{b}_1$ AND $\mathsf{b}_1 \neq \perp$ **then**
  return 1
else
  return 0
end

From this definition it is evident, that a voting system that casts the votes in plaintext, cannot possess individual verifiability. In this respect, traditional voting systems do not possess individual verifiability. A simple way to achieve this property in electronic voting is to use a probabilistic encryption scheme like ElGamal [Gam85].

A similar game can be used in the case of a voting scheme that supports internal authentication, by a registration authority that creates and distributes credentials to voters. Again, the adversary generates the parameters for the voting system and simulates the RA in registering the voters and generating their private and public credentials. A sanity check is performed that all voters have different private keys, since two voters with the same private key are essentially the same. Subsequently the adversary selects two honest voters and two voting options for them. The adversary simulates the RA in executing the vote-authorization and casting protocol with the voter to produce the ballots. Even though the adversary participates in the protocol, its input partly originates from the honest voters. As a result, at least these parts will be unique and therefore distinguishable.

Algorithm 4.2: $\mathsf{IndVer}^{\mathsf{int}}_{\mathcal{A},\mathsf{VS}}$ from [SFC15]

Input : security parameter λ
Output: $\{0,1\}$

$(\mathsf{prms}, \mathsf{pk}_{\mathsf{RA}}, \mathsf{sk}_{\mathsf{RA}}, \mathsf{pk}_{\mathsf{TA}}, \mathsf{CS}) \leftarrow \mathcal{A}(1^{\lambda})$
$\left\{(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{VS.Register}\langle \mathcal{A}(\mathsf{sk}_{\mathsf{RA}}), \mathsf{V}_i() \rangle\right\}_{i=1}^{n}$
if $\exists (i,j) : \mathsf{sk}_i = \mathsf{sk}_j$ AND $i \neq j$ **then**
  return 0
end
$\mathsf{VR} := \{\mathsf{pk}_i\}_{i=1}^{n}$
$\mathsf{V}_{\mathtt{Corr}} \leftarrow \mathcal{A}(\textbf{corrupt})$
$(\mathtt{vt}_0, \mathtt{vt}_1, i, j) \leftarrow \mathcal{A}()$
if $i, j \in \mathsf{V}_{\mathtt{Corr}}$ OR $i = j$ **then**
  return 0
end
$\mathsf{b}_i := \mathsf{VS.Vote}\langle \mathcal{A}(\mathsf{sk}_{\mathsf{RA}}), \mathsf{V}_i(\mathtt{vt}_0, sk_i), \mathsf{prms}, \mathsf{pk}_{\mathsf{TA}}, \mathsf{V}_{\mathtt{El}}, \mathsf{CS}, \mathsf{BB} \rangle$
$\mathsf{b}_j := \mathsf{VS.Vote}\langle \mathcal{A}(\mathsf{sk}_{\mathsf{RA}}), \mathsf{V}_j(\mathtt{vt}_1, sk_j), \mathsf{prms}, \mathsf{pk}_{\mathsf{TA}}, \mathsf{V}_{\mathtt{El}}, \mathsf{CS}, \mathsf{BB} \rangle$
if $\mathsf{b}_i = \mathsf{b}_j$ AND $\mathsf{b}_i \neq \perp$ **then**
  return 1
else
  return 0
end

Definition 4.2: Individual verifiability

A voting system VS with external (internal) authentication satisfies individual verifiability, if for all adversaries $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that: $\Pr\left[\mathsf{IndVer}^{\mathsf{ext(int)}}_{\mathcal{A},\mathsf{VS}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$

4.3.2 Universal verifiability

Universal verifiability is a concept first explored in [SK95] to contrast individual verifiability in mixnet-based anonymous channels. While individual verifiability guaranteed delivery for a single message, or, in the case of voting, information that a single vote was included in the tally, universal verifiability allows every interested party, internal or external, to verify that all messages were processed correctly or that all the votes were tallied. Universal verifiability is the most studied property of electronic voting systems with many definitions present in the literature (e.g. [Ben87; JCJ05; KTV12b; KZZ15a; Cor+14; SFC15]) and many more, surveyed in [Cor+16].

The essence of universal verifiability is that the adversary cannot come up with a tally $\mathsf{T}_{\mathcal{A}}$ that is different from the correct tally of the election, along with fabricated adversarial evidence BB, $\pi_{\mathsf{T}_{\mathcal{A}}}$ that cause the incorrect tally to pass verification. In order to express the correct result of the election, we make use of a function correct – tally that retrieves the ballot contents from each ballot in the BB and provides them to the result function in order to calculate the fair objective outcome of the election, regardless of the influence of corrupted parties. It serves as a baseline to compare the output of the tally function which is fed from the contents of the BB. The various definitions in the literature define many ways to compute this result function in an ideal way; actually, only the corrupted voters are of interest. In [KZZ15a], there must exist an extractor algorithm that on input the election transcript can extract the votes and compute the result. In other definitions [KTV10; Cor+14], only the existence of such votes is required, not their exact specification. In [JCJ05] a game-based definition of correctness is given, and assuming the tally is correct then verifiability is defined against it. For the correct – tally function we use the definition from [SFC15], which states that a candidate component of a tally is l, if and only if there are exactly l ballots in the BB cast for the specific candidate. The game in Algorithm 4.3 from [SFC15] captures the essence of of universal verifiability.

In the case of voting with credentials, the definition must be adapted. This is the point of the game in Algorithm 4.3. The result function changes though to support only valid votes and not all cast votes, since some of the votes cast will not be counted.

Based on these games the definition of universal verifiability can be provided:

Algorithm 4.3: $\mathsf{UniVer}^{\mathsf{Ext}}_{\mathcal{A},\mathsf{VS}}$ from [SFC15]

Input : security parameter λ
Output: $\{0,1\}$

$(\mathsf{prms}, \mathsf{pk}_{\mathsf{TA}}, \mathsf{CS}, \mathsf{BB}, \mathtt{T}_{\mathcal{A}}, \pi_{\mathtt{T}_{\mathcal{A}}}) \leftarrow \mathcal{A}(1^{\lambda})$
$\mathtt{T} \leftarrow \mathsf{correct-tally}(\mathsf{BB})$
if $\mathtt{T}_{\mathcal{A}} \neq \mathtt{T}$ AND $\mathsf{VS.Verify}(\mathtt{T}_{\mathcal{A}}, \pi_{\mathtt{T}_{\mathcal{A}}}, \mathsf{prms}, \mathsf{pk}_{\mathsf{EA}}, \mathsf{BB}, \mathsf{CS}, \mathtt{V}_{\mathtt{El}}) = 1$ **then**
  return 1
else
  return 0
end

Algorithm 4.4: $\mathsf{UniVer}^{\mathsf{ext}}_{\mathcal{A},\mathsf{VS}}$ from [SFC15]

Input : security parameter λ
Output: $\{0,1\}$

$(\mathsf{prms}, \mathsf{pk}_{\mathsf{RA}}, \mathsf{sk}_{\mathsf{RA}}, \mathsf{pk}_{\mathsf{TA}}, \mathsf{sk}_{\mathsf{TA}}, \mathsf{CS}) \leftarrow \mathcal{A}(1^{\lambda})$
$\left\{(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{VS.Register}\langle \mathcal{A}(\mathsf{sk}_{\mathsf{RA}}), \mathtt{V}_i() \rangle\right\}_{i=1}^{n}$
if $\exists (i,j) : \mathsf{sk}_i = \mathsf{sk}_j$ AND $i \neq j$ **then**
  return 0
end
$\mathsf{VR} := \{\mathsf{pk}_i\}_{i=1}^{n}$
$(\mathsf{prms}, \mathsf{pk}_{\mathsf{TA}}, \mathsf{CS}, \mathsf{BB}, \mathtt{T}_{\mathcal{A}}, \pi_{\mathtt{T}_{\mathcal{A}}}) \leftarrow \mathcal{A}(\{\mathsf{sk}_i\}_{i=1}^{n}, \mathsf{VR}, 1^{\lambda})$
$\mathtt{T} \leftarrow \mathsf{correct-tally}(\mathsf{BB})$
if $\mathtt{T}_{\mathcal{A}} \neq \mathtt{T}$ AND $\mathsf{VS.Verify}(\mathtt{T}_{\mathcal{A}}, \pi_{\mathtt{T}_{\mathcal{A}}}, \mathsf{prms}, \mathsf{pk}_{\mathsf{EA}}, \mathsf{BB}, \mathsf{CS}, \mathtt{V}_{\mathtt{El}}) = 1$ **then**
  return 1
else
  return 0
end

Definition 4.3: Universal verifiability

A voting system VS with external (internal) authentication satisfies universal verifiability, if for all adversaries $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that: $\Pr\left[\mathsf{UniVer}^{\mathsf{ext(int)}}_{\mathcal{A},\mathsf{VS}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$

Beyond these essential universal verifiability definitions there are other things that must be considered. [Cor+16] highlights the following points:

- *Behavior of honest voters toward verification of their ballots.* Early definitions of universal verifiability [Ben87], expect all the voters to perform the verification process (either for ballot casting or for tallying). However, this not what happens in reality. Most voters do not verify that their votes are present in the BB nor that the system took them into account. As a result, an adversary can drop or alter votes without being detected. The latter is of course more serious. A verifiability definition should have a bound on the number of honest voters that do not perform the verification process.
- *Existence of a registration authority.* Although universal verifiability is mainly concerned with the tallying process, the latter can depend on the eligibility of voters. Identification, authentication and eligibility is usually performed by a registration authority and represented by handling tokens to the voters that function as credentials. A corrupt registration authority can handle the same credential to many voters or handle invalid credentials to some others leading to not counted votes. Furthermore, it can create credentials for non-existent voters or many credentials for one voter leading to ballot stuffing. All these must be taken into account during tallying.
- *Behavior of the* BB. In many voting systems (such as the initial version of Helios [Adi08] and the JCJ scheme [JCJ05]), the BB plays only a passive role, as a datastore of votes. 'Smarter' BB can validate proofs of correct ballot formation or perform duplicate weeding (i.e. check that there is no identical copy of the ballot ciphertext). In voting systems with RAs, such as Helios-C [Cor+14] or Belenios [CGG19], it can additionally validate the credentials of the voters. In these cases. a corrupt BB can drop votes claiming an invalid credential or stuff ballots for voters that did not cast any.

Although the last two observations are related to eligibility verifiability, they also affect the outcome of the elections. As a result, they are part of universal verifiability as well. In fact, in systems with a RA, [Cor+14] two types of universal verifiability can be defined:

- *Weak* universal verifiability assumes that both the BB and the RA are honest. The TA is corrupt as always.
- *Strong* universal verifiability assumes that the BB and the RA are not *concurrently* dishonest. As a result, in order to be verifiable, a voting system must withstand attacks by an adversary that controls both the TA and the BB, while the RA is honest and attacks by an adversary that controls both the TA and the RA but not the BB.

Weak universal verifiability can be defined using the game in Algorithm 4.5 from [Cor+14]. In this game, the adversary generates the keying material of the TA that is under his control. Since the RA and the BB are assumed honest, he can access them only through the respective oracles **Register**, **Cast**, which create the voter credentials and cast the ballots respectively to avoid ballot stuffing. The oracle **Vote** represents the votes of the honest voters, which are maintained as tuples in the list `Hon`. The corruption of voters by the adversary occurs through calls to the **Corrupt** oracle, which reveal the voter credentials maintained in the list `Corr`. The $\mathcal{A}$ produces a tally after he has invoked the oracles at will. The adversary loses the game if it cannot cast more ballots than the maximum number of corrupt voters and all the votes of the honest voters are taken into account, assuming that the tallying function admits *partial tallying* i.e. $\mathsf{result}(A \cup B) = \mathsf{result}(A) \oplus \mathsf{result}(B)$ for some commutative operation $\oplus$.

Strong universal verifiability can be defined using the games in Algorithm 4.6 and Algorithm 4.7 from [Cor+14] where the BB or the RA are respectively corrupted. In Algorithm 4.6 the adversary controls the casting and in Algorithm 4.6 he controls the registration, therefore the relative oracles are not omitted.

In both games the objective of the adversary is to cause a tally to be accepted if the number of duplicate or stuffed votes exceeds the number of corrupted voters or (some of) the votes of the honest voters that did not check are not taken into account.

4.3.3 Eligibility Verifiability

Eligibility verifiability was first defined in [KRS10] as the property that allows anyone to verify that each tallied ballot was cast by a voter with the right to vote and that no voter cast more than two counted ballots. In the game in Algorithm 4.8, the adversary must produce a valid ballot for a credential that it does not possess i.e. not belonging to a corrupt voter. Note, that even though $\mathcal{A}$ executes the registration phase with the voter, he cannot know the private part of the voter's credential as it is completed with the help of input submitted by the voter. Furthermore, it is assumed following

Algorithm 4.5: Weak universal verifiability game $\mathsf{UniVer}^{\mathsf{Weak}}_{\mathcal{A},\mathsf{VS}}$ from [Cor+14]

Input : security parameter λ
Output: $\{0,1\}$

Oracle Register(i)
  $(\mathsf{pk}_i, (\mathsf{sk}_i, \mathsf{pk}_i)) := \mathsf{VS.Register}\langle \mathsf{RA}(\mathsf{sk}_{\mathsf{RA}}), \mathsf{V}_i() \rangle$
  $\mathsf{V}_{\mathtt{El}} \Leftarrow (i, \mathsf{pk}_i)$
  $\mathsf{V}'_{\mathtt{El}} \Leftarrow (i, \mathsf{pk}_i, \mathsf{sk}_i)$

Oracle Corrupt(i)
  if $i \in \mathsf{V}_{\mathtt{El}}$ **then**
    $\mathsf{V}_{\mathtt{Corr}} \Leftarrow (i, \mathsf{pk}_i, \mathsf{sk}_i)$
  else
    return $\perp$
  end

Oracle Vote$(i, \mathtt{vt})$
  if $i \in \mathsf{V}_{\mathtt{El}}$ AND $i \notin \mathsf{V}_{\mathtt{Corr}}$ **then**
    if $\exists (i, \cdot, \cdot) \in \mathsf{V}_{\mathtt{Hon}}$ **then**
      $\mathsf{V}_{\mathtt{Hon}} := \mathsf{V}_{\mathtt{Hon}} \setminus \{(i, \cdot, \cdot)\}$
    end
    $\mathsf{b} := \mathsf{VS.Vote}\langle \cdot, \mathsf{V}_i(\mathtt{vt}_i^{\mathsf{V}_{\mathtt{Hon}}}, \mathsf{sk}_i), \cdot \rangle$
    $\mathsf{V}_{\mathtt{Hon}} \Leftarrow (i, \mathtt{vt}_i^{\mathsf{V}_{\mathtt{Hon}}}, \mathsf{b})$
  else
    return $\perp$
  end

Oracle Cast(i, b)
  $\mathsf{BB} \Leftarrow (i, \mathsf{b})$

$(\mathsf{prms}, \mathsf{pk}_{\mathsf{TA}}, \mathsf{sk}_{\mathsf{TA}}) \leftarrow \mathcal{A}(1^\lambda)$
$(\mathtt{T}_{\mathcal{A}}, \pi_{\mathtt{T}_{\mathcal{A}}}) \leftarrow \mathcal{A}^{\mathbf{Register}, \mathbf{Corrupt}, \mathbf{Vote}, \mathbf{Cast}}()$
if $\mathsf{VS.Verify}(\mathtt{T}_{\mathcal{A}}, \pi_{\mathtt{T}_{\mathcal{A}}}, \cdot) = 0$ OR $\mathtt{T}_{\mathcal{A}} = \perp$ **then**
  return 0
end
if $\exists n_{\mathsf{V}_{\mathtt{Corr}}} : 0 \le n_{\mathsf{V}_{\mathtt{Corr}}} \le |V_{\mathtt{Corr}}|$ AND $\exists \{\mathtt{vt}_i^{\mathsf{V}_{\mathtt{Corr}}} \in \mathsf{CS}\}_{i=1}^{n_{\mathsf{V}_{\mathtt{Corr}}}}$:
$T_{\mathcal{A}} = \mathsf{result}(\mathtt{vt}_i^{\mathsf{V}_{\mathtt{Corr}}}) \oplus \mathsf{result}(\mathtt{vt}_i^{\mathsf{V}_{\mathtt{Hon}}})$ **then**
  return 0
else
  return 1
end

Algorithm 4.6: Strong universal verifiability game $\mathsf{UniVer}^{\mathsf{Weak}}_{\mathcal{A},\mathsf{VS}}$ with malicious BB from [Cor+14]

Input : security parameter λ
Output: $\{0,1\}$

Oracle Register(i)
  /* Same as in Algorithm 4.5 */
Oracle Corrupt(i)
  /* Same as in Algorithm 4.5 */
Oracle Vote$(i, \mathtt{vt})$
  /* Same as in Algorithm 4.5 */

$(\mathsf{prms}, \mathsf{pk}_{\mathsf{TA}}, \mathsf{sk}_{\mathsf{TA}}) \leftarrow \mathcal{A}(1^\lambda)$
$(\mathtt{T}_{\mathcal{A}}, \pi_{\mathtt{T}_{\mathcal{A}}}, \mathsf{BB}) \leftarrow \mathcal{A}^{\mathbf{Register},\mathbf{Corrupt},\mathbf{Vote}}()$
if $\mathsf{VS.Verify}(\mathtt{T}_{\mathcal{A}}, \pi_{\mathtt{T}_{\mathcal{A}}}, \cdot) = 0$ OR $\mathtt{T}_{\mathcal{A}} = \perp$ **then**
  return 0
end
$\mathsf{V}_{\mathtt{Chck}} = \{(i^{\mathsf{V}_{\mathtt{Chck}}}, \mathtt{vt}_i^{\mathsf{V}_{\mathtt{Chck}}}, \mathtt{b}_i^{\mathsf{V}_{\mathtt{Chck}}})\}_{i=1}^{|\mathsf{V}_{\mathtt{Chck}}|}$
if $\exists n_{\mathsf{V}_{\mathtt{Corr}}} : 0 \le n_{\mathsf{V}_{\mathtt{Corr}}} \le |\mathsf{V}_{\mathtt{Corr}}|$ AND $\exists\{\mathtt{vt}_i^{\mathsf{V}_{\mathtt{Corr}}}\}_{i=1}^{n_{\mathsf{V}_{\mathtt{Corr}}}}$
$\exists n' : 0 \le n' \le |\mathsf{V}_{\mathtt{Hon}}| - |\mathsf{V}_{\mathtt{Chck}}|$ AND $\exists\{\mathtt{vt}'_i\}_{i=1}^{n'}$ // Honest voters that did not check
$\mathtt{T}_{\mathcal{A}} = \mathsf{result}(\mathtt{vt}_i^{\mathsf{V}_{\mathtt{Corr}}}) \oplus \mathsf{result}(\mathtt{vt}_i^{\mathsf{V}_{\mathtt{Chck}}}) \oplus \mathsf{result}(\mathtt{vt}'_i)$ **then**
  return 0
else
  return 1
end

Algorithm 4.7: Strong universal verifiability game $\mathsf{UniVer}^{\mathsf{Weak}}_{\mathcal{A},\mathsf{VS}}$ with malicious RA from [Cor+14]

Input : security parameter λ
Output: $\{0,1\}$

Oracle Corrupt(i)
  /* Same as in Algorithm 4.5 */

Oracle Vote$(i, \mathtt{vt})$
  /* Same as in Algorithm 4.5 */

Oracle Cast(i, b)
  /* Same as in Algorithm 4.5 */

$(\mathsf{prms}, \mathsf{pk}_{\mathsf{TA}}, \mathsf{sk}_{\mathsf{TA}}) \leftarrow \mathcal{A}(1^\lambda)$
$(\mathtt{T}_{\mathcal{A}}, \pi_{\mathtt{T}_{\mathcal{A}}}) \leftarrow \mathcal{A}^{\mathbf{Corrupt},\mathbf{Vote},\mathbf{Cast}}()$
if $\mathsf{VS.Verify}(\mathtt{T}_{\mathcal{A}}, \pi_{\mathtt{T}_{\mathcal{A}}}, \cdot) = 0$ OR $\mathtt{T}_{\mathcal{A}} = \perp$ **then**
  return 0
end
$\mathsf{V}_{\mathtt{Chck}} = \{(id_i^{\mathsf{V}_{\mathtt{Chck}}}, \mathtt{vt}_i^{\mathsf{V}_{\mathtt{Chck}}}, \mathrm{b}_i^{\mathsf{V}_{\mathtt{Chck}}})\}_{i=1}^{|\mathsf{V}_{\mathtt{Chck}}|}$
if $\exists n_{\mathsf{V}_{\mathtt{Corr}}} : 0 \leq n_{\mathsf{V}_{\mathtt{Corr}}} \leq |\mathsf{V}_{\mathtt{Corr}}|$ AND $\exists \{\mathtt{vt}_i^{\mathsf{V}_{\mathtt{Corr}}} \in \mathsf{CS}\}_{i=1}^{n_{\mathsf{V}_{\mathtt{Corr}}}}$
$\exists n' : 0 \leq n' \leq |\mathtt{Hon}| - |\mathtt{Chck}|$ AND $\exists \{\mathtt{vt}'_i\}_{i=1}^{n'}$ // Honest voters that did not check
$\mathtt{T}_{\mathcal{A}} = \mathsf{result}(\mathtt{vt}_i^{\mathsf{V}_{\mathtt{Corr}}}) \oplus \mathsf{result}(\mathtt{vt}_i^{\mathsf{V}_{\mathtt{Chck}}}) \oplus \mathsf{result}(\mathtt{vt}'_i)$ **then**
  return 0
else
  return 1
end

[SFC15], that $\mathcal{A}$ learns some credentials during the voting process, from voters that cast their ballots, by the means of coercion. They are assumed to be part of a set $V_{\texttt{Coer}}$.

Algorithm 4.8: $\mathsf{EliVer}^{\mathsf{ext}}_{\mathcal{A},\mathsf{VS}}$ from [SFC15]

Input : security parameter λ
Output: $\{0,1\}$

$(\mathsf{prms}, \mathsf{pk}_{\mathsf{RA}}, \mathsf{sk}_{\mathsf{RA}}, \mathsf{pk}_{\mathsf{TA}}, \mathsf{CS}) \leftarrow \mathcal{A}(1^\lambda)$
$\left\{(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{VS.Register}\langle \mathcal{A}(\mathsf{sk}_{\mathsf{RA}}), \mathsf{V}_i() \rangle\right\}_{i=1}^{n}$
if $\exists (i,j) : \mathsf{sk}_i = \mathsf{sk}_j$ AND $i \neq j$ **then**
  return 0
end
$\mathsf{VR} := \{\mathsf{pk}_i\}_{i=1}^{n}$
$\mathsf{V}_{\texttt{Corr}} \leftarrow \mathcal{A}(\textbf{corrupt})$
$(\texttt{vt}, j, \mathsf{b}_j) \leftarrow \mathcal{A}(\{\mathsf{sk}_i\}_{i \in \cup \mathsf{V}_{\texttt{Coer}}}, \mathsf{VR}, 1^\lambda)$
if $\exists j : \mathsf{b}_j := \mathsf{VS.Vote}\langle \mathcal{A}(\mathsf{sk}_{\mathsf{RA}}), \mathsf{V}_j(\texttt{vt}, sk_j), \mathsf{prms}, \mathsf{pk}_{\mathsf{TA}}, \mathsf{CS}, \mathsf{BB} \rangle$ **then**
  return 1
else
  return 0
end

Definition 4.4: Eligibility verifiability

A voting system VS with external authentication satisfies eligibility verifiability, if for all adversaries $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that: $\Pr\left[\mathsf{EliVer}^{\mathsf{ext}}_{\mathcal{A},\mathsf{VS}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$

Definition 4.5: Election verifiability

A voting scheme with external authentication provides election verifiability if it provides individual and universal verifiability.
A voting scheme with internal authentication provides election verifiability if it provides individual, universal and eligibility verifiability.

As we saw earlier, revealing if a voter participated in an election is illegal in some jurisdictions. A relevant variation, private eligibility verifiability, was proposed in [KTV15].

4.4 Coercion resistance

Coercion resistance or incoercibility is one of the most important goals for the realization of remote electronic voting. Its absence means that there is no way to make sure whether a voter is expressing her own will or is following the commands of a coercer standing over her shoulder. However, it can also be a valid concern for supervised

voting as well, as the short ballot assumption and the relevant real-world attacks of section 1.2 reveal. In the literature there are models for the supervised setting, the remote setting as well general models.

Coercion resistance is treated as an extended form of ballot secrecy; while the latter protects honest voters only from passive adversaries, incoercibility protects from the combination of dishonest voters (receipt-freeness) that want to sell their votes and active adversaries that want to dictate a voting strategy on the voter. These gradient notions of privacy are well known in the voting literature and are respectively referred to as *IO coercion, semi-honest coercion* and *active* coercion, in [Alw+15]. We begin by reviewing the respective notions and models.

4.4.1 Receipt-Freeness

Ballot secrecy is not an optional security property of voting systems. The ballot contents must stay secret, whether the voter wishes it or not. At first glance, this is at odds with receipts generated for the purposes of individual verifiability [BT94]. The Vote functionality, in the model in section 4.1, generates a receipt, that the voter can input to the VerifyBallot functionality to check that her vote will be counted. As we saw in section 4.2, this receipt can simply be the randomness used to encrypt the voter choice in the ballot. The voter can use this receipt to recreate the ballot and compare it with the one that resides in the BB. However, the same sequence of actions, can be performed by a malicious voter wanting to sell her vote or by an honest voter that is coerced to vote in a specific way. To thwart this attack a voting system must emulate the 'plausible deniability' offered by physical voting booths [BT94]. This can be simply achieved by not generating receipts as in traditional elections. But that, albeit the fact that it is impossible with probabilistic encryption schemes, would not satisfy individual verifiability. The challenge is on how to combine the two.

The first definition of receipt-freeness and the first such protocol was given in [BT94]. Informally, a voting protocol is receipt-free if it can there exist no other protocol that provides the same inputs and provides a receipt. The main way proposed to implement this property is that the voter does not produce the encrypted ballots herself, but instead they are produced by the EA, which also generates a public proof to convince about their well-formedness. The voter enters a voting booth (cf. subsection 2.1.2) where he is given private data in a deniable manner, to be convinced about the plaintext encrypted, so that he can choose her vote. If the voter is forced to reveal these private data, she can provide a 'forgery' that is equally convincing to the coercer. The problem with [BT94] is a convincing realization of the voting booth. Later, in [SK95], the voting booth requirement is relaxed, by using a one-way untappable channel from the EA to the voter.

Another formal definition of receipt freeness was given in [Oka97]. According to it, a voting system is receipt-free, if a voter V exists such that for any adversary $\mathcal{A}$, V can cast a vote vt different from the vote required by $\mathcal{A}$, such that the TA counts this vote and $\mathcal{A}$ accepts the public view of the protocol (BB). They also propose a modification of the [FOO92] voting scheme, which makes use of the anonymous channel already present and an untappable channel from the voter to the EA.

Later, the construction of [HS00] provide a generic way to implement the voting booth, only requiring an untappable channel (one-way voting booth from the EA to the voter). The EA again constructs deterministic encryptions of ballots, using some predetermined randomness. The ballot list is shuffled i.e. permuted and reencrypted. Each voter is presented with the shuffled list and is given the designated verifier proof of correct reencryption of subsection 2.5.2 through the untappable channel. As a result, the voter learns to which candidates the transformed ballots correspond. For this proof to be constructed each voter is assumed to hold a private key. To cast the ballot, the voter does not perform any computation, but simply points to her selection. If coerced, she can simulate the received proof to show that she complied with the coercer demands. Since the coercer cannot be sure of what is received, she cannot sell her vote. Note that a different shuffle must be performed for each voter. In a different case if the coercer was a voter (or if she controls some corrupted voters), he could learn the permutation and deduce, how her target voted.

Game-based definitions of receipt-freeness These first schemes, claimed receipt-freeness intuitively, without providing formal definitions or proofs. According to [FQS19], the formal analysis of receipt-freeness was in the symbolic setting (e.g. [JV06]) until DEMOS [KZZ15a], which was the first game-based receipt-freeness definition. Their definition is a side-effect of their privacy definition. The relevant part in their privacy/receipt-freeness game is that the adversary presents two voting options to the challenger who plays the role of an honest voter as input to the Vote functionality. The challenger flips a coin $\mathfrak{b}$ and posts one of them to the BB. The challenger returns the ballot and if $\mathfrak{b} = 0$ the real transcript of the interaction, or a simulated view otherwise. This ability to simulate is according to [KZZ15a] the reason the DEMOS has receipt-freeness. [FQS19] mentions several problems with this first game-based definition. Firstly, it inherits the problem of the respective privacy definitions that it does not apply to all voting rules (cf. section 4.5). Secondly, it is incomplete as it excludes schemes were receipt-freeness is achieved through re-voting. More importantly, it is focused on supervised voting schemes, where it is difficult for the adversary to obtain credentials, before or during vote-casting.

Another game-based definition of receipt-freeness was defined for the BeleniosRF

voting scheme [Cha+16]. This recent scheme adds receipt-freeness to the Belenios voting scheme [CGG19], a variation of Helios-C [Cor+14], where each voter is assumed to have a private key to sign the ballot. BeleniosRF uses the cryptographic primitive of signatures on *rerandomizable* ciphertexts proposed in [Bla+11]. This primitive consists of an encryption scheme Enc that allows reencryption and a digital signature scheme $\mathcal{DS}$, such that when a ciphertext created by Enc is rerandomized, the signature is adapted to verify on the new ciphertext. These functionalities can be performed without any access to secret keys. BeleniosRF utilizes signatures on rerandomizable ciphertexts to alter the randomness of an encrypted ballot. When the voter casts the ballot, a rerandomizing server reencrypts the ciphertext and the accompanying signature, before posting it to the BB. As a result, the randomness used to encrypt the plaintext, cannot be used as a receipt, since it has been altered as well. The rerandomizing server is part of the EA and adapts the proofs of validity that accompany the ballot. Note that this does not break individual or eligibility verifiability as the rerandomized signatures are still publicly verifiable. However, BeleniosRF is not universally verifiable as a collusion of the RA and the rerandomizing server can alter votes.

More interestingly, the authors of [Cha+16] propose the first game-based definition of receipt-freeness. Their definition is an extension of BPRIVof [Ber+15] (cf. Definition 4.7). It uses two BBs, where $\mathcal{A}$ has access to only one according to the value of $\mathfrak{b}$. Tallying occurs always on BB_0 and if $\mathcal{A}$ views BB_1 the correctness proof is simulated. There are also the same oracles as in the game in Algorithm 4.10, where $\textbf{Vote}(i, \mathtt{vt}_0, \mathtt{vt}_1)$ casts a ballot for $\mathtt{vt}_0$ in BB_0 and a ballot for $\mathtt{vt}_1$ in BB_1 (for the same V_i), $\textbf{Cast}(i, \mathsf{b})$ which casts the same ballot b in both and $\mathsf{Tally}(\mathfrak{b})$ which performs the tally always on BB_0. For the definition of coercion resistance, a new oracle $\textbf{Receipt}(i, \mathsf{b}_0, \mathsf{b}_1)$ is defined which posts b_0 in BB_0 and b_1 in BB_1. Note that **Receipt** differs from **Vote** as it operates on ballots and not on plain votes.

A voting scheme is receipt-free if the adversary cannot distinguish which board he is viewing, except with negligible probability.

Definition 4.6: Strong Receipt-Freeness from [Cha+16]

A voting scheme VS provides strong receipt-freeness if for every PPT algorithm $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\lambda)$ and an efficient algorithm Sim such that:

$$\Pr\left[\mathsf{sRF}^0_{\mathcal{A},\mathsf{VS}}(\lambda) = 1\right] - \Pr\left[\mathsf{sRF}^1_{\mathcal{A},\mathsf{VS}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$$

The intuition of why the inclusion of **Receipt** is enough to model receipt-freeness is that even if the $\mathcal{A}$ encodes some data (that can serve as a receipt) into $\mathsf{b}_0 \in \mathsf{BB}_0$ and

Algorithm 4.9: $\mathsf{sRF}^{\mathfrak{b}}_{\mathcal{A},\mathsf{VS}}$ from [Cha+16]

Input : security parameter λ
Output: $\{0,1\}$

Oracle Receipt$(i, \mathsf{b}_0, \mathsf{b}_1)$
  if $\mathsf{Valid}(\mathsf{b}_0, \mathsf{BB}_0)$ AND $\mathsf{Valid}(\mathsf{b}_1, \mathsf{BB}_1)$ **then**
    $\mathsf{BB}_0 \Leftarrow \mathsf{b}_0$
    $\mathsf{BB}_1 \Leftarrow \mathsf{b}_1$
  else
    return $\perp$

Oracle Vote$(i, \mathtt{vt}_0, \mathtt{vt}_1)$
  `/* Same as in BPRIV  Algorithm 4.10  */`

Oracle Cast(i, b)
  `/* Same as in BPRIV  Algorithm 4.10  */`

Oracle Tally$(\mathfrak{b})$
  `/* Same as in BPRIV  Algorithm 4.10  */`

$(\mathsf{prms}, \mathsf{pk}_{\mathsf{TA}}, \mathsf{sk}_{\mathsf{TA}}) \leftarrow \mathsf{VS.Setup}(1^\lambda)$
$\mathsf{CS} \leftarrow \mathcal{A}()$
$\mathfrak{b}' \leftarrow \mathcal{A}^{\mathbf{Receipt},\mathbf{Vote},\mathbf{Cast},\mathbf{Tally}}(\mathsf{prms}, \mathsf{pk})$
return $\mathfrak{b} = \mathfrak{b}'$

different data in $\mathsf{b}_1 \in \mathsf{BB}_1$ so that he can distinguish the two BBs he will not be able to do so. It is especially important, to take into account that receipt-freeness extension of BPRIV only applies to voting schemes where ballot casting is not interactive and where the voter casts a single ballot. As a result, it cannot capture coercion-resistant schemes, which by definition, are receipt-free and are based on re-voting, or other techniques. A valid critique made by [FQS19] is that it requires that the ballot is changed before being posted to the BB. All in all, it is focused on a particular protocol and is not generic enough. A generic receipt-freeness definition still eludes researchers.

Deniable vote updating Another strategy for receipt-freeness was used in systems of [LHK16; BKV17; Ach+15; LQAT20]. These systems use deniable re-voting or deniable vote updating. The adversary forces the voter to cast a ballot $\mathsf{b}_{\mathcal{A}}$. The voter obeys but can later update the ballot to b_{V} without the adversary noticing this. There are two variations of this technique; the second vote either cancels the first or updates it to match the option that is really preferred by the voter. The latter option is usually found in homomorphic voting systems and depends on the candidate encoding. Additionally, the voter must be aware that of the value of the ballot being canceled to successfully update it. In fact, this technique can be generalized to more than two votes: The voter can cast as many votes as she likes - the vote to be counted in the

end will be the homomorphic sum of all the ballots that belong to her. The idea is that the voter can provide as many receipts as the adversary requests, but $\mathcal{A}$ we will not be sure he has seen them all. For this reason, the votes must not be visibly linked to each other; in [LHK16] this is achieved by using a verifiable shuffle. In [BKV17] this is achieved by casting dummy null votes - however trust is required to the posting agents. Of course, the greatest disadvantage of re-voting based techniques is that if the $\mathcal{A}$ always watches the voter then he can block the voter from updating their vote.

4.4.2 The JCJ coercion resistance framework

The first comprehensive framework for coercion resistance was proposed in [JCJ05]. The goal of voting schemes that provide coercion resistance is not to allow the adversary to perform the coercion attack. This is achieved in a game-theoretic way; the adversary will not be motivated to coerce if he cannot check that his attack succeeded. The JCJ proposal accomplishes this through a combination of two defense techniques: Multiple votes per voter and authentication using anonymous credentials. Each vote is authenticated by an anonymous token. During the registration phase the voter receives a genuine credential. This is meant to be used when the voter is not under coercion and will authenticate the intended vote. Under coercion, she supplies an indistinguishable but fake credential to accompany the vote. The TA must count only the votes that correspond to authentic credentials. This must take place in a verifiable manner for the voter, but without publicly disclosing which votes are discarded so that the coercer cannot verify compliance.

Adversarial model The adversary can be a vote buyer and additionally perform the following attacks:

- Randomization attack: The voter is forced to cast a specific random vote, which is handed by the adversary. His goal in such an attack might be to diminish a known advantage one candidate might have (in a specific precinct).
- Forced abstention attack: The voter is forced not to vote. This attack could be considered a variation of the randomization attack if we consider that there is an extra null candidate representing abstention. However, it is slightly stronger, as even a null vote has a side effect; a message is transmitted. On the other hand, abstention means that the voter does not send a single message. While this attack is quite simple in nature, it is very difficult to defend against, as we will see later.
- Simulation attack: The adversary can force the voter to reveal her private key and then vote on her behalf. This attack is stronger than the randomization and

forced abstention attacks since an attacker knowing private keys, can perform both.

The fake credential mechanism can thwart all these attacks. If the voter is requested to vote randomly, then the voter does, so using the fake credential. If the voter is forced to give up her credential in a simulation attack, then the voter gives up her fake credential. If the voter is ordered to abstain, then she casts no vote with the fake credential. In all cases the real vote is cast using the authentic credential.

Assumptions In order to provide coercion resistance the JCJ framework makes the following assumptions:

- Moment of privacy: It is assumed that the voter is not controlled by the coercer at all times. This is a minimal assumption; a totally controlled voter cannot deceive the adversary.
- Untappable registration: If the adversary can obtain the registration credentials then he can easily mount a simulation attack. While the general use of untappable channels hinders the scalability of the voting system, the registration phase can occur only once and the credentials obtained can be used for many elections. Furthermore, this untappability is permanent; the registration transcript cannot leak. For this, three possibilities are offered:
  - The majority of the RA is honest and there is a mechanism for the secure erasure of the registration transcript.
  - No corruption of the RA is allowed, and the registration transcript can be simulated.
  - The minority of the RA is corrupted, and the voter knows their identities so that he can simulate their transcripts.
- Anonymous casting: An adversary that can map ballots to voter identities can at least find out if a forced abstention attack has been successful. As a result, casting through a channel that leaks identities is incompatible with coercion resistant.
- Corruptions: The adversary controls a minority of the members of the EA and a minority of all voters. The corruption of voters is static, while the corruption of authorities is adaptive.
- Coercer uncertainty: The attacker must be unaware of how honest voters vote. Otherwise, the tally could leak information on a vote contents, in the manner

we described in section 4.5. This can be achieved if there are voters (e.g. pro-democracy organizations) that cast dummy votes, that should be counted, without this fact being conferred to the coercer.

Model In order to formalize the coercion deception strategy of the voters a function fakekey is assumed that produces an anonymous credential indistinguishable from the real one. While few details are given in [JCJ05], it is assumed that the fakekey has hidden randomness from the adversary while it allows him to verify that the credential presented has been generated in a normal manner by the registration authorities, without allowing him to guess if it is authentic or not.

To reason about coercion resistance, the JCJ framework defines *two* games between the adversary (coercer) and a challenger (voter). In the *real* game, the challenger setups the election and $\mathcal{A}$ statically chooses which voters to corrupt. The latter yield their private credentials. The coercer chooses a single (uncorrupted) voter to coerce. Now the challenger flips a coin $\mathfrak{b}$. If $\mathfrak{b} = 0$, the challenger will evade coercion. For this reason fakekey is invoked and its result is handed to $\mathcal{A}$. Subsequently, the voter casts her real ballot. To make the attack stronger the preference of the voter is selected by $\mathcal{A}$. If $\mathfrak{b} = 1$, the voter allows to be simulated by providing its credential to $\mathcal{A}$. The challenger then proceeds to tally the election and provides to the adversary the result with a proof of correctness. The coercer must guess $\mathfrak{b}$ i.e. check if the voter followed his instructions.

The probability of success of the adversary in correctly guessing $\mathfrak{b}$ is compared to the corresponding probability of winning an *idealized* version of the above game, where he cannot use the private credentials of the corrupt voters. The rationale behind this game is to exactly measure the effect of the cryptographic primitives and the voting protocol to the probability of the adversary discovering if his attack succeeded or not. For instance, the adversary could win the real game by instructing the coerced voter to cast a ballot for a candidate that is expected to receive no votes. If tallying indicates that this candidate still does not receive any vote, then $\mathcal{A}$ will know that the coerced cheated. This, however, can happen in any voting scheme, regardless of the primitives and the workflow used to stop coercion attacks. The ideal game will exhibit such a behavior as well, so the comparison with the real game will is meant to nullify its effect. More specifically in the ideal game, $\mathcal{A}$ merely enumerates the preferences of the corrupt voters to the challenger. The coerced voter does not use fakekey but always hands the real credential to $\mathcal{A}$. If $\mathfrak{b} = 0$ the coerced voter simply casts the real ballot using the real credential. For this game to work, the real tally function cannot be used. As a result, an *ideal* tally is defined. This version handles the honest voter ballots as in the real case. The corrupt voter's choices are simply

added to the result. For the coerced voter, if $\mathfrak{b} = 0$ then it adds only her real vote and not the one cast by the adversary, so as not to exhibit one more ballot in the result which will differentiate it from the real tally and the case of $\mathfrak{b} = 1$. The ideal tally function also checks for valid credentials and duplicate ballots.

Still, however, the adversary could bypass the cryptographic primitives of the voting scheme to discover if the voter followed his instructions. For instance, in a forced abstention attack where no other honest voters vote, the coercer can simply check if the total number of ballots is greater than the number of ballots cast by the coerced voters. This attack is thwarted by the assumption about the coercer uncertainty.

JCJ Implementation

Additionally, [JCJ05] proposed a protocol that implements this framework, while proceeds in the following steps:

1. **Setup Phase:** Key pairs are created for registration RA and tallying authorities TA with keys $(\mathsf{pk}_{\mathsf{RA}}, \mathsf{sk}_{\mathsf{RA}})$ and $(\mathsf{pk}_{\mathsf{TA}}, \mathsf{sk}_{\mathsf{TA}})$, respectively. Corruption of a minority of members of both authorities can be tolerated for coercion resistance.

2. **Registration Phase:** During this phase, the voter identity is validated, and the voter credentials are generated. They comprise a public and private pair derived from a random anonymous token θ_i which can be as simple as a random number (or more elaborate as is the case in later refinements). The anonymous token is encrypted with the public key of the talliers and placed on the BB $C_{i1} = \mathsf{Enc}_{\mathsf{pk}_{\mathsf{TA}}}(\theta_i)$. This is the public component, while the token itself is the private component, which is transmitted using the untappable channel. The voter roll is built from the public encryptions $\mathsf{VR} = \{(i, C_{i1})\}_{i=1}^{n}$. To avoid corruption during this phase, the assumptions of section 4.4.2 apply.

3. **Voting Phase:**

   - Ballots are cast to the BB as usual.

   - The ballot is a modified ElGamal encryption of both the candidate choice and the credential. This modified version, proved in [JCJ05] to have the IND-CPA property, accommodates the needs of the proof of coercion resistance. More specifically, the ballot has the following form:

     $$\mathsf{b} = (\mathsf{Enc}_{\mathsf{pk}_{\mathsf{TA}}}(\mathtt{vt}_i), \mathsf{Enc}_{\mathsf{pk}_{\mathsf{TA}}}(\theta_i)) = ((g_1^{r_1}, g_2^{r_1}, \mathtt{vt}_i \cdot h^{r_1}), (g_1^{r_2}, g_2^{r_2}, \theta_i \cdot h^{r_2}))$$

     where r_1, r_2 are random values.

- The ballot is completed by proofs of knowledge of vote and credential, proof of vote validity, namely that the candidate index is valid, and proof that the first two components of the encryptions use the same randomness. These proofs are essential to achieving coercion resistance, by the following rationale:
  - Since the voter roll is public, an attacker against verifiability might spoof a credential by reencrypting a voter roll entry. A legitimate voter must prove that he knows the credential.
  - An invalid vote might indicate a forced abstention or a randomization attack.
- The voting phase takes place using an anonymous channel. This is particularly important as to thwart the forced abstention attack and to make the coercer unaware of the actual vote position in the BB.

4. **Tallying Phase:** The authorities collect the ballots from the BB and compute the election result. Because there are more ballots than voters, on account of coercion evasion, there must some preprocessing, in order to distinguish the votes that must be counted. This pre-tallying phase was aptly named *vote authorization* by [Sch+11] and consists of the following sub-phases:
   - **Invalid Ballot Removal:** Before counting begins, the proofs of correctness are extracted and verified. Ballots with invalid proofs are discarded.
   - **Duplicate Ballot Removal:** If multiple ballots correspond to the same credential, they are filtered and a single ballot per credential is kept. To this end:
     - From ballots with valid proofs, two lists A_1, B_1 are created. A_1 contains the encrypted votes and B_1 the encrypted credentials.
     - Encrypted credentials from B_1 are compared with each other using PET and if the result is 1, then it means that there are votes with duplicate credentials. Only one item per credential is kept according to some rule. The changes are cascaded to A_1.
   - **Fake Ballot Removal:** Votes with fake credentials are eliminated, with the following procedure:
     - The lists A_1, B_1 as well as the voter roll VR are forwarded to a verifiable Shuffle to anonymize their contents. Let A_2, B_2, VR' be the new lists.

* Each encrypted credential in B_2 is compared to each item in VR' using PET . If the result indicates 0, then the entry in B_2 and the corresponding one in A_2 are discarded. A new list of encrypted votes A_3 is returned from this process. Because of the use of the Shuffle the coercer loses track of the credential index in the voter roll and the vote index in the BB. If this vote corresponds to a fake credential it will be removed from BB before counting without raising any suspicion.

– The resulting list of encrypted votes A_3 is decrypted and the votes are counted to create the tally T. Proof of correct computation is also provided π_{T}.

This protocol is proved in [JCJ05] to be coercion resistant by comparing two games. In the *real* coercion resistance game, the $\mathcal{A}$ executes the voting protocol while also controlling a set of corrupted voters $V_{\mathtt{Corr}}$. The goal of the game is to tell if a targeted voter cr has followed his instruction. To this end a coin is tossed and if $b = 1$ the voter allows to be impersonated, while if $b = 0$ the voter invokes the fakekey functionality, provides its output to $\mathcal{A}$ and then uses her valid credential to cast the real vote. A simplistic model would declare the scheme to be coercion resistant if the coercer could not successfully distinguish the coin result after viewing the BB contents, the tally, and the proof of correct computation. However, this is not the case, as the tally could leak information to the coercer, in a manner like the one we saw in the case of privacy. As a result, the advantage of the coercer should be calculated in relation to an ideal voting system that ideally tallies the votes. This comparison will reveal how much the actual voting system impacts coercion. To this end, an *ideal* voting experiment is defined where the coercer does not have access to the contents of the BB, but only to the outputs of the tally, i.e. the results and the number of canceled votes. Additionally, $\mathcal{A}$ does not have access to the private credentials of the voters. The authors of [JCJ05] prove that their protocol is coercion resistant if the DDH assumption holds.

The main problem of the [JCJ05] protocol is its time complexity. Assuming that there are n eligible voters in VR and v votes cast, then the total number of PET performed in B_1 are $\mathcal{O}(v^2)$ and the total number of PET performed in B_2 is $\mathcal{O}(nv)$. Note that $v > n$ because of multiple votes. This makes the JCJ framework inefficient for practical usage, considering both running time and space required. Ideally, such a scheme should operate using $\mathcal{O}(n + v)$ steps.

JCJ variations

Various efforts in the literature have tried to overcome the performance bottleneck of the JCJ scheme. In this section, we review the most important ones, where importance is defined in relation to our proposed protocol in section 5.2.

A first line of relevant proposals are due to [Smi05; WAB07]. They achieve by blinding the credentials and then stripping off the encryption randomization. As a result, they efficiently compare by using a hash table. More specifically, if $(a, b) = (g^r, \theta h^r)$ is an ElGamal encryption of credential θ, where s is the shared private key and $h = g^s$ is the public key and r the randomness employed, then the randomness can be removed as follows:

- The authorities select a second private key z which should be generated in a distributed fashion.
- They blind the ciphertext by computing $(a^{sz}, b^z) = (g^{rsz}, \theta^z g^{rsz})$.
- Subsequently they divide the components which leave only the plaintext credential θ^z in a blinded form.
- The ciphertext is transformed to a deterministic fingerprint.
- Duplicates can be discovered by using a hash table in linear time.

[Smi05] proposes, with efficiency in mind, to split the blinded credential to two pieces and use the first half as a key to the hash table. This approach is problematic since it will imply an increase in hashing collisions. This side-effect is particularly problematic during fake detection and removal, since a valid credential might collide with a fake one, and be wrongly removed, thus altering the election results, and violating fairness. [WAB07] skips this problem by using the complete blinded credential and appropriately adjusting the system parameters, such that θ^z is unique.

Both these approaches suffer from a variant of the Pfitzmann tagging attack [Pfi94] initially observed in [Ara+10]: The coercer obtains the voter fake credential θ (in an impersonation attack or by forcing the voter to reveal it). Then it casts two votes: one accompanied with an encryption of θ and one with an encryption of θ^2. Since both θ and θ^2 are fake, the blind hash table method will output both θ^z and θ^{2z} as invalid. The coercer can square every item found in the rejected list and if a match is found, then he will know for a fact that the voter supplied him with a fake credential.

[Spy+12] noticed that the tagging attack is irrelevant during duplicate removal. As a result, the blind hash tables can be used there thus achieving efficiency, as this phase can be performed in time $\mathcal{O}(\nu)$. In order to achieve removal of ballots with fake credentials in $\mathcal{O}(n)$ steps, the authors of [Spy+12] propose that during vote casting the voter encrypts the index of the VR where his credential is stored. During the tallying phase the index will be decrypted, and the encrypted credential will be retrieved from the initial version of the voting roll. Now a single PET suffices to check the two credentials and decide if the vote should be counted or not. For this scheme to work the system must make sure that a uniform distribution of fake indices appears. To this

end it is proposed that the tallying authorities themselves append fake votes. This has also the effect that a coercer that monitors the bulletin board *cannot tell whether a target voter made use of his moment of privacy.* In our proposed protocol we borrow many ideas from [Spy+12].

CIVITAS In [CCM08], the first detailed implementation of [JCJ05] is provided. First of all, a concrete specification of the credential generation and distribution is given, and the faking mechanism is discussed. Our proposal in section 5.2 supports both mechanisms, so the details are given there. The general idea is that the members of the RA generate the credential in a distributed manner and provide the voter with designated verifier proofs. In order to fake the credential, a coerced voter can simply select a fake credential share $\hat{s_j}$, which is assumed belongs to a teller not controlled by $\mathcal{A}$. To solve the scalability problem, Civitas employs parallelism. The voters are partitioned in *virtual precincts*, called blocks. Vote authorization and tallying happens independently in each block, and the results are aggregated. As a result, the complexity of the scheme is $\mathcal{O}\left(Bn_{Bmax}^2\right) + \mathcal{O}(Bn_{Bmax}n_{Bmin})$ where n_{Bmax} is the maximum number of votes per block, B is the number of blocks and n_{Bmin} is the minimum number of voters per block.

Anonymity Sets Another option to make tallying more efficient in the JCJ scheme, rests on the voter posting except for her credential a set of other credentials selected randomly from the voter roll. These credentials, function as an *anonymity* set to hide its real credential and to constrain the checks.

In [CH11] the voter presents the real credential accompanied by $\eta - 1$ credentials from the voter roll. More specifically in Selections:

- The voter encrypts the credential θ using exponential ElGamal. This is done to enable rerandomization for use in multiple elections and more importantly to remove fake votes.
- During vote casting the voter commits to g^θ and rerandomizes her entry from the voter roll. Also, she randomly picks $\eta - 1$ encrypted credentials from the voter roll and embeds them into the ballot. Along with the standard [JCJ05] proofs she proves that her credential is indeed a rerandomization of one of the η credentials, without of course revealing which one (ala [CDS94]).
- During the vote authorization phase the deduplication process occurs in linear time as the same value g^θ is present in all of them. The last vote per g^θ is kept.
- For fake credential removal, the commitment is treated as a deterministic encryption of the credential. As such it is randomized from the standard [JCJ05]

mixnet and is 'PETted' with the rerandomized voter roll entry. A real vote is determined by a successful such test.

A practical aspect of the coercion resistance schemes that were presented above, relates to how does the voter creates fake credentials when under coercion. This issue is quite vague in the literature reviewed. The [CH11] scheme tackles this problem using the notion of *panic passwords*. During registration, the voter selects a password to be used during vote casting. In reality, the system partitions the space of possible passwords into three categories:

- The actual password to authenticate the voter and submit the real credential.
- The panic passwords which indicate that the user is under coercion and generate fake credentials. The interaction however is indistinguishable from the one that takes place with the actual password.
- The inadmissible passwords that indicate authentication failure and do not allow vote casting.

Ideally panic passwords should be a sparse subset of inadmissible passwords and the actual password is a pre-selected panic password. To illustrate the concept, better, we refer to the *5-Dictionary* introduced in [CH08] and used in Selections. The password space consists of any combination of five words. The valid passwords are any combination of five words from an agreed-upon dictionary. A particular combination is selected as the actual password during registration. Any other combination from the dictionary is a panic password. Any other five-word combination from the password space is invalid.

A similar protocol is proposed in [Sch+11] that can be seen as a combination of Selections and [Spy+12]. The voter instead of posting only the index of the VR where the claimed credential is stored, it posts a set of $\eta - 1$ more indices in cleartext, where $\eta \leftarrow_{\$} [n-1]$. As a result, the ballot posted from the voter has the form $\mathsf{b} = (\mathsf{Enc}_{\mathsf{pk}_{\mathsf{TA}}}(v_i), \mathsf{Enc}_{\mathsf{pk}_{\mathsf{TA}}}(\theta_i), I)$ where $I \subset \mathsf{VR}$ AND $i \in I$. The EA posts replicate the ballot for all $j \in I$. As a result, for V_i the set $\{\mathsf{Enc}_{\mathsf{pk}_{\mathsf{TA}}}(\mathtt{vt}_i), \mathsf{Enc}_{\mathsf{pk}_{\mathsf{TA}}}(\theta_i), \mathsf{VR}_j\}$. In order for the vote to be authorized a PET is performed between and $\mathsf{Enc}_{\mathsf{pk}_{\mathsf{TA}}}(\theta_i)$ and VR_j. The advantage of this approach against Selections is that no zero knowledge proofs are required for the credentials.

Structured credentials A different approach to the duplicate and fake credential detection, tries to avoid the blind comparisons inherent in all the proposals so far, and results in a new line of research [AFT07; Ara+10; AT13], the defining characteristic of which is that structure is added to the credentials. As a result, a credential is not merely a group element or an alphanumeric string, but a tuple of elements some items

of which are used for the identification of voters, and other for duplicate detection and invalid credential removal.

For instance, in [AFT07], the credential for V_i is a tuple (r_i, a_i, b_i, c_i) where r_i is a random index, a_i is a random group element, $b_i = a_i^{\mathsf{sk}_1}$ and $c_i = a_i^{\mathsf{sk}_{1\mathsf{RA}} + r_i \mathsf{sk}_{1\mathsf{RA}} \mathsf{sk}_{2\mathsf{RA}}}$ where $\mathsf{sk}_{1\mathsf{RA}}, \mathsf{sk}_{2\mathsf{RA}}$ are secret keys of the RA. (a_i, b_i, c_i) can be made public, while r_i must be kept secret. This tuple is created during registration for each voter and is transmitted to the voter according to the assumptions set by the [JCJ05] framework (untappable channel, trust in a particular member of the RA). The fakekey functionality is made possible by the following observation: if (r_i, a_i, b_i, c_i) is a valid credential then $(r_i, a_i^{\psi}, b_i^{\psi}, c_i^{\psi})$ is also a valid credential. So, the voter can pick a random index and produce a new credential by raising the known credential to a random power. The same observation allows for a single registration phase to be reused for many elections.

The ballot in [AFT07] consists of an encryption of the vote $\mathtt{vt}$ and the elements of the tuple using the public key of the TA:

$$\mathsf{b}_i = (\mathsf{Enc}_{\mathsf{pk}_{\mathsf{TA}}}(\mathtt{vt}_i), a_i, \mathsf{Enc}_{\mathsf{pk}_{\mathsf{TA}}}({a_i}^{r_i}), \mathsf{Enc}_{\mathsf{pk}_{\mathsf{TA}}}({b_i}^{r_i}), \mathsf{Enc}_{\mathsf{pk}_{\mathsf{TA}}}(c_i), o^{r_i}, \pi)$$

where o is a random group element and π is a set of proofs of ballot well-formedness. The value o^{r_i} is used for duplicate detection and removal through a hashtable. The casting phase requires an anonymous channel. Counting is performed jointly by the registration and tallying authorities. RA checks that the credentials are unique and valid using its secret keys, and TA decrypts and counts the corresponding valid votes. The same approach is used in a follow-up work [Ara+10], with the difference that the credentials are shorter and different security assumptions is used for the security of the scheme.

A major possible weakness in both cases is that the existence of o^{r_i} could be used as a receipt for vote selling or to break coercion resistance. Indeed, if the ballot (that contains the value o^{r_i}) is posted to the BB as is then the coercer could demand that the voter present an r_i such that an exponentiation could yield the posted value. However, in [AFT07] it is not exactly specified which parts of the ballot are stored in the BB. Another weakness of this line of work is that the authorities can easily generate fake credentials that are identical to valid ones and insert ballots. This weakness is addressed in [AT13].

Board Flooding The [JCJ05] framework has another problem. The BB contains many fake votes that will not be counted, but are used only to fool the coercer. This

design choice could yield another attack vector as first pointed in [KHF11]: An adversary could disrupt an election by causing a denial of service attack, through deliberately injecting fake votes. In the original version this has a quadratic effect, however in the linear version this is not so important. A solution first proposed in [KHF11] combines *dummy* credentials together with a smart BB. The former replace JCJ fake credentials and are the mechanism to evade coercion. They are given to the voter during registration. In particular each V_i is granted a (different) number d_i of fake credentials. As a result, not every group element is a potential credential but only the ones that are received during registration. This implies that everything else is rejected from the BB automatically upon submission. As a result, there will be a bound on the items present in the BB, which consists only of votes accompanies with real and dummy credentials. The downside of this proposal is that a portion of the voters (the ones that receive the minimum number credentials) are more vulnerable to coercion as the adversary will demand that they provide exactly so many and they will not be able to present fake ones.

Our proposal (chapter 5) combines aspects of all these variations of [JCJ05]. To formally prove coercion resistance we adapt the model and game-based definitions of the JCJ framework. While other more rigorous models do exist, we find the JCJ framework the best compromise between theory and practice. To illustrate the reason behind our preference we compare our model with the notion of universally composable incoercibility [Alw+15]. The latter applies not only to voting schemes but to any protocol for secure multi-party computation. A complete analysis of [Alw+15] is beyond the scope of this thesis, as it assumes knowledge of the *universal composability (UC)* framework of [Can01]. This framework aims to guarantee that the security guarantees of a protocol are maintained when they are combined with other (insecure) protocols. This guarantee does not apply to [JCJ05] and by extension to our work.

Nevertheless, both works have some things in common: They both assume that there is some 'space of doubt' or coercer uncertainty, so that the election tally does not leak if the voter submitted to the attack. Both assume local coercion and corruption, in the sense that the voters do not need to know who else is corrupted or coerced in order to fool both adversaries. The latter can communicate through the UC environment or the game adversary. They also both assume a trusted hardware token, at least in principle since later instantiations of JCJ [UH12] provide mental coercion resistance, as well.

The UC incoercibility definition expects the security properties to emanate solely from the cryptographic primitive used in the protocol. Intuitively this means that the coercer will be present when the voter tries to cheat. This is a very strong security property. On the other hand the JCJ framework 'cheats' by using *out-of-band*

capabilities: revoting and a moment of privacy (since when the voter is before the coercer she will simply do as she is told) As a result, the voter is not confined only to cryptographic countermeasures but can also use real-world means. This makes the resulting protocols more practical, also leads to less rigorous definitions and analysis. On the flip side, insofar as our understanding of [Alw+15] permits, the UC incoercibility definition cannot deal with the forced abstention attack as even in the case of active coercion, the adversary does not seem to block communications. The JCJ framework deals with this attack at the expense of an anonymous channel.

4.5 Ballot secrecy

Voter privacy is a tricky property to formulate. As we saw in section 1.2 there are many levels of privacy protection. None of them are absolute, as the result, combined with the tallying rules, leaks information. For instance:

- In a unanimous result, everyone knows how everyone voted.
- If all voters except for one, have voted for the same choice, then the one that differs knows how everyone else voted.
- In the general case, every voter knows the probability that a random voter picked a specific candidate from the percentage of the total votes this candidate received.

As a result, a definition for all types of privacy is limited by what is leaked by the election result. However, this is independent of the voting system used and not limited to electronic voting systems. We are interested in the effect of the electronic voting system, in particular, the cryptographic protocol used, on vote privacy. In the current section, we begin our discourse on election privacy, starting with ballot secrecy. In the following section, we will deal with stronger notions of privacy such as receipt-freeness and coercion resistance.

According to [CS13] ballot secrecy concerns the protection of a ballot's contents from a *passive* adversary, i.e. an adversary that simply monitors but does not try to affect a target voter or the voting protocol.

4.5.1 Trust assumptions

The adversary is as we saw passive, in its interactions with the target voter. However, he can corrupt other voters adaptively and use them by dictating their behavior in order to learn how the target voter voted. The talliers are considered honest for privacy, but the adversary can partly corrupt them. This corruption is extremely important

in the case of homomorphic voting systems such as Helios, where the talliers must decrypt the vote output. As only a minority of corrupted talliers is tolerated, a trust assumption is that the adversary should not corrupt beyond this threshold. However, it is an open problem to study how this is perceived by voters, as some might object to the fact that they must trust the talliers for their vote not to be revealed.

4.5.2 Security games for ballot secrecy

We use the BPRIV definition of [Ber+15] to model privacy for our proposed voting scheme in chapter 5. This definition combines the best elements of the literature (up to that point) and has been used to model the privacy of the Helios voting system.

First approaches to ballot secrecy

Before, we describe the BPRIV definition, we summarize the characteristics of other definitions up to BPRIV by following the excellent review found in [Ber+15]. The most common approach to modeling ballot secrecy, adapts the indistinguishability games that express cryptographic secrecy in the setting of voting systems. The adversary, instead of distinguishing the encryption of two messages, tries to distinguish between two BBs ($\mathsf{BB}_0, \mathsf{BB}_1$) that contain different variations of voting scenarios. The general description of the indistinguishability games is the following:

- The challenger setups the voting protocol. Both $\mathsf{BB}_0, \mathsf{BB}_1$ are initially empty.
- The adversary can cast as many ballots as he wishes, representing the corrupted voters. These ballots are posted to both BBs.
- The challenger posts a different variation of some behavior in each BB, by presenting two voting choices to an honest voter. The honest voter executes the voting/casting protocol and casts a ballot for each choice in each BB.
- The adversary may continue to post items to both BBs.
- The challenger flips a coin and presents the respective BB.
- The adversary must guess which BB he is viewing.
- Ballot secrecy holds if $\mathcal{A}$ cannot distinguish which BB he is viewing except with negligible probability.

In some first privacy definitions [BT94], the variation in behavior for the challenger is to post different permutations of votes for two honest voters, i.e. $\{(\mathsf{V}_0, \mathtt{vt}_0), (\mathsf{V}_1, \mathtt{vt}_1)\} \in \mathsf{BB}_0$ and $\{(\mathsf{V}_0, \mathtt{vt}_1), (\mathsf{V}_1, \mathtt{vt}_0)\} \in \mathsf{BB}_1$. A variation of this definition [Ben87] considers the fact that the tally must remain the same for the honest voters. For this reason, it employs the result function that we saw in the respective verifiability definitions.

The challenger, except for $\mathsf{BB}_0, \mathsf{BB}_1$ maintains two lists $\mathcal{L}_0^{\mathtt{Hon}}, \mathcal{L}_1^{\mathtt{Hon}}$ where the votes of the honest voters are kept. The adversary casts ballots to both BB. For the challenge, $\mathcal{C}$ posts a vote $\mathtt{vt}_0$ in $\mathsf{BB}_0, \mathcal{L}_0^{\mathtt{Hon}}$ and a different $\mathtt{vt}_1$ in $\mathsf{BB}_1, \mathcal{L}_1^{\mathtt{Hon}}$. If the results for the honest voters in $\mathcal{L}_0^{\mathtt{Hon}}, \mathcal{L}_1^{\mathtt{Hon}}$ are different, the experiment aborts. The problem with these definitions is that in certain cases and tallying rules, both tallies for the honest voters are equal, but the adversary can tell the BBs apart by taking advantage of the voters he controls.

To overcome this limitation, [Ber+11] propose a variation of this experiment. The first bulletin board, BB_0 always contains the votes of the honest voters. The second BB_1, replaces the honest votes with null, fake votes. Counting takes place always on BB_0. When the adversary requests to see the tally, he is presented with the tally and one $\mathsf{BB}_0, \mathsf{BB}_1$. He must guess which board he has been given.

This definition has a problem again that illustrates the conflict of ballot secrecy and verifiability. If $\mathcal{A}$ can examine proofs of correct ballot formation and tallying, he can immediately reject the ballot box with the fake votes, as these proofs will not validate. This situation is remedied by the variation presented in [BPW12]. When the adversary is presented with the real BB the real tally and actual proofs are given to him. When the adversary is presented with the fake BB the real tally is given to him but the proofs are simulated to match the contents of the presented BB. Unfortunately, this definition is not sound as it can characterize as private, protocols that are not [Ber+15]. The definitions of [Ben87] and [BPW12] can be combined so that when the tallies for the honest voters are equal then the real tally is given to $\mathcal{A}$. If not, then $\mathcal{A}$ is presented with one BB.

Ballot secrecy with trusted talliers - BPRIV

We now turn to BPRIV, which is an indistinguishability game as well. The adversary must distinguish between two ballot boxes: BB_0 contains the votes of the honest voters as well as the ballots posted by $\mathcal{A}$ and BB_1 that contains fake ballots that will not be counted. The result is always computed from BB_0, but when the adversary is viewing BB_1 the proofs must be simulated from the data available in BB_1 for the tally of BB_0. The BPRIV definition is displayed in Algorithm 4.10, where the adversary has access to 3 oracles. The oracle **Vote** represents the honest voters - the adversary specifies two options $\mathtt{vt}_0, \mathtt{vt}_1$ and the challenger casts them to the respective BB. The oracle **Cast** allows the adversary to cast any ballot to both BBs. The oracle **Tally** performs the tally according to the described rules.

Algorithm 4.10: $\mathsf{BPRIV}^{\mathfrak{b}}_{\mathcal{A},\mathsf{VS}}$ from [Ber+15]

Input : security parameter λ
Output: $\{0,1\}$

Oracle Vote$(i, \mathtt{vt}_0, \mathtt{vt}_1)$
- $\mathsf{b}_0 := \mathsf{Vote}(i, \mathtt{vt}_0)$
- $\mathsf{b}_1 := \mathsf{Vote}(i, \mathtt{vt}_1)$
- **if** $\mathsf{Valid}(\mathsf{b}_0, \mathsf{BB}_0)$ AND $\mathsf{Valid}(\mathsf{b}_1, \mathsf{BB}_1)$ **then**
  - $\mathsf{BB}_0 \Leftarrow \mathsf{b}_0$
  - $\mathsf{BB}_1 \Leftarrow \mathsf{b}_1$
- **else**
  - return $\perp$

Oracle Cast(i, b)
- **if** $\mathsf{Valid}(\mathsf{b}, \mathsf{BB}_{\mathfrak{b}})$ **then**
  - $\mathsf{BB}_0 \Leftarrow \mathsf{b}$
  - $\mathsf{BB}_1 \Leftarrow \mathsf{b}$
- **else**
  - return $\perp$

Oracle Tally$(\mathfrak{b})$
- **if** $\mathfrak{b} = 0$ **then**
  - $(\mathtt{T}, \pi_{\mathtt{T}}) := \mathsf{Tally}(\mathsf{sk}_{\mathsf{TA}}, \mathsf{prms}, \mathsf{CS}, \mathsf{BB}_{\mathfrak{b}})$
  - return $(\mathtt{T}, \pi_{\mathtt{T}})$
- **else**
  - $(\mathtt{T}, \pi_{\mathtt{T}}) := \mathsf{Tally}(\mathsf{sk}_{\mathsf{TA}}, \mathsf{prms}, \mathsf{CS}, \mathsf{BB}_0)$
  - $\pi'_{\mathtt{T}} := \mathsf{Sim}(\mathsf{sk}_{\mathsf{TA}}, \mathsf{prms}, \mathsf{CS}, \mathsf{BB}_0, \mathsf{BB}_1, \mathtt{T})$
  - return $(\mathtt{T}, \pi'_{\mathtt{T}})$

$(\mathsf{prms}, \mathsf{pk}_{\mathsf{TA}}, \mathsf{sk}_{\mathsf{TA}}) \leftarrow \mathsf{VS.Setup}(1^{\lambda})$
$\mathsf{CS} \leftarrow \mathcal{A}()$
$\mathfrak{b}' \leftarrow \mathcal{A}^{\mathbf{Vote},\mathbf{Cast},\mathbf{Tally}}(\mathsf{prms}, \mathsf{pk})$
return $\mathfrak{b} = \mathfrak{b}'$

Definition 4.7: BPRIV ballot secrecy

A voting scheme VS is private according to BPRIV if for every PPT algorithm $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\lambda)$ and an efficient algorithm Sim such that:

$$\Pr\left[\mathsf{BPRIV}^{0}_{\mathcal{A},\mathsf{VS}}(\lambda) = 1\right] - \Pr\left[\mathsf{BPRIV}^{1}_{\mathcal{A},\mathsf{VS}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$$

The BPRIV definition deals with leakage from the contents of the BB and the proof of correct tally. However, the operations of the tally phase itself might also leak enough data for the adversary to be able to distinguish between the BBs, e.g. from the filtering of the duplicate ballots found in voting schemes that allow revoting for coercion resistance. To deal with this problem, [Ber+15] accompany the ballot privacy definition with two extra properties, that apply to voting schemes that allow for revoting in particular:

- Strong consistency: The result of the election T as obtained by the application of the tally phase is the same as the result of the election obtained directly from decrypting the ballots. As a result the tally algorithm does not leak anything.
- Strong correctness: Ballots that originate from honest voters are accepted regardless of the contents of the BB at the time they are posted.

More formally:

Definition 4.8: BPRIV strong consistency

A voting scheme VS has strong consistency relative to a result function result if there exist:

- An extraction algorithm Extract that receives the election secret key and a ballot and outputs the identity of the voter that cast it and the ballot contents. More formally:

$$\forall (i, \mathtt{vt}) \in \mathsf{V} \times \mathsf{CS} : \Pr\left[\mathsf{Extract}(\mathsf{sk}_{\mathsf{TA}}, \mathsf{VS.Vote}(i, \mathtt{vt})) = (i, \mathtt{vt})\right] = 1 - \mathsf{negl}(\lambda)$$

- An independent ballot validation algorithm ValidInd that receives the tallying public key and a ballot and outputs if the ballot is valid, such that

$$\forall (\mathsf{BB}, \mathsf{b}) \leftarrow \mathcal{A}() : \mathsf{VS.Valid}(\mathsf{BB}, \mathsf{b}) = 1 \Rightarrow \mathsf{ValidInd}(\mathsf{pk}_{\mathsf{TA}}, \mathsf{b}) = 1$$

for which:

$$\Pr\left[\left\{\begin{array}{l}(\mathsf{pk}_{\mathsf{TA}}, \mathsf{sk}_{\mathsf{TA}}) = \mathsf{VS.Setup}(\lambda) \\ \mathsf{BB} \leftarrow \mathcal{A}() \\ (\mathsf{T}, \pi_{\mathsf{T}}) \leftarrow \mathsf{VS.Tally}(\mathsf{sk}_{\mathsf{TA}}, \mathsf{BB}) \\ \text{return } (\mathsf{T} = \mathsf{result}(\{(i, \mathtt{vt}_i)\}_{i=1}^{n}))\end{array}\right\} = 1\right] = 1 - \mathsf{negl}(\lambda)$$

where $\mathsf{BB} = \{\mathsf{b}_i : \mathsf{ValidInd}(\mathsf{b}_i) = 1\}_{i \in \mathsf{V}}$

Definition 4.9: BPRIV strong correctness

A voting scheme VS satisfies strong correctness if

$$\Pr\left[\left\{\begin{array}{l}(\mathsf{BB}, i, \mathtt{vt}_i) \leftarrow \mathcal{A}() \\ \mathsf{b}_i \leftarrow \mathsf{VS.Vote}(i, \mathtt{vt}_i) \\ \text{return } \mathsf{VS.Valid}(\mathsf{BB}, \mathsf{b}_i)\end{array}\right\} = 0\right] = 1 - \mathsf{negl}(\lambda)$$

Ballot secrecy with untrusted talliers and anonymous channels

We now present a variation of BPRIV to express ballot secrecy in voting schemes that do not require trust in talliers for ballot secrecy and use anonymous channels. This variation is named U-BPRIV and is presented in Algorithm 4.11. Its design follows the rationale of BPRIV of [Ber+15] as expanded in [Cha+16] for voting schemes with registration (cf. Algorithm 4.9). The elections are recorded into two bulletin boards $\mathsf{BB}_0, \mathsf{BB}_1$, and the objective of the adversary, denoted by $\mathcal{A}$ and computationally bounded for this model, is to distinguish them. $\mathcal{A}$ can actively participate in

the elections, corrupt voters ($\mathsf{V}_{\mathtt{Corr}}$) and collect all data generated by VS and honest voters ($\mathsf{V}_{\mathtt{Hon}}$). The honest RA is modeled as a call to the **Register** oracle. Honest and adversarial vote casting is represented by the **Vote**, **Cast** oracles respectively. All the oracle calls execute the respective functionalities and return their outputs along with the protocol transcript Trans and leaked data Aux (e.g. communication addresses and timing information). These form the view of the adversary, denoted as $\mathsf{view}_{\mathcal{A}}$. Tallying is performed by $\mathcal{A}$ and therefore there is no respective oracle as in BPRIV. In the end, $\mathcal{A}$ is presented with one of the two bulletin' boards $\mathsf{BB}_{\mathfrak{b}}$ and their objective is to distinguish which one they are seeing.

In more detail, the challenger $\mathcal{C}$ takes the role of the RA, the BB, and the honest voters. Initially, it executes a Setup functionality to create the registration parameters. The adversary generates the voter roll, the candidate slate CS, and the tallying parameters. The voters complete the registration process and receive the credentials in physical or electronic form. They are not restricted only to public and private key pairs but they can also be encrypted group elements as in [JCJ05]. This process is denoted with the call to the **Register** oracle. The communication with the selected BB (according to $\mathfrak{b}$) has a transcript $\mathsf{Trans}_{\mathfrak{b}}$ and leaks some information denoted as $\mathsf{Aux}_{\mathfrak{b},\mathsf{Register}}$. This, along with the public result of Register is provided to the $\mathcal{A}$ via its view.

The core of the game is the ballot casting phase, which is represented by the **Vote** and **Cast** for honest and corrupted voters respectively. If V_i is corrupted, then $\mathcal{C}$ hands the private credentials sk_i giving full control to $\mathcal{A}$.

The challenger retains control of the honest voters. The adversary schedules concurrent executions of the **Vote** and **Cast** functionalities for all voters, in the most *favorable manner* to them. If a voter is honest, then $\mathcal{C}$ plays her role, receives 2 selections $\mathtt{vt}_0, \mathtt{vt}_1 \in \mathsf{CS}$ picked by $\mathcal{A}$ and provides in return the results of **Vote**, namely a ballot b, proofs of validity and leaked data due to the use of the communication channels. The challenger flips a random coin to decide which BB to use to cast their vote. This models the anonymous channel and prevents $\mathcal{A}$ from winning trivially. In all cases, the view of the adversary as well as the auxiliary information is updated after the execution of a functionality. All ballots are checked for validity for their respective BB. When all voters have finished executions of their protocols, the adversary is presented with one of the bulletin boards and performs tallying on it. Finally, $\mathcal{A}$ tries to guess which board he was presented with.

Based on this game, we provide our variation of BPRIV privacy with untrusted talliers for secrecy and anonymous channel:

Algorithm 4.11: $\text{U-BPRIV}^{\mathfrak{b}}_{\mathcal{A},\mathsf{VS}}$ with anonymous casting

Input : security parameter λ, election information L, corruption tolerance t
Output: $\{0,1\}$
Oracle Register(i,L)
  if $i \in \mathsf{L}$ **then**
    $(\mathsf{sk}_i,\mathsf{pk}_i) \leftarrow \mathsf{VS.Register}(\mathsf{sk}_{\mathsf{RA}}),i)$
    $\mathsf{BB}_\beta \Leftarrow \mathsf{pk}_i \quad$ for $\beta \in \{0,1\}$
    return $(\mathsf{pk}_i, \mathsf{Trans}_{\beta,\mathsf{Register}}, \mathsf{Aux}_{\beta,\mathsf{Register}}) \quad$ for $\beta \in \{0,1\}$
  else
    return $\perp$
Oracle Vote$(i,\mathtt{vt}_0,\mathtt{vt}_1)$
  $\beta \leftarrow_\$ \{0,1\}$
  $\mathsf{b}_\beta := \mathsf{VS.Vote}(i,\mathtt{vt}_\beta)$
  $\mathsf{b}_{1-\beta} := \mathsf{VS.Vote}(i,\mathtt{vt}_{1-\beta})$
  if $\mathsf{VS.Valid}(\mathsf{b}_\beta,\mathsf{BB}_\beta)$ AND $\mathsf{VS.Valid}(\mathsf{b}_{1-\beta},\mathsf{BB}_{1-\beta})$ **then**
    $\mathsf{BB}_\beta \Leftarrow \mathsf{b}_\beta$
    $\mathsf{BB}_{1-\beta} \Leftarrow \mathsf{b}_{1-\beta}$
    return $(\mathsf{Trans}_{\beta,\mathsf{Vote}}, \mathsf{Aux}_{\beta,\mathsf{Vote}}) \quad$ for $\beta \in \{0,1\}$
  else
    return $\perp$
Oracle Cast(b)
  if $\mathsf{VS.Valid}(\mathsf{b},\mathsf{BB}_\beta) \quad$ *for* $\beta \in \{0,1\}$) **then**
    $\mathsf{BB}_\beta \Leftarrow \mathsf{b} \quad$ for $\beta \in \{0,1\}$
    return $(\mathsf{Trans}_{\beta,\mathsf{Cast}}, \mathsf{Aux}_{\beta,\mathsf{Cast}}) \quad$ for $\beta \in \{0,1\}$
  else
    return $\perp$
$(\mathsf{prms},\mathsf{pk}_{\mathsf{RA}},\mathsf{sk}_{\mathsf{RA}}) \leftarrow \mathsf{VS.Setup}(1^\lambda)$
$(\mathsf{VR},\mathsf{CS},\mathsf{pk}_{\mathsf{TA}},\mathsf{sk}_{\mathsf{TA}}) \leftarrow \mathcal{A}(1^\lambda)$
$\mathsf{V}_{\mathtt{Corr}} \leftarrow \mathcal{A}(\mathsf{L},\textbf{corrupt})$
if $|\mathsf{V}_{\mathtt{Corr}}| > t$ **then**
  return $\perp$
$\mathsf{V}_{\mathtt{Hon}} \leftarrow I \backslash \mathsf{V}_{\mathtt{Corr}}$
$\text{view}_{\mathfrak{b},\mathcal{A}} \leftarrow \mathcal{A}^{\textbf{Register},\textbf{Vote},\textbf{Cast}}(\mathsf{BB}_{\mathfrak{b}})$ where

$$\text{view}_{\mathfrak{b},\mathcal{A}} = (\mathsf{Trans}_{\mathfrak{b},\mathsf{Register}}, \mathsf{Aux}_{\mathfrak{b},\mathsf{Register}}, \mathsf{Trans}_{\mathfrak{b},\mathsf{Vote}}, \mathsf{Aux}_{\mathfrak{b},\mathsf{Vote}}, \mathsf{Trans}_{\mathfrak{b},\mathsf{Cast}}, \mathsf{Aux}_{\mathfrak{b},\mathsf{Cast}})$$

$\mathtt{T} \leftarrow \mathcal{A}(\mathsf{BB}_{\mathfrak{b}},\textbf{tally})$
$\mathfrak{b}' \leftarrow \mathcal{A}(\text{view}_{\mathfrak{b},\mathcal{A}},\textbf{guess})$
return $\mathfrak{b} = \mathfrak{b}'$

Definition 4.10: U-BPRIV ballot secrecy

A voting scheme VS is private if for every PPT algorithm $\mathcal{A}$ there exists a negligible function negl such that for every L:

$$\Pr\left[\text{U-BPRIV}^{0}_{\mathcal{A},\text{VS},t}(1^{\lambda},\text{L})=1\right]-\Pr\left[\text{U-BPRIV}^{1}_{\mathcal{A},\text{VS},t}(1^{\lambda},\text{L})=1\right]\leq \text{negl}(\lambda)$$

To model voting schemes that allow revoting with U-BPRIV we also define strong consistency and strong correctness as in Definition 4.8 and Definition 4.9. For brevity, we do not repeat them here.

4.5.3 Privacy based on blind signatures

The approach towards privacy in Helios-related schemes is based on trust. The TA will decrypt nothing but the aggregate of the votes in the homomorphic version and only the individual votes after anonymization in the mixnet based version. In practice, this trust assumption is enforced only using threshold cryptosystems, where the power to decrypt via the secret key is split to many shareholders assuming to have conflicting interests.

Another line of work towards privacy-oriented voting schemes, utilizes the unlinkability property of blind signatures. In fact, voting was a proposed application from their inception [Cha83]. Indeed, the scheme proposed there combines privacy and individual verifiability (using modern terms). It utilized a registration authority RA to handles eligibility checks and requires and anonymous channel: The general workflow is as follows [1]:

- V_i blinds the ballot and submits it along with election identifying information to the RA.
- The RA validates the voter data, and if V_i has the right to vote, signs the blinded ballot and returns it.
- V_i validates the signature of the RA, unblinds the signatures and posts the signed ballot *anonymously*.
- The RA receives the signed ballots, validates the RA's signature, and posts them to BB for verification.
- Each voter can individually verify her ballot through a random pattern embedded to it, known only to him.

[1] A large part of the material in subsection 4.5.3 is based on [Gro14]

The unlinkability of blind signatures, prohibits the RA to link signing with verification sessions, which intuitively provides privacy to the scheme as the RA cannot use the voter identification information obtained during the signing request to identify who cast a particular vote. Individual verifiability is provided by pinpointing a vote in the BB by using the random pattern embedded into it. The problem with [Cha83] is twofold. Firstly, it lacks fairness as the RA knows the intermediate voting results. Also, in the case of a dispute, the voter must show their vote, thus defeating privacy. While dispute-resolution or accountability is an exceedingly difficult property to achieve, a solution towards this direction was proposed in [FOO92]. The voting scheme described there is based on the separation of functions between 2 entities.

– The RA, that knows the voter's identity but not the actual vote. As a result, it can efficiently check eligibility and authorize the vote.

– The TA, that knows the vote contents but not the voter identity. As a result, it can provide the counting. In fact, its role is not essential, as the counting can be performed by any interested party.

The workflow is depicted in Figure 4.1 and detailed as following:

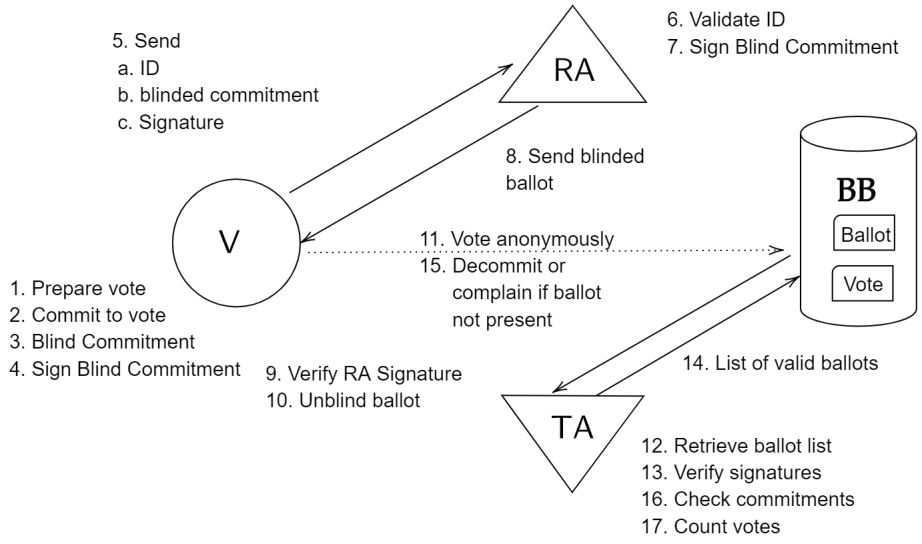


FIGURE 4.1: Voting with blind signatures [FOO92]

– Setup: An RSA digital signature scheme is initialized by executing $\mathcal{DS}.\mathsf{KGen}$. The RA obtains a key-pair $(\mathsf{sk}_{\mathsf{RA}}, \mathsf{pk}_{\mathsf{RA}})$, while no key is required for the TA.

– Register: Each V_i obtains a similar key pair $(\mathsf{sk}_i, \mathsf{pk}_i)$.

– Vote: Let $\mathtt{vt}_i$ be the choice of V_i.

  * The ballot is created by committing to $\mathtt{vt}_i$ using randomness r_i, $\mathsf{b}_i = \mathsf{Commit}(\mathtt{vt}_i, r_i)$. The commitment scheme must be binding and ensures that the voter cannot

behave differently in the preparation and generation phases. Moreover, it substitutes the random pattern of [Cha83] that the voter must embed into her vote in order to verify it in the BB.

* The ballot is blinded, using the properties of the RSA scheme: $\mathsf{b}'_i = \mathsf{Blind}(b_i, r'_i)$
* The blinded ballot is signed using the private key of the voter, producing a signature $\overline{\sigma}_{\mathsf{V}_i}$.
* The voter submits $(i, \mathsf{b}'_i, \overline{\sigma}_{\mathsf{V}_i})$ to the RA, where i denotes voter identifying information.
* Upon receipt, the RA validates $\overline{\sigma}_{\mathsf{V}_i}$ using pk_i, obtained through the identity i, the eligibility of V_i and checks for double requests to defend against double voting.
* If all checks turn out ok, it signs the blinded ballot producing $\overline{\beta}_i$.
* The RA sends $\overline{\beta}_i$ to V_i.
* After all voters have submitted their votes the RA announces the total number n of eligible voters.

– Cast: The voter executes Unblind and obtains the plain signature $\overline{\sigma}_i$ of the RA. The signature can be validated with the public key of the authority $\mathsf{pk}_{\mathsf{RA}}$. The tuple $(\mathsf{b}_i, \overline{\sigma}_i)$ is submitted to the TA through an anonymous channel, to hide the network identity of the voter from the TA.

– VerifyBallot The voter checks that the commitment is present in the BB.

– Tally:

* The TA validates $\{\overline{\sigma}_i\}_{i=1}^n$ using $\mathsf{pk}_{\mathsf{RA}}$. Note that if the TA cheats and fails to validate a signature, the affected voter can present the ballot with the received signature as submitted $(\mathsf{b}_i, \overline{\sigma}_i)$ and prove that her vote should be counted without revealing it. All valid ballots are indexed and publicly announced.
* Each voter opens the commitments sent through during the voting phase, by posting the index and the opening values r_i.

– Verify Everybody can verify the result, by computing it on their own.

Many aspects of the security of the scheme depend on the properties provided by the anonymous channel. Note that the privacy provided from [FOO92] is of a different nature, as it bears no trust assumptions.

If a perfectly anonymous channel is used, then the system provides *everlasting privacy* (cf. section 4.6). While dispute freeness is provided against a corrupted TA, it is not provided against corrupt voters leaving the system open to Denial-of-Service attacks, as a corrupt voter might send an invalid opening value and used it to object later or to cancel the vote of another participant.

The scheme [FOO92] provides individual verifiability if the commitments posted are binding. It does not provide strong universal verifiability [Cor+14] as a corrupted RA can stuff ballots. But it provides weak universal verifiability assuming the commitment scheme used is binding.

[FOO92] Variations An important drawback of [FOO92] is that it requires voter interaction in at least three stages, namely during authorization, voting and opening. The last of these is the most problematic, as it requires from a voter to wait until everybody else has cast their votes. This provides inefficiency especially if we contrast it to the *vote-and-go* approach of [CGS97]. On the other hand, it is conceptually simple, efficient and supports many counting functions. As a result, many improvements have been proposed.

In [Ohk+99], the authors reduce the number of voter interactions by one step. They achieve this by replacing the commitment scheme with a threshold encryption scheme. More specifically the TA generates a key pair, where the private part is split. Instead of committing to her choice, the voter encrypts with the public key of the TA. The encrypted vote is then blind-signed by the RA and sent with the signature to the BB during the voting phase. This is the last step of the voter interaction, assuming nothing goes wrong. After everybody has voted, the voter can simply check the BB for her encrypted vote and object if it is not found by revealing the authority signature. Instead of the opening phase, the counters collectively decrypt each vote and write the result to the bulletin board. Subsequently, the votes are aggregated. In order to protect [Ohk+99] fairness and verifiability attacks, that are now commonplace, but were not well researched when it was published, the voter must provide a proof π_{Enc} of correct encryption of her choice and the talliers must provide proof of correct decryption π_{Dec} when producing the result. We assume that the proofs use the strong Fiat-Shamir transform (cf. section 2.4.1).

The augmented [Ohk+99] scheme is presented in Figure 4.2, Figure 4.3,Figure 4.4.

The proposed voting scheme of this thesis extends the scheme in [Ohk+99] for coercion resistance (cf. chapter 3).

We now prove the following, which does not exist in the literature, as far as we are aware :

$\mathsf{AugFOO.Setup}(1^\lambda)$	$\mathsf{AugFOO.Register}(1^\lambda, i)$
The RA invokes	Each V_i
$\mathsf{prms}_{\mathcal{BS}} = \mathcal{BS}.\mathsf{Setup}(1^\lambda)$	$(\mathsf{sk}_i, \mathsf{vk}_i) \leftarrow \mathcal{DS}.\mathsf{KGen}(1^\lambda)$
$(\mathsf{sk}_{\mathsf{RA}}, \mathsf{vk}_{\mathsf{RA}}) \leftarrow \mathcal{BS}.\mathsf{KGen}(1^\lambda)$	$\mathsf{BB} \Leftarrow (i, \mathsf{vk}_i)$
$\mathsf{prms}_{\mathcal{ES}} \leftarrow \mathcal{ES}.\mathsf{Setup}(1^\lambda)$	
$(\mathsf{sk}_{\mathsf{TA}}, \mathsf{pk}_{\mathsf{TA}}) \leftarrow \mathcal{ES}.\mathsf{KGen}(1^\lambda)$	
$\mathsf{prms}_{\mathcal{DS}} \leftarrow \mathcal{DS}.\mathsf{Setup}(1^\lambda)$	
$\mathsf{prms}_{\mathsf{NIZK}} \leftarrow \mathsf{NIZK.Setup}()$	

Figure 4.2: The functionalities Setup, Register of the AugFOO scheme

$\mathsf{AugFOO.Vote}\langle \mathsf{RA}(\mathsf{sk}_{\mathsf{RA}}), \mathsf{V}_i(\mathtt{vt}_i)\rangle$

V_i		RA
$\mathsf{b}_i \leftarrow \mathcal{ES}.\mathsf{Enc}(\mathsf{pk}_{\mathsf{TA}}, \mathtt{vt}_i)$		
$\pi_{\mathsf{V}_i} \leftarrow \mathsf{NIZK.Prove}(\mathtt{vt}_i \in \mathsf{CS})$		
$r_i \leftarrow\!\!\$\ \mathsf{prms}_{\mathcal{BS}}$		
$\mathsf{b}'_i \leftarrow \mathcal{BS}.\mathsf{Blind}(\mathsf{b}_i, r_i)$		
$\overline{\sigma}_i \leftarrow \mathcal{DS}.\mathsf{Sign}(\mathsf{sk}_i, \mathsf{b}'_i)$	send over $(\mathrm{i}, \mathsf{b}'_i, \overline{\sigma}_i)$ $\longrightarrow$	
		Check signature $\overline{\sigma}_i$
		Check eligibility for V_i
		$\overline{\beta}_{i,\mathsf{RA}} \leftarrow \mathcal{BS}.\mathsf{Sign}(\mathsf{sk}_{\mathsf{RA}}, \mathsf{b}'_i)$
	$\longleftarrow$ send over $\overline{\beta}_{i,\mathsf{RA}}$	
$(\mathsf{b}_i, \overline{\sigma}_{i,\mathsf{RA}}) = \mathcal{BS}.\mathsf{Unblind}(\overline{\beta}_{i,\mathsf{RA}})$		
$\mathsf{BB} \leftsquigarrow (\mathsf{b}_i, \pi_{\mathsf{V}_i}, \overline{\sigma}_{i,\mathsf{RA}})$		

Figure 4.3: The protocol Vote of the AugFOO scheme

$\mathsf{AugFOO.Valid}(\mathsf{BB}, \mathsf{b})$	$\mathsf{AugFOO.Tally}(\mathsf{sk}_{\mathsf{TA}})$
if $\mathsf{b} \notin \mathsf{BB}$	The TA
and $\mathsf{NIZK.Verify}(\pi_{\mathsf{V}_i}) = 1$	**for** $i = 1$ **to** n **do**
and $\mathcal{BS}.\mathsf{Verify}(\overline{\sigma}'_{i,\mathsf{RA}}) = 1$	**if** $\mathcal{BS}.\mathsf{Verify}(\overline{\sigma}_{i,\mathsf{RA}}) = 1$
then	**and** $\mathsf{NIZK.Verify}(\pi_{\mathsf{V}_i}) = 1$
return 1	**then**
else	$\mathtt{vt}_i = \mathcal{ES}.\mathsf{Dec}(\mathsf{sk}_{\mathsf{TA}}, \mathsf{b}_i)$
return 0	Apply counting rule
	end if
	end for

Figure 4.4: The functionalities Valid, Tally of the AugFOO scheme

Lemma 4.1: AugFOO is U-BPRIV

The AugFOO scheme is private according to U-BPRIV.

Proof. We define a sequence of games beginning from the adversary interacting with the challenger of U-BPRIV^0 and concluding with the adversary interacting with the challenger of U-BPRIV^1. The differences in each game are detected by the adversary with negligible probability.

For the proof, we also require the technical assumption, that all inputs to the **Vote** oracle are equal as multisets.

- $Game_0$ is the U-BPRIV^0 game. BB_0 is built through a succession of calls to oracles **Vote**, **Cast** from the challenger and calls **Cast** from the adversary. This means that, for each tuple $(\mathsf{b}_i, \pi_{\mathsf{V}_i}, \overline{\sigma}_{i,\mathsf{RA}})$ posted by the challenger, $\mathcal{C}$ can internally maintain the tuple $(i, \mathtt{vt}_0, \mathsf{b}_0, \pi_{\mathsf{V}_0}, \overline{\sigma}_0, \mathtt{vt}_1, \mathsf{b}_1, \pi_{\mathsf{V}_1}, \overline{\sigma}_1)$ where b_0 is the ballot for $\mathtt{vt}_0$ in BB_0 and b_1 is the ballot for $\mathtt{vt}_1$ in BB_1. Note that while $\mathcal{C}$ knows in which BB each of the ballots for $\mathtt{vt}_0, \mathtt{vt}_1$ ends up, $\mathcal{A}$ cannot track this information because of the random coin that simulates the anonymous casting and decides in which BB the ballot will end up. As a result, he cannot trivially win the game by instructing all ballots that end up in BB_0 to have a specific option $\mathtt{vt}'_0$ and all ballots that end up in BB_1 another $\mathtt{vt}'_1$.
- $\{Game_1^i\}_{i\in[n]}$. For each honest voter i, the challenger selects another honest voter i' such that $\mathtt{vt}_i = \mathtt{vt}_{i'}$. Then it replaces swaps $(\mathsf{b}_{i0}, \pi_{\mathsf{V}_{i0}}, \overline{\sigma}_{i0})$ in for request i in $\mathsf{BB}_\mathfrak{b}$ with $(\mathsf{b}_{i'1}, \pi_{\mathsf{V}_{i'1}}, \overline{\sigma}_{i'1})$ from or request i' $\mathsf{BB}_{1-\mathfrak{b}}$ according to the internal table of tuples it maintains from the calls to oracle **Vote**. Recall that if the tuple originates from a **Cast** call, then there is no change, as the adversary posts the same ballot in both boards. In the case where the tuple originates from a **Vote** call, the change is indistinguishable from the point of view of $\mathcal{A}$ since both ciphertexts hide the same votes and the tally does not change. Additionally, the proofs are swapped as well and the signature is created on the ciphertext.

It is easy to see that $Game_1^n$ is U-BPRIV^1. Since each game in the sequence is indistinguishable to the adversary, the initial and final games are also indistinguishable. As a result, the AugFOO scheme provides privacy according to U-BPRIV.

The AugFOO scheme also provides strong consistency and strong correctness. To argue about this we define the following functionalities:

$$\mathsf{Extract}(\mathsf{sk}_{\mathsf{TA}}, (\mathsf{b}_i, \pi_{\mathsf{V}_i}, \overline{\sigma}_{i,\mathsf{RA}})) =$$
$$\text{if } \mathcal{BS}.\mathsf{Verify}(\overline{\sigma}_{i,\mathsf{RA}}) \text{ AND } \mathsf{NIZK}.\mathsf{Verify}(\pi_{\mathsf{V}_i}) \text{ then } \mathcal{ES}.\mathsf{Dec}_{\mathsf{sk}_{\mathsf{TA}}}(\mathsf{b}_i) \text{ else } \perp$$

and

$$\mathsf{ValidInd}(\mathsf{b}_i, \overline{\sigma}_{i,\mathsf{RA}}) = (\mathcal{BS}.\mathsf{Verify}(\overline{\sigma}_{i,\mathsf{RA}}) = 1 \,\text{AND}\, \mathsf{NIZK}.\mathsf{Verify}(\pi_{\mathsf{V}_i}) = 1)$$

Regarding strong consistency, we note that the actual tally is computed by essentially applying the functionalities Extract and ValidInd, which is the same as AugFOO.Valid except for the duplicate check. Furthermore, the Extract functionality will recover the correct voter choice assuming a sound zero-knowledge proof system and correct encryption and signing schemes.

Strong correctness holds because even an adversarial BB cannot reject an honestly generated ballot. Since π_{V_i} is generated honestly and the RA is assumed honest, the only ways to invalidate a ballot is to create an exact duplicate of the vote in question. To do this, however, the adversary must guess the randomness used for encryption, which can occur with negligible probability.

■

4.6 Everlasting Privacy

As we saw in section 1.2 ballot secrecy allows voters to express their true preference without repercussions. In electronic elections, it is usually provided by cryptographic schemes, that rely on hardness assumptions. These, however, may be broken in the future. As a result, a computationally powerful, future, oppressive regime might obtain the vote contents and use them to better control their subjects. This constitutes an indirect coercion attempt [MN06] in the present and it is quite easy to achieve as secret ballots and election-related data are made widely available by e-voting schemes in order to provide verifiability [Ber+17]. Furthermore, since authoritarian regimes also control state and infrastructure agencies, their view will be not limited only to publicly available information but will also contain 'insider' data. Everlasting privacy, a term proposed by [MN06], is the property that protects voting protocols from such adversaries [2].

Before [MN06], there have been previous works that tackle the same problem, even if they do not exactly employ the term everlasting privacy. For instance, in [Cra+96] the voter uses the information-theoretically hiding Pedersen commitment scheme to commit to the vote. The openings are then secret shared to the authorities using private channels and homomorphically combined. To be verifiable, all exchanged data are stored in a bulletin board, modeled as a public broadcast channel with memory.

[2] Parts of this section appear in [GPZ20]

Unfortunately, an adversary that hoards its contents can later use his advanced capabilities to break the privacy of the encrypted shares and reconstruct the votes. The older blind signature-based protocol of [FOO92], achieves everlasting privacy goal, if one assumes a *perfectly* anonymous channel (as Theorem 3 of [FOO92] points). It resembles the shuffling of the ballot box contents, which in traditional elections provides a sense of everlasting privacy to the average voter, who as a human is computationally restricted.

The protocols of Moran and Naor [MN06; MN10] further elaborate on providing everlasting privacy through perfectly hiding commitment schemes. They propose a concrete voting system that provides universal verifiability, receipt-freeness and everlasting privacy. Additionally, they do not require the voter to perform complex calculations which makes their scheme easily usable by humans. In more details, their proposal consists of two authorities that communicate through a private channel and cooperate in order to produce the commitments that the voter selects. To tally the votes, the authorities work together (privately again) to shuffle the commitments and their openings. The latter are encrypted separately using a homomorphic cryptosystem providing computational secrecy and as a result, there are two 'parallel' shuffles. In the end, the perfectly hiding commitments can be safely opened to produce the result. Everlasting privacy is achieved under the assumption that the two authorities do not collude, and the commitment openings are not made public and thus available to the future adversary. If only a single authority is honest, then the scheme of Moran and Naor only provides computational privacy, while if both authorities are corrupted then the system provides only correctness. Despite proving the security of their protocol in the UC framework, the threat model for everlasting privacy is not formally captured. It merely rests on the perfect secrecy of the commitment scheme and an informal description of the adversary's capabilities. Note that in the future an attacker, that functions as an insider, can have an equivalent effect as if at least one of the authorities was corrupted, which means that the system of [MN10] does not provide everlasting privacy under this stronger threat model.

Subsequent works further elaborate and generalize this technique of splitting voting data into public and private parts, where the private data are never given to the adversary thus achieving a special version of everlasting privacy - towards the public. For instance, in [DGA12] the authors apply this procedure to the Helios [Adi08] voting system, by replacing the exponential ElGamal encryptions with Pedersen commitments that are published to the bulletin board. Their opening values are sent to the tallier encrypted through private channels. In [CPP13], a relevant primitive - commitment consistent encryption (CCE) is introduced. It allows the voters to derive commitments from their encrypted votes. These commitments are then posted to a

public bulletin board for verifiability purposes. If they are perfectly hiding, then the voting scheme has everlasting privacy. Tallying takes place in parallel using a private bulletin board, where the decryption of the result of the homomorphic combination of the votes takes place. They also provide security definitions for the privacy properties of their scheme but not for everlasting privacy in general. Furthermore, in [BDV13] this splitting technique is applied to create two synchronized mixnets that operate in parallel, mixing public commitments and private decommitment values, respectively.

The central idea in all the works presented so far is that a future adversary might be more powerful in terms of computing power, but he will lack access to data contemporary to the election or private data available to the authorities. This was noted and formalized in [Ara+13] with the notion of *practical* everlasting privacy. However, the formalization used the applied pi-calculus and not the more expressive indistinguishability cryptographic games. Using automated tools, the authors of [Ara+13] proved that the protocols of [MN10] and [DGA12] possess practical everlasting privacy. However, they did not apply their definition to schemes based on blind signatures and anonymous channels. Moreover, the reliance on private channels assumes an external adversary, who has a view of the system similar to the view of the voter. This excludes adversaries that cooperate with the election authorities, who in our opinion are more powerful and more likely to be the perpetrators of a future attack.

More recent works revisit the idea of an anonymous channel to add everlasting privacy to voting schemes. As the anonymous channel is a necessary condition for coercion resistance, these schemes also try to combine these two goals. In [LH15], the voter casts an unencrypted choice to the bulletin board along with commitments to their voting credential. The use of an anonymous channel and the fact that the voting credential consists of two parts, prevents a future adversary from associating the choice of a voter with her identity. A variation of this protocol was presented in [LHK16] to offer coercion resistance using deniable vote updating. To achieve coercion resistance, votes can be overwritten and only the last one counts. As a result, a voter under coercion can save her real vote for the end. This is a much stronger assumption than a simple moment of privacy required by the JCJ framework; for example, an adversary who is able to cast a last-minute vote achieves coercion. In [Iov+17], a version of Selene enhanced for JCJ coercion resistance is equipped with everlasting privacy towards the public with the use of pseudonyms. However, the creation process of pseudonyms and their relationship to real voter IDs and credentials requires trust assumptions and private channels between the members of the registration authority.

In our work [GPZ17; Gro+18; GPZ19; Gro+20], presented in chapter 3 and concluded in chapter 5 we try to combine everlasting privacy and coercion resistance under

weaker assumptions. We start with the scheme of [FOO92] (cf. subsection 4.5.3) and we solve the ballot stuffing problem with the PACBS primitive we described in chapter 3. The conditional verifiability property of PACBS also assists to achieve coercion resistance. The architecture of the proposed voting scheme allows tallying without trusting the authorities. The blindness of the signatures along with the use of an anonymous channel facilitates everlasting privacy. To reason about the way to achieve it we introduce the following formalization.

4.6.1 Game based definitions for everlasting privacy

Our model considers an adversary, who can corrupt voters and use them to learn what the honest voters voted. More specifically, our adversary is assumed to have the following capabilities:

- They can passively (as there will be no vote casting) examine the public ballot data found in the BB, without any further distinguishing information (e.g. which ballots belong to honest voters and which to corrupted ones).
- They can cooperate with the contemporary adversary $\mathcal{A}$ and utilize the voters controlled by them. Consequently, they can pinpoint the ballots originating from adversarial voters.
- They can utilize leaked election and communication data, obtained in the real world by taking control of state and communication agencies, such as election authorities and internet service providers.

We incorporate these cases in our definitions, by assuming a pair of algorithms $(\mathcal{A}, \hat{\mathcal{A}})$ where $\mathcal{A}$ is a PPT algorithm and $\hat{\mathcal{A}}$ is computationally unbounded. The former participates actively in the election by corrupting voters and the latter looks at the election transcript. leakage from communication channels denoted Aux and the information gathered by $\mathcal{A}$.

For the everlasting privacy property, we define three games to capture the differences in the strategy and knowledge of the future adversary. These games depend on U-BPRIV defined in Algorithm 4.11. In all of them the adversary $\hat{\mathcal{A}}$ is unbounded and invokes the election system that is controlled by the challenger.

In particular, the weaker version of everlasting privacy WE-BPRIV is meant to capture a strong adversary which views only the publicly available information in the BB for an election. Based on his computational power he can compute the tally (e.g. by decrypting) No more data is available to him.

Algorithm 4.12: Weak everlasting privacy game $\mathsf{WE\text{-}BPRIV}^{\mathfrak{b}}_{\hat{\mathcal{A}},\mathsf{VS}}$

$(\mathsf{BB}_{\mathfrak{b}}, T) \leftarrow \hat{\mathcal{A}}^{\mathsf{VS}}()$
$\mathfrak{b}' \leftarrow \hat{\mathcal{A}}(\mathsf{BB}_{\mathfrak{b}}, T, \textbf{guess})$
return $\mathfrak{b} = \mathfrak{b}'$

This is formalized in Algorithm 4.12, where the future adversary $\mathcal{A}$ invokes the voting system. After the execution, the adversary receives the BB picked by the coin $\mathfrak{b}$ and the tally and tries to guess β.

Definition 4.11: Weak everlasting privacy

A voting scheme VS has the weak everlasting privacy property, if for every algorithm $\hat{\mathcal{A}}$ there exists a negligible function negl such that for every L it holds that:

$$\Pr\left[\mathsf{WE\text{-}BPRIV}^{0}_{\hat{\mathcal{A}},\mathsf{VS}}(\lambda, \mathsf{L}) = 1\right] - \Pr\left[\mathsf{WE\text{-}BPRIV}^{1}_{\hat{\mathcal{A}},\mathsf{VS}}(\lambda, \mathsf{L}) = 1\right] \leq \mathsf{negl}(\lambda)$$

In stronger versions of everlasting privacy, the future adversary remains unlimited computationally, but gradually has access to increasing data to utilize. In the 'plain' everlasting privacy game the future adversary considers the transcripts obtained by the contemporary adversary during the execution of the protocol. Note that since corruption information is used the restriction t on the number of corrupted voters applies as well.

Algorithm 4.13: Everlasting privacy $\mathsf{E\text{-}BPRIV}^{\mathfrak{b}}_{\mathcal{A},\hat{\mathcal{A}},\mathsf{VS},t}$

$(\mathtt{vt}_0, \mathtt{vt}_1, \mathtt{Corr}) \leftarrow \hat{\mathcal{A}}()$
$(\mathsf{BB}_{\mathfrak{b}}, \mathsf{Trans}_{\mathfrak{b}}, T) \leftarrow \hat{\mathcal{A}}^{\mathsf{VS},\mathcal{A}}()$
$\mathfrak{b}' \leftarrow \hat{\mathcal{A}}(\mathsf{BB}_{\mathfrak{b}}, \mathsf{Trans}_{\mathfrak{b}}, \textbf{guess})$
return $\mathfrak{b} = \mathfrak{b}'$ AND $|V_{\mathtt{Corr}}| \leq t$

Definition 4.12: Everlasting privacy

A voting scheme VS has the everlasting privacy property, if for every pair of algorithms $(\mathcal{A}, \hat{\mathcal{A}})$ there exists a negligible function negl such that for every L it holds that:

$$\Pr\left[\mathsf{E\text{-}BPRIV}^{0}_{\mathcal{A},\hat{\mathcal{A}},\mathsf{VS},t}(\lambda, \mathsf{L}) = 1\right] - \Pr\left[\mathsf{E\text{-}BPRIV}^{1}_{\mathcal{A},\hat{\mathcal{A}},\mathsf{VS},t}(\lambda, \mathsf{L}) = 1\right] \leq \mathsf{negl}(\lambda)$$

Finally, in the strongest version of everlasting privacy SE-BPRIV, the computationally unbounded adversary $\hat{\mathcal{A}}$ obtains all data generated by the protocol both main and auxiliary.

Algorithm 4.14: Strong everlasting privacy $\mathsf{SE\text{-}BPRIV}^{\mathfrak{b}}_{\hat{\mathcal{A}},\mathsf{VS},t}$

$(\mathtt{vt}_0, \mathtt{vt}_1, \mathtt{Corr}) \leftarrow \hat{\mathcal{A}}()$
$(\mathsf{BB}_{\mathfrak{b}}, \mathsf{view}_{\mathcal{A}}, \mathsf{Trans}_{\mathfrak{b}}, \mathsf{Aux}_{\mathfrak{b}}, T) \leftarrow \hat{\mathcal{A}}^{\mathsf{VS},\mathcal{A}}()$
$\mathfrak{b}' \leftarrow \hat{\mathcal{A}}(\mathsf{BB}_{\mathfrak{b}}, \mathsf{Trans}_{\mathfrak{b}}, \mathsf{Aux}_{\mathfrak{b}}, \textbf{guess})$
return $\mathfrak{b} = \mathfrak{b}'$ AND $|V_{\mathtt{Corr}}| \leq t$

Definition 4.13: Strong everlasting privacy

A voting scheme VS has the strong everlasting privacy property, if for every pair of algorithms $(\mathcal{A}, \hat{\mathcal{A}})$ there exists a negligible function negl such that for every L it holds that:

$$\Pr\left[\mathsf{SE\text{-}BPRIV}^{0}_{\mathcal{A},\hat{\mathcal{A}},\mathsf{VS},t}(\lambda, \mathsf{L}) = 1\right] - \Pr\left[\mathsf{SE\text{-}BPRIV}^{1}_{\mathcal{A},\hat{\mathcal{A}},\mathsf{VS},t}(\lambda, \mathsf{L}) = 1\right] \leq \mathsf{negl}(\lambda)$$

4.6.2 Application of the new everlasting privacy definitions

We now apply our definitions to two representative schemes; one for each of the main approaches to privacy in the literature.

The AugFOO scheme provides strong everlasting privacy

First, we characterize the everlasting privacy provided by the AugFOO scheme (cf. Figure 4.2, Figure 4.3,Figure 4.4)

Theorem 4.1: AugFOO and strong everlasting privacy

The AugFOO scheme provides strong everlasting privacy.

Proof. The data available to $\hat{\mathcal{A}}$ for an honest voter are the following:

- All the public keys and parameters.
- The public data posted on the BB in the registration and voting phases $(i, \mathsf{b}'_i, \overline{\sigma}_i)$, $(\mathsf{b}_i, \pi_{\mathsf{V}_i}, \overline{\sigma}_{i,\mathsf{RA}})$.
- The private keys $\mathsf{sk}_{\mathsf{RA}}, \mathsf{sk}_{\mathsf{TA}}$.
- The private transcript of the corrupt voters that reveal their vote.

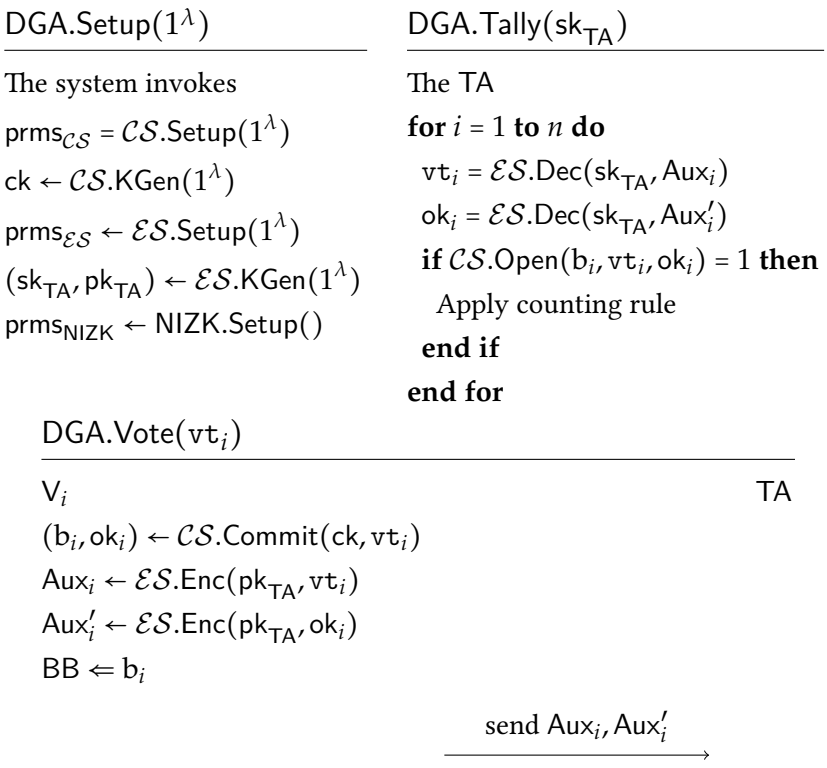


FIGURE 4.5: The DGA scheme of [DGA12]

Note that because of anonymous casting the leaked data Trans, Aux are nullified, which means that $\mathcal{A}$ cannot identify the voter. The tuple $(i, \mathsf{b}'_i, \overline{\sigma}_i)$ is also useless to the adversary since b'_i is blinded with information-theoretic security. While the challenger consistently places both $(i, \mathsf{b}'_i, \overline{\sigma}_i), (\mathsf{b}_i, \pi_{\mathsf{V}_i}, \overline{\sigma}_{i,\mathsf{RA}})$ in the same BB, $\hat{\mathcal{A}}$ does not get any advantage because of the unlinkability of blind signatures. From b_i, $\hat{\mathcal{A}}$ can obtain the choice of the voter. However, like Lemma 4.1, the anonymous channel used during casting prohibits $\hat{\mathcal{A}}$ from distinguishing in which BB this choice has been placed. ■

The DGA scheme provides everlasting privacy

Informally, in DGA the voters publish a Pedersen commitment of their vote in the BB, along with a witness indistinguishable proof of correct ballot formation. They also submit the respective openings to the TA, the Helios server. The system has two variations; it either homomorphically combines the commitment ballots or it mixes them. The same operation is performed on the openings in parallel. We describe DGA more formally in Figure 4.5.

As DGA is a variation of Helios, it can be proved secure using plain BPRIV, by adapting the proof in [Ber+15]. Concerning everlasting privacy, the authors of DGA admit in [DGA12] that it only provides everlasting privacy towards the public. Towards the election authority, DGA provides Helios-level privacy, which means that a corrupted

or computationally powerful TA can reveal the selections of the voters. We can reach the same results by applying our model.

Theorem 4.2: DGA **and strong everlasting privacy**

The DGA scheme does not provide strong everlasting privacy.

Proof. In Algorithm 4.14, $\hat{\mathcal{A}}$ obtain the auxiliary data:

$$\mathsf{Aux} = \{\mathsf{Aux}_i, \mathsf{Aux}'_i\}_{i=1}^{n} = \{\mathcal{ES}.\mathsf{Enc}(\mathsf{pk}_{\mathsf{TA}}, \mathtt{vt}_i), \mathcal{ES}.\mathsf{Enc}(\mathsf{pk}_{\mathsf{TA}}, \mathsf{ok}_i)\}_{i=1}^{n}$$

Since they are computationally unbounded, they can break the encryption scheme and obtain both $\mathtt{vt}_i, \mathsf{ok}_i$ for all the voters. They can also obtain Trans_β for $\beta \in \{0,1\}$ and the contents of both $\mathsf{BB}_0, \mathsf{BB}_1$ which now contain the commitments from $\mathcal{A}$. The former also includes all the choices selected by $\mathcal{A}$ for voters in both $\mathsf{V}_{\mathtt{Corr}}$ and $\mathsf{V}_{\mathtt{Hon}}$. Using $\mathtt{vt}_i, \mathsf{ok}_i$ they can validate all commitments in both bulletin boards and deduce which of $\mathsf{BB}_0, \mathsf{BB}_1$ they are viewing with certainty. ■

Theorem 4.3: DGA **and everlasting privacy**

The DGA scheme provides everlasting privacy.

Proof. Firstly, we note that DGA provides weak everlasting privacy. In Algorithm 4.12, $\hat{\mathcal{A}}$ views only the contents of both BB, having no information about which ballots belong to corrupt voters. However, the commitment scheme hides the ballot contents perfectly and thus $\hat{\mathcal{A}}$ cannot win Algorithm 4.12.

In Algorithm 4.13, $\hat{\mathcal{A}}$ obtains additionally the view of $\mathcal{A}$, namely the options of the corrupt voters and the choices $\mathtt{vt}_0, \mathtt{vt}_1$ from $\mathcal{A}$ used in the calls to **Vote**. These pieces of data provide no advantage to $\hat{\mathcal{A}}$, as the option of corrupted voters where already known to them and the options of the honest voters are honestly hidden. ■

4.6.3 Discussion

Our models and their evaluation indicate that it is not enough to use an information-theoretically hiding scheme to achieve strong everlasting privacy. To do so, such schemes need to be accompanied by an anonymous casting phase. Interestingly, this is the intuition applied to physical on-site voting systems. The voters do not merely hide the ballot (by concealing it inside an envelope) but they also anonymize it by mixing it with identical envelopes inside the ballot box.

On the other hand, a critic of the anonymity approach can point out that it trades one problem for another. Instead of a perfectly hiding system, a perfectly anonymous

scheme is required to provide strong everlasting privacy. In our view, however, this is not the case. An anonymous channel might not be in the (full) control of a future adversary. It might be distributed, operated (in part) by non-governmental organizations and it might even transcend national boundaries. In theory, this seems easier to accomplish, than trusting only the election authorities to keep private or securely destroy sensitive data such as decommitment values even from insiders. Furthermore, alternative methods of anonymity 'on the client-side' can be applied. For instance, it isn't hard to imagine a voting scheme where the locally-connected voters of a particular polling station create *small-scale* anonymity sets to obfuscate their ballots. As a result, by keeping at least a single component of the anonymous channel out of the control of the future adversary, strong everlasting privacy can be attained.

4.7 Relations between properties and models

To conclude this chapter, we review the relations between the various properties we described in the previous sections.

We begin, with the conflict of verifiability and privacy that is evident in many voting schemes. [Che+10] prove that universal verifiability and unconditional privacy cannot exist unless everyone votes. The reasoning for this, is that for universal verifiability there must be a list of eligible voters and corresponding individual votes that are summed to the tally. An adversary that is not constrained, can find for all sub lists of voters, that contain one less voter, the corresponding tally. By subtracting it from the original tally, he can uncover the preference of the voter that is excluded. Furthermore, in the same work it is proved that universal verifiability and receipt-freeness cannot coexist, unless private channels are present. The intuition behind this proof, rests on the fact that we accounted for in subsection 4.4.1, where the randomness used to construct a ballot can serve as a receipt. Since, this randomness, along with the vote uniquely determines the ballot it must be used to verify the ballot. As a result, if the receipt is absent, the scheme is not universally verifiable and if it is present it is not receipt-free.

Some interesting results concern individual verifiability (cf. subsection 4.3.1). It would seem reasonable to assume that if a voting scheme possesses individual verifiability, it possesses universal verifiability. However, as we saw in subsection 4.4.1, in BeleniosRF, a voter can verify her ballot, by checking the signature. However, the RA and the rerandomizing server can collude and undetectably change the contents of the ballot. Note, that for universal verifiability, every member of the EA must be regarded as being corrupted. As a result, BeleniosRF provides individual verifiability but not universal. [SFC15] reached the same result by constructing a scheme

with in which each voter posts her vote in plain together with a random nonce, i.e. $(\perp, (\mathtt{vt}_i, r_i)) \leftarrow \mathsf{VS.Vote}\langle \mathsf{EA}(), \mathsf{V}_i(\mathtt{vt}_i), \mathsf{prms}, \mathsf{V}_{\mathtt{El}}, \mathsf{CS}\rangle$ where $r_i \leftarrow_{\$} \{0,1\}^{\lambda}$. This scheme obviously possesses individual verifiability but is amenable to ballot stuffing. The reverse does not hold either. One can construct a scheme that has universal verifiability, where each individual voter cannot pinpoint her vote in the BB. In fact, the original version of the [FOO92] is such a scheme. As a result, universal verifiability and individual verifiability are orthogonal properties.

[SFC15] prove that if a scheme provides eligibility verifiability, then it will also provide individual verifiability. The reason is that eligibility verifiability guarantees that everybody can verify that a ballot is associated to a particular public key. This means that the holder of the private key, can pinpoint their ballot and thus the scheme is individually verifiable.

More surprisingly, [CL18] proves that if a scheme provides privacy then it provides individual verifiability, or for the contrapositive if the scheme does not provide individual verifiability it does not provide privacy. The reason for this is that if voters cannot verify that their ballots will be counted, then a corrupt EA can replace all the ballots except for one and thus learn what a particular voter voted. An interesting consequence of this result, is that voters who check their ballots, protect the privacy of other voters.

Ballot secrecy is also implied by receipt-freeness. If a scheme is not private, then there is no need for a receipt since it is public who everyone voted for. This is also evident from the BPRIV definitions in Algorithm 4.10 and Algorithm 4.9. It removes the need for an unbounded adversary. For this reason, everlasting privacy implies privacy. Regarding the reverse direction, it clearly does not hold. In fact, [KTV11] show some counter-intuitive results if one tries to quantitatively characterize ballot secrecy and coercion-resistance for specific voting schemes. Contrary to one's intuition: increasing privacy by forcing the voters to cast their ballots in a particular format, reduces coercion resistance. Informally, everlasting privacy is also related to coercion-resistance, by the observation that if a system reveals to a future adversary the contents of the vote, then the voter is de-incentivized to cast her true preference. However, this is not a direct consequence of any model and as a result, we do not form a logical implication.

A contested relation in the literature, concerns receipt-freeness and coercion resistance. These concepts do not have clear boundaries, as is evident from the fact that many schemes that are coercion resistant according to some models, are not receipt-free ([KZZ15a; Cha+16; FQS19]). Receipt-freeness defends against corrupt

voters, while coercion resistance defends against active adversaries and assumes anti-coercion measures from the voter. Since a corrupt voter will not take this anti-coercion strategy, a scheme that is not receipt-free will not be coercion resistance. In the [JCJ05] setting, things are clear: since a coercion resistant scheme, must be receipt-free, coercion-resistance implies receipt-freeness.

These relations are depicted in Figure 4.6.

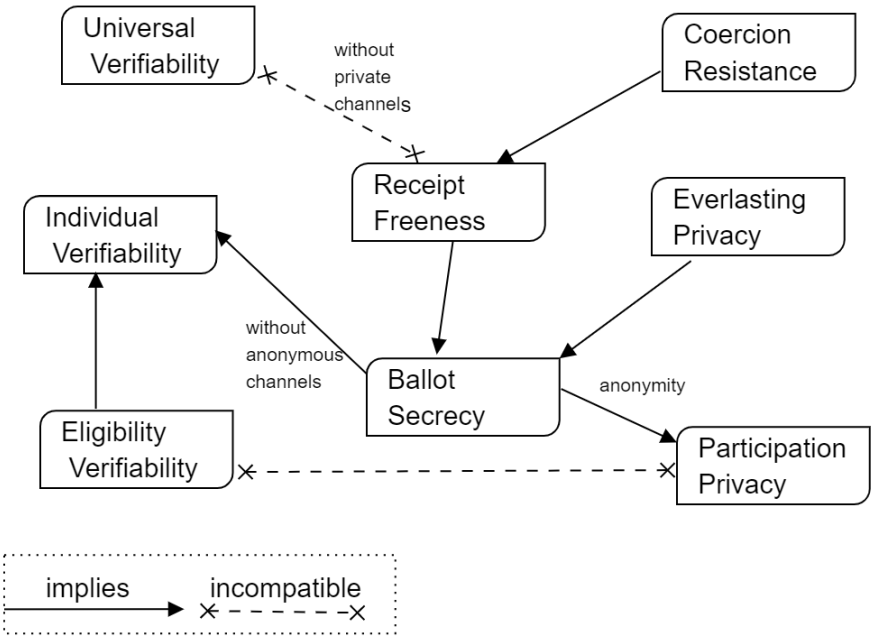


Figure 4.6: Relations between security properties of voting schemes

5 Voting with Publicly Auditable Conditional Blind Signatures

> The first principle is that you must not fool yourself and you are the easiest person to fool.
>
> Richard Feynman

In this chapter [1], we present a voting scheme that utilizes the PACBS primitive we analyzed in chapter 3. The main idea of this scheme is that the blind eligibility checks of [FOO92] are combined with a credential-based coercion resistant check and embedded into the PACB signature. This means that if the credential provided by the voter during the authorization phase, is the one created during registration, then the signature will be valid, and the vote will be counted. Otherwise, the signature will be invalid, and the vote will be discarded because it will be considered a result of coercion. While this is the intuition behind our scheme, there are many more details that need to be considered. For instance, the designated verifier of the PACBS must indicate that the signature is invalid for verifiability, without informing the coercer about this fact. Furthermore, the duplicates inherent in [JCJ05], must be weeded out in a verifiable manner. All these aspects of the scheme are detailed in the current chapter. The claimed properties are also proved using ideas from the models in chapter 4.

5.1 Overview

One of the disadvantages of the [JCJ05] coercion resistance scheme is the quadratic number of checks required to weed out coerced and duplicate votes. As we saw in section 4.4.2, there have been many efforts in the literature, trying to speed up this process, without sacrificing coercion resistance and verifiability. Our approach was first detailed in [GPZ17], where we noted a new possibility stemming from the combination of the scheme in [FOO92] with its variations and [JCJ05]-type schemes.

[1] An extension of [GPZ17; Gro+18; Zac18; Gro+20]

Recall, that in [FOO92; Ohk+99], the voter sends an authorization request consisting of identification information and a blinded vote. The RA checks eligibility and then signs the vote. Our idea is to embed credential weeding into this eligibility verification performed by the RA. This can improve the complexity of detecting coerced votes, bringing it down to a linear function of the number of *voters* (instead of votes as in [JCJ05]). During this authorization phase, the voter identity is known, so only the credentials assigned to it should be considered, instead of checking the tokens of *all* voters. As a result, the RA would be able to tell if a particular vote should be counted. For the actual counting, however, this fact should be conveyed to the TA, which can be done by extending the scope of the RA's signature. Instead of only indicating eligibility, it can also signal if the authenticating credential is genuine or fake, for the vote to be counted by the TA or not. All these must be done verifiably. Every election stakeholder should be able to audit the process to check that the two authorities followed the protocol. However, the voters, should also be convinced that only the vote corresponding to the registered credential was counted. This fact should not be conveyed to the coercer. Our novel contribution is that all these checks are embedded the PACBSprimitive. Our voting scheme [Gro+18] uses the OSPACBS instantiation of PACBS section 3.6.

The general workflow, from the point of view of a single voter V_i, is depicted in Figure 5.1.

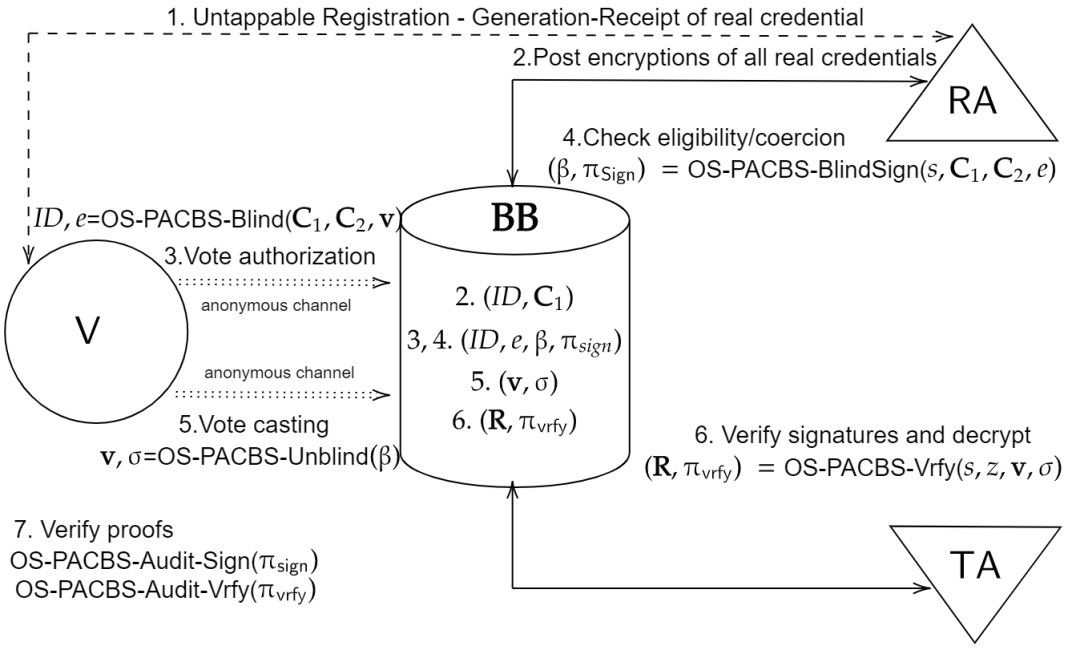


Figure 5.1: PACBS voting for an individual voter

The election authority EA consists of two sub-authorities RA, TA that handle the registration and tallying functionalities respectively. They, in turn, consist of many

members, with conflicting interests, that share cryptographic keys. Real-world elections are usually organized by a central entity so it is not uncommon for the functionalities of the RA, TA to be performed by the *same* functional election authority. As a result, the assumption that the authorities can share a cryptographic key (in this case the PACBS secret key s) is not restricting or unrealistic. Of course, all their members are considered corrupted to provide for universal verifiability.

The scheme $\mathsf{VS}_{\mathsf{PACBS}}$ of Figure 5.1 consists of five phases. We detail them below and match them with the functionalities defined in Definition 4.1:

- *Setup* comprising the functionalities $\mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Setup}$, $\mathsf{VS}_{\mathsf{PACBS}}.\mathsf{SetupElection}$.
- *Registration* implemented through $\mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Register}$.
- *Voting* is split into two sub-phases: authorization and casting realized by the functionalities $\mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Vote}$, $\mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Cast}$. The functionality $\mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Valid}$ is executed after casting and performs ballot weeding. Helper functionalities used in this case are $\mathsf{VS}_{\mathsf{PACBS}}.\mathsf{fakekey}$, $\mathsf{VS}_{\mathsf{PACBS}}.\mathsf{chaffvote}$, $\mathsf{VS}_{\mathsf{PACBS}}.\mathsf{dupauth}$ to generate a fake credential, inject fake votes for coercion resistance and to weed out duplicate authorization requests.
- *Tally*, which is implemented by $\mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Tally}$.
- *Verification* which comprises of $\mathsf{VS}_{\mathsf{PACBS}}.\mathsf{VerifyBallot}$, $\mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Verify}$.

In short, the protocol operates as follows (a detailed analysis will be provided in section 5.2).

The EA executes the $\mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Setup}$ functionality to create the cryptographic parameters of the system. These parameters are to be used across multiple elections. Each voter V_i participates in the generation of the real credential through an untappable pre-election registration. This is implemented using the $\mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Register}$ functionality, that generates a credential intended for use in many elections. The EA executes the $\mathsf{VS}_{\mathsf{PACBS}}.\mathsf{SetupElection}$ functionality to create a particular election. When the registration phase ends, the RA posts a list of pairs consisting of voter identity i and an encrypted genuine credential to the publicly available BB. Assume that voter V_i has obtained a credential θ_i encrypted as $\boldsymbol{C}_{i1}$ under the RA public key. Then $(i, \boldsymbol{C}_{i1})$ will be included in the public credential list.

During the voting phase, V_i first performs an authorization request, by executing $\mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Vote}$. To this end, she recomputes an encryption of the credential as $\boldsymbol{C}_{i2}$ and computes the encryption v_i of her vote $\mathtt{vt}_i$. If the voter is under coercion, she will use a fake credential θ_i'. Otherwise, she will use the genuine, making $\boldsymbol{C}_{i2}$ encrypt the same plaintext as $\boldsymbol{C}_{i1}$ (i.e. θ_i). Subsequently, she executes the $\mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Vote}$ functionality,

by initiating the OSPACBS.Sign protocol with the RA. More specifically, $\boldsymbol{v}_i$ is blinded using the OSPACBS.Blind algorithm. Both encryptions $\boldsymbol{C}_{i1}, \boldsymbol{C}_{i2}$ are attached to the authorization request. Note that there is no need to encrypt $\boldsymbol{v}_i$ with the same public key as $\boldsymbol{C}_{i1}, \boldsymbol{C}_{i2}$. However, for simplicity, we assume that this is the case. Then V_i produces the OSPACBS.Blind output e_i, and submits the vote authorization request, which includes voter identification information as in [FOO92]. The RA uses this information to find out if V_i is eligible to vote and uses the OSPACBS.BlindSign functionality to compute the blind signature $\overline{\beta}_i$, conditional to the predicate $\mathsf{pred}(\boldsymbol{C}_{i1}, \boldsymbol{C}_{i2}) = 1 \Leftrightarrow \mathsf{Dec}(\boldsymbol{C}_{i1}) = \mathsf{Dec}(\boldsymbol{C}_{i2})$ and verifiable by the TA. OSPACBS.BlindSign also produces the proof $\pi_{i,\mathsf{Sign}}$. V_i retrieves $\overline{\beta}_i$ and invokes the OSPACBS.Unblind functionality to obtain the plain signature $\overline{\sigma}_i$ which is then recast with $\boldsymbol{v}_i$, as the ballot for V_i, using the $\mathsf{VS}_{\mathsf{PACBS}}$.Cast functionality. In order to accept the voter ballot the BB executes the $\mathsf{VS}_{\mathsf{PACBS}}$.Valid functionality.

For tallying, the TA invokes the $\mathsf{VS}_{\mathsf{PACBS}}$.Tally functionality which must count only the votes that correspond to uncoerced ballots i.e. ballots for which $\mathsf{pred}(\boldsymbol{C}_{i1}, \boldsymbol{C}_{i2}) = 1$ when signed by the RA. This can be determined by the execution of the OSPACBS.Verify functionality for each ballot - signature pair. However, if the TA is 'careless' and simply executes this functionality, its published result will notify the coercer if his coercion attempt succeeded, beating the purpose of the fake credentials mechanism. To avoid this problem, the TA will perform the validation in two steps: Firstly, it will compute $\boldsymbol{R}_i$ and $\pi_{i,\mathsf{Verify}} = (\pi_{i1}, \pi_{i2}, \pi_{i3})$ as specified in Algorithm 3.13 and post them to the BB. Then it will use a verifiable shuffle to disassociate all entries from their identity and from when they were originally cast. The result of the shuffle will be verifiably decrypted and counted if and only if $\mathsf{Dec}(\boldsymbol{R}_i) = 1$. The π_{i4} will be posted to the BB. By the construction of the PACBS scheme, this means that only the votes that were not a product of coercion, as indicated by the usage of the correct credential will be counted.

The $\mathsf{VS}_{\mathsf{PACBS}}$.VerifyBallot can to provide individual verifiability. It includes the functionality AuditSign. The $\mathsf{VS}_{\mathsf{PACBS}}$.Verify functionality can be invoked by any interested party and reveals if every participant followed the protocol. To the voter, however, who has the private input of which credential she used to cast her ballot this will also signify if the ballot was counted or not. This is not checkable by the coercer who lacks this secret piece of information and only receives the knowledge of the public. This means, that the system is receipt-free as well.

5.2 PACBS Voting Scheme Specification

We now detail how the functionalities from Definition 4.1 are instantiated using PACBS. In the following, all public data are appended to the BB. For readability, we assume that the BB is split into sections, where the appropriate data from each functionality are stored. These sections are identified by the use of an appropriate subscript.

5.2.1 Setup phase

The setup phase contains the $\mathsf{VS}_{\mathsf{PACBS}}$.Setup and the $\mathsf{VS}_{\mathsf{PACBS}}$.SetupElection functionalities that aim to initialize the underlying cryptosystem and select the election parameters for each election, respectively.

$\mathsf{VS}_{\mathsf{PACBS}}$.Setup functionality

The election authority EA initializes the PACBS scheme by invoking the PACBS.Gen algorithm (Algorithm 3.11). All the generated private keys are split between the agents, but we treat them as controlled by a single entity. For simplicity, we will use the encryption scheme that is employed in PACBS to encrypt credentials, to also encrypt the vote. However, the latter is not required as any IND-CPA secure cryptosystem could be used for this task. A random group generator g is also selected to be used to encode the initial voter credentials into group elements. The credentials are assumed to be mapped to elements of $\mathbb{Z}_q$ (ultimately). All the public parameters generated are stored in a special section of the BB in the end.

Algorithm 5.1: The $\mathsf{VS}_{\mathsf{PACBS}}$.Setup algorithm executed by the EA for all elections

Input : security parameter λ
Output: $\mathsf{prms}, \mathsf{sk}, \mathsf{pk}$

$(\mathsf{prms}, \mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{PACBS.PACBS.Gen}(1^\lambda)$
/* $\mathsf{prms} = (q, \mathbb{G}, g_1, g_2, v, \mathsf{pred}, \mathsf{H}_1, \mathsf{H}_2)$ */
/* $\mathsf{sk} = (s, z)$ */
/* $\mathsf{pk} = (k, h)$ */
$g \leftarrow_\$ \mathbb{G}$
$\mathsf{prms} := (q, \mathbb{G}, g_1, g_2, v, \mathsf{pred}, \mathsf{H}_1, \mathsf{H}_2, g)$
$\mathsf{sk}_{\mathsf{Enc}} := z$
$\mathsf{pk}_{\mathsf{Enc}} := h$
$\mathsf{sk}_{\mathsf{PACBS}} := s$
$\mathsf{pk}_{\mathsf{PACBS}} := k$
$\mathsf{BB}_{\mathsf{prms}} \Leftarrow (\mathsf{prms}, \mathsf{pk}_{\mathsf{Enc}}, \mathsf{pk}_{\mathsf{PACBS}})$

$\mathsf{VS}_{\mathsf{PACBS}}$.SetupElection functionality

At the beginning of each election the EA generates the list of candidates (candidate slate CS) and the list of eligible voters $\mathsf{V}_{\mathtt{El}}$. For the former, a suitable encoding must be selected so that the candidates are mapped to the message space of the cryptosystem. This encoding will of course affect the counting function. While our framework can be adapted to both homomorphic and plain vote counting, in Algorithm 5.2 a random group element is selected and assigned to each candidate. When the ballots are decrypted, the plaintexts are compared to the respective 'prototypes' of the $\mathsf{CS} \in \mathsf{BB}$ and then counted. For the latter, additional external information L is required (e.g. citizenship or membership records, start time and end time $T_{\text{start}}, T_{\text{end}}$). We assume an external functionality check that given a voter identity and the database L, returns if the voter in question has the right to vote or not. In the end, both lists are appended to the BB.

Algorithm 5.2: The $\mathsf{VS}_{\mathsf{PACBS}}$.SetupElection algorithm executed by the EA for a particular election

Input : $n, m, \mathsf{prms}, \mathsf{L}, \{\boldsymbol{C}_{i1}\}_{i=1}^{n}$
Output: $\mathsf{CS}, \mathsf{V}_{\mathtt{El}}$

$\mathsf{CS} := \varnothing$
for $j \in [m]$ **do**
  $\mathtt{vt}_j \leftarrow_{\$} \mathbb{G}$
  $\mathsf{CS} := \mathsf{CS} \cup \mathtt{vt}_j$
end
$\mathsf{V}_{\mathtt{El}} := \varnothing$
for $i \in [n]$ **do**
  if $\mathsf{check}(i, \mathsf{L}) = 1$ **then**
    $\mathsf{V}_{\mathtt{El}} := \mathsf{V}_{\mathtt{El}} \cup (i, \mathsf{Rebase}(\boldsymbol{C}_{i1}))$
  end
end
$\mathsf{BB}_{\text{election}} \Leftarrow (\mathsf{CS}, \mathsf{V}_{\mathtt{El}})$

The $\mathsf{VS}_{\mathsf{PACBS}}$.SetupElection functionality assumes that there exists a list of encryptions of the registered voter credentials produced by the multi-election $\mathsf{VS}_{\mathsf{PACBS}}$.Register functionality. This list is presented as input to $\mathsf{VS}_{\mathsf{PACBS}}$.SetupElection. To allow credential reuse for many elections a Rebase functionality transforms them from their initial form g^{θ} to their new form g_0^{θ}, where g_0 is a random group generator computed for each new election. The proof π_{Rebase} is a conjunction of a Schnorr proof (π_S from Figure 2.1) and Chaum-Pedersen proofs (π_{CP} from Figure 2.2). This functionality can be performed in a distributed manner, to reduce the trust required to the election authority.

Algorithm 5.3: The Rebase algorithm for changing credentials between elections

Input : $\boldsymbol{C}_1 = (g^r, g^\theta h^r)$
Output: $\boldsymbol{C}'_1$

$b \leftarrow_\$ \mathbb{Z}_q$
$g_0 := g^b$
$\mathsf{BB}_{\text{election}} \Leftarrow g_0$
$\boldsymbol{C}'_1 := ((g^r)^b, (g^\theta h^r)^b) = (g^{rb}, g_0^\theta\, h^{rb})$
$\pi_{\mathsf{Rebase}} := \mathsf{NIZK}\{(\mathsf{prms}, \boldsymbol{C}_1, \boldsymbol{C}'_1, g_0), (b) : g_0 = g^b \text{ AND } \boldsymbol{C}'_1 = \boldsymbol{C}_1^b\}$
return $(\boldsymbol{C}'_1, \pi_{\mathsf{Rebase}})$

5.2.2 Registration phase

$\mathsf{VS}_{\mathsf{PACBS}}$.Register functionality

The registration phase aims to create a genuine voter credential. As such, it is overly sensitive to the goal of coercion resistance, since a possible leak can result in a simulation attack. As a result, the generated credential must be *deniable* (cf. the specific assumptions in section 4.4). We assume that the registration phase takes place through an untappable channel, that is implemented by physical means i.e. in-person registration. This might be contrary to the online voting concept, but in our case the credentials can be reused for many elections, through the Rebase mechanism. Our system can support both registration methods in the literature:

- Using something the voter 'knows', like a password. This approach was used in Selections [CH11], where the voter registers a password from a panic password system [CH08]. For instance, the RA could set up a commonly used set of words. Each password consists of a combination of several words. The voter registers a particular combination as the normal password. Any other combination is considered a panic password. In both cases, the user experience is the same, so that the coercer cannot distinguish the type of password typed by the user. Note that in such a system, invalid passwords (maybe due to typing errors), are different from panic passwords and can be treated in the usual way - the user can simply retry.
- Using something the voter 'has', like a tamper-resistant cryptographic token (e.g. smart-card) that has a cryptographic key embedded and can be used to perform cryptographic computations - reencryptions, (designated verifier) proofs computations etc. This approach was used in CIVITAS [CCM08].

In the former case, [CH11], we assume that there exists a function $\phi : \{0,1\}^* \to \mathbb{Z}_q$ that transforms a password to a group index θ. Subsequently, the voter encrypts the corresponding group element g^θ using the public key of the RA and submits it. The RA

posts a rerandomization of the encrypted credential to the BB. The proof π_{i1} proves knowledge of a lifted ElGamal plaintext (cf. π_m in Figure 2.5). The proof π'_{i1} proves correct reencryption (cf. π_{ReEnc} in section 2.4.1). If the voter is assumed to use an untrusted computational device to perform the computations then this process must be repeated for α credentials, with the user selecting a single one using a mechanism such as Benaloh challenges [CH11; Ben06].

Common input: prms, $\mathsf{pk}_{\mathsf{RA}}$, the voter ID i
Assumption: Runs over an untappable channel

$\mathbf{V}_i$ creates a credential:

- selects a valid password pwd_i from a panic password system
- computes $\theta_i := \phi(\mathsf{pwd}_i)$
- computes $\boldsymbol{C}_{i1} := \mathsf{Enc}_h(g^{\theta_i}, r_i)$ and $\pi_{i1} := \mathsf{NIZK}\{(\mathsf{prms}), (\theta_i, r_i) : \boldsymbol{C}_{i1} = \mathsf{Enc}_h(g^{\theta_i}, r_i)\}$
- submits $(\boldsymbol{C}_{i1}, \pi_{i1})$ to the RA

RA creates the voter roll entry

- computes $\boldsymbol{C}'_{i1} := \mathsf{ReEnc}(\boldsymbol{C}_{i1})$
  and $\pi'_{i1} := \mathsf{NIZK}\{(\mathsf{prms}, \boldsymbol{C}_{i1}, \boldsymbol{C}'_{i1}), (r') : \boldsymbol{C}'_{i1} = \mathsf{ReEnc}_h(\boldsymbol{C}_{i1})\}$
- $\mathsf{BB}_{\mathsf{Register}} \Leftarrow (\boldsymbol{C}'_{i1}, \pi'_{i1})$

FIGURE 5.2: The $\mathsf{VS}_{\mathsf{PACBS}}$.Register protocol executed through an untappable channel (Selections version)

In the latter case, [CCM08], the RA functionality is split among many members EA_j. The voter creates a credential share with each member and derives the final credential from their combination. The proof π_{ij} proves knowledge of a lifted ElGamal plaintext (cf. π_m in Figure 2.5). We assume that the voter has a key pair $(\mathsf{pk}_{\mathsf{V}_i}, \mathsf{sk}_{\mathsf{V}_i})$ that is going to be used for the designated verifier proof of correct reencryption (δ_{ReEnc} from Figure 2.9) that achieves deniability.

5.2.3 Voting phase

Coercion evasion functionality fakekey

To achieve coercion resistance our voting protocol utilizes a function fakekey that the voter executes when the coercer is present. This function creates a new anonymous and indistinguishable credential that the voter will use to cast her coerced vote. The implementation of this function depends on how the credential was created and registered.

Common input: $\mathsf{prms}, \mathsf{pk}_{\mathsf{RA}}$, the voter ID $i, \mathsf{pk}_{\mathsf{V}_i}$
Private RA input: $\mathsf{sk}_{\mathsf{RA}}$
Assumption: Runs over an untappable channel

Each member EA_j of the RA generates its credential part:

- selects $\theta_{ij} \leftarrow\!\!\$\, \mathbb{Z}_q$
- computes $s_{ij} := g^{\theta_{ij}}$
- computes $\boldsymbol{S}_{ij} := \mathsf{Enc}_h(s_{ij}, r)$, $\pi_{ij} := \mathsf{NIZK}\{(\mathsf{prms}), (\theta_{ij}, r) : \boldsymbol{S}_{ij} = \mathsf{Enc}_h(s_{ij}, r)\}$
- appends $\boldsymbol{S}_{ij}$ to the BB

Voter i registers with each EA_j

- receives $\boldsymbol{S}'_{ij} := \mathsf{ReEnc}_h(\boldsymbol{S}_{ij})$ along with a designated verifier proof of correct reencryption $d_{ij} := \delta_{\mathsf{ReEnc}} = \mathcal{DVP}\{(\mathsf{prms}, \mathsf{pk}_{\mathsf{RA}}, \mathsf{pk}_{\mathsf{V}_i}, \boldsymbol{S}'_{ij}, \boldsymbol{S}_{ij}), (\mathsf{sk}_{\mathsf{EA}_j}), \boldsymbol{S}'_{ij} = \mathsf{ReEnc}_h(\boldsymbol{S}_{ij})\}$
- verifies the proof
- computes $\boldsymbol{C}_{i1} := \prod_j \boldsymbol{S}'_{ij} = \mathsf{Enc}_h(\prod_j s_{ij})$

FIGURE 5.3: The $\mathsf{VS}_{\mathsf{PACBS}}$.Register protocol executed through an untappable channel (CIVITAS version) between V_i and EA_j

In case, the system of [CH11] is used, when the voter invokes fakekey a password is requested. Under coercion the user provides a panic password. The function ϕ recognizes the case and does not reject the password. However, it encodes it to a different element θ' of $\mathbb{Z}_q$ thus producing a different credential.

Algorithm 5.4: The fakekey functionality for evading coercion assuming [CH11]

Input : A password pwd'_i from a panic password system
Output: encrypted credential

while pwd'_i *is invalid* **do**
  $\mathsf{pwd}'_i := \mathsf{RequestPassword}()$
end
$\theta'_i := \phi(\mathsf{pwd}'_i)$
$\boldsymbol{C}_{i2} := \mathsf{Enc}_h(g_0^{\theta'_i}, r_i)$, $\pi_{i2} := \mathsf{NIZK}\{(\mathsf{prms}), (\theta'_i, r_i) : \boldsymbol{C}_{i2} = \mathsf{Enc}_h(g_0^{\theta'_i}, r_i)\}$
return $(\boldsymbol{C}_{i2}, \pi_{i2})$

The proof π_{i2} proves knowledge of a lifted ElGamal plaintext (cf. π_m in Figure 2.5).

In case, the system of [CCM08] is used the voter must have at least one trusted registration server, whose credential he can fake. Subsequently, she will also fake the corresponding designated verifier proof. The process in Algorithm 5.5 can be generalized for many trusted EA_j:

The proof π_{ij} proves knowledge of a lifted ElGamal plaintext (cf. π_m in Figure 2.5).

Algorithm 5.5: The fakekey functionality for evading coercion assuming [CH11]

Input :
Credential shares obtained during registration $s_{ij}, \boldsymbol{S}_{ij}, \boldsymbol{S}'_{ij}$
Designated verifier proofs obtained during registration d_{ij}
Identity of a least one trusted RA: j^*

Output: encrypted credential

$\theta_{ij^*} \leftarrow\$ \mathbb{Z}_q$
$s_{ij^*} := g^{\theta_{ij^*}}$
$\boldsymbol{S}_{ij^*} := \mathsf{Enc}_h(s_{ij^*}, r_{ij^*})$, $\pi_{ij^*} = \mathsf{NIZK}\{(\mathsf{prms}), (s_{ij^*}, r_{ij^*}) : \boldsymbol{S}_{ij^*} := \mathsf{Enc}_h(\theta_{ij^*}, r_{ij})\}$
$\boldsymbol{S}'_{ij^*} := \mathsf{ReEnc}_h(\boldsymbol{S}_{ij^*})$
$d_{ij^*} := \delta_{\mathsf{ReEnc}} = \mathcal{DVP}\{(\mathsf{prms}, \mathsf{pk}_{\mathsf{RA}}, \mathsf{pk}_{\mathsf{V}_i}, \boldsymbol{S}'_{ij^*}, \boldsymbol{S}_{ij^*}), (\mathsf{sk}_{V_i}), \boldsymbol{S}'_{ij^*} = \mathsf{ReEnc}_h(\boldsymbol{S}_{ij^*})\}$
$\boldsymbol{C}_{i2} := \prod_j \boldsymbol{S}'_{ij}$
return $\boldsymbol{C}_{i2}, \{d_{ij}\}_j$

During the essential moment of privacy, the voter will not use this fakekey but instead will use her real credential created during the registration phase.

$\mathsf{VS}_{\mathsf{PACBS}}$.Vote functionality

The voter selects the preferred candidate `vt` from CS, encrypts her selection with the RA's public key to produce $\boldsymbol{v}$ and provides proof of knowledge and validity of plaintext π_v. Since the candidates are group elements (cf. Algorithm 5.2), the voter proves that $\boldsymbol{v}$ is an encryption of one of these elements. Such a proof can be constructed use a disjunction of proofs of correct ElGamal encryption of a known message (section 2.4.1).

Then she presents a credential θ', that might be the original θ or the result of fakekey if the voter is under coercion. This is encrypted with the RA's public key to produce $\boldsymbol{C}_2$ along with a proof of knowledge of plaintext $\pi_{\boldsymbol{C}_2}$. This proof is very important in the analysis of the verifiability of VS.PACBS, as an attacker cooperating with a corrupted RA could include a reencryption of the credential in the voter roll.

Finally, the voter and the RA invoke the OSPACBS.Sign protocol. During the protocol execution, the RA makes some additional checks before continuing in issuing the blind signature. In particular

- The RA checks the validity of the proof of knowledge issued by the voter.
- The RA, using PETs, examines whether there was another message with the same credential which would mean a duplicate authorization request. The RA states whether the request is a duplicate or not and if not continues into issuing the blind signature.

Common input: $\mathsf{prms}, \mathsf{pk}_{\mathsf{RA}}, i, \mathsf{BB}_{\text{election}}$
V_i's private input: $\mathtt{vt}_i$
RA's private input: $\mathsf{sk}_{\mathsf{RA}}$
Assumption: Runs over an anonymous channel

V_i prepares the ballot and creates the authorization request:

- $(v_i, \pi_{v_i}) := (\mathsf{Enc}_h(\mathtt{vt}_i; r_{\mathtt{vt}_i}), \mathsf{NIZK}\{(g_1, h, \mathsf{CS}, v), (\mathtt{vt}_i, r_{\mathtt{vt}_i}) : \mathtt{vt}_i \in \mathsf{CS} \text{ AND } v_i = \mathsf{Enc}_h(\mathtt{vt}_i, r_{\mathtt{vt}_i})\})$
- If the voter *is under coercion* executes the fakekey functionality $(\boldsymbol{C}_{i2}, \pi_{C_{i2}}) := \mathsf{fakekey}()$
- If the voter *is not under coercion* computes a new encryption of her normal credential $(\boldsymbol{C}_{i2}, \pi_{C_{i2}}) := (\mathsf{Enc}_h(g_0^{\theta_i}, r_i), \mathsf{NIZK}\{(\mathsf{prms}), (\theta_i, r_i) : \boldsymbol{C}_{i2} = \mathsf{Enc}_h(g_0^{\theta_i}, r_i)\}$

RA and V_i invoke the OSPACBS.Sign protocol with communication through the BB:

1. V_i retrieves $\boldsymbol{C}_{i1}$ from $\mathsf{BB}_{\text{election}}$
2. V_i executes the OSPACBS.Blind algorithm

$$(e_i, u_{i1}, u_{i2}, d_i) := \mathsf{OSPACBS.Blind}(\mathsf{prms}, \mathsf{pk}_{\mathsf{RA}}, \boldsymbol{C}_{i1}, \boldsymbol{C}_{i2}, v_i)$$

$$\mathsf{BB}_{\text{vote}} \Leftarrow (i, e_i, \boldsymbol{C}_{i2}, \pi_{C_{i2}})$$

   where e_i is the public output and (u_{i1}, u_{i2}, d_i) are the blinding values selected from V_i
3. The RA validates the authorization request and provides the blind signature:
   - Firstly, it retrieves $\boldsymbol{C}_{i1}$ from $\mathsf{BB}_{\text{election}}$
   - If $\pi_{C_{i2}}$ is invalid the RA ignores the request.
   - The RA checks for duplicate requests with the same credential $\boldsymbol{C}_{i2}$ and proofs $\pi_{\text{i,dup1}}, \pi_{\text{i,dup2}}$ are added in a special section BB_{dup} (cf. Algorithm 5.10).

$$\mathsf{BB}_{\text{dup}} \Leftarrow (i, e_i, \boldsymbol{C}_{i2}, g_0^{\alpha_i \theta_i}, \pi_{\text{i,dup1}}, \pi_{\text{i,dup2}})$$

   - If no duplicates are found then the RA executes the OSPACBS.BlindSign to produce the blind signature:

$$(\overline{\beta}_i, \pi_{i,\mathsf{Sign}}) := \mathsf{OSPACBS.BlindSign}(\mathsf{prms}, \mathsf{sk}_{\mathsf{RA}}, \boldsymbol{C}_{i1}, \boldsymbol{C}_{i2}, e_i)$$

   - The tuple $(\boldsymbol{C}_{i1}, \boldsymbol{C}_{i2}, \overline{\beta}_i, \pi_{i,\mathsf{Sign}})$ is appended to $\mathsf{BB}_{\text{issued}}$ along with the duplicate proofs (cf. Algorithm 5.10).

$$\mathsf{BB}_{\text{issued}} \Leftarrow (i, e_i, \boldsymbol{C}_{i1}, \boldsymbol{C}_{i2}, \overline{\beta}_i, \pi_{i,\mathsf{Sign}}, g_0^{\alpha\theta_i}, \pi_{\text{i,dup1}}, \pi_{\text{i,dup2}})$$

4. V_i unblinds the signature: $\overline{\sigma}_i := \mathsf{OSPACBS.Unblind}(\mathsf{prms}, \mathsf{pk}_{\mathsf{RA}}, e_i, \overline{\beta}_i, (u_{i1}, u_{i2}, d_i))$
5. V_i prepares the ballot: $\mathrm{b}_i := (v_i, \pi_{v_i}, \overline{\sigma}_i)$

FIGURE 5.4: $\mathsf{VS}_{\mathsf{PACBS}}$.Vote Vote authorization protocol using OSPACBS.Sign

$\mathsf{VS}_{\mathsf{PACBS}}$.Cast functionality

The voter V_i constructs her ballot using the encrypted vote, the proof of knowledge of the selection and the conditional blind signature from the Authorize protocol. Then she appends her authorized ballot b_i and the corresponding proof to the BB. We refer to this part of the bulletin board as $\mathsf{BB}_{\text{cast}}$.

Algorithm 5.6: Vote casting $\mathsf{VS}_{\mathsf{PACBS}}$.Cast

Input : b_i
Output: $\mathsf{BB}_{\text{cast}}$
Assumption: Runs over an anonymous channel

$\mathsf{BB}_{\text{cast}} \Leftarrow (\mathsf{b}_i, \mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Valid}(\mathsf{b}_i, \mathsf{BB}_{\text{cast}}))$

For the ballot to be accepted, it must be of the correct format, which means that it must consist of 3 elements. The ciphertext should be unique to defend against ballot-copying attacks. Additionally, the second element must be a valid proof. The signature cannot be checked at this stage since the BB lacks the verification key. The ballot is marked with the result of the validity test and appended to the $\mathsf{BB}_{\text{cast}}$.

Algorithm 5.7: $\mathsf{VS}_{\mathsf{PACBS}}$.Valid Vote validation from BB

Input : $\mathsf{b}_i = (\boldsymbol{v}_i, \pi_{v_i}, \overline{\sigma}_i), \mathsf{BB}_{\text{cast}}$
Output: $\{0, 1\}$

foreach $\mathsf{b}_l \in \mathsf{BB}_{\text{cast}}$ **do**
  if $\mathsf{b}_l \neq (\boldsymbol{v}, \pi_v, \overline{\sigma})$ **then**
    `// Incorrect format`
    return 0
  end
  if $\boldsymbol{v}_i$ *is not unique* OR π_{v_i} *is invalid* **then**
    return 0
  end
end
return 1

Implementing the anonymous channel

[JCJ05] dictates that a necessary condition for coercion resistance is the existence of an anonymous channel to defend against a forced abstention attack. In VS.PACBS we require such a channel in the authorization To implement such a channel, a system such as Tor [DMS04] might be used. Alternatively, voters, third parties and other interested authorities might cast 'chaff' votes on behalf of registered voters in an effort to increase the anonymity set of vote casting identities.

This is implemented with the chaffvote functionality in Algorithm 5.8.

Algorithm 5.8: The chaffvote functionality to implement an anonymous channel

Input : $\mathsf{CS}, \mathsf{V}_{\mathtt{El}}$
Output: $\mathsf{BB}_{\text{vote}}, \mathsf{BB}_{\text{cast}}$

$\mathsf{V}'_{\mathtt{El}} \leftarrow_{\$} 2^{\mathsf{V}_{\mathtt{El}}}$
foreach $i \in \mathsf{V}'_{\mathtt{El}}$ **do**
  $\{t_{i1}, t_{i2}\} \leftarrow_{\$} [T_{\text{start}}, T_{\text{end}}]$
end
foreach $i \in \mathsf{V}'_{\mathtt{El}}$ **do**
  Wait until t_{i1}
  $\theta_i \leftarrow_{\$} \mathbb{Z}_q$
  $\mathtt{vt}_i \leftarrow_{\$} \mathsf{CS}$
  $\mathsf{b}_i = (\boldsymbol{v}_i, \pi_{\boldsymbol{v}_i}, \overline{\sigma}_i) := \mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Vote}\langle \mathsf{RA}(\mathsf{sk}_{\mathsf{RA}}), \mathsf{V}_i(\mathtt{vt}_i, \theta_i), \cdot \rangle$
  Wait until t_{i2}
  $\mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Cast}(\mathsf{b}_i)$
end

Every interested party that wants to participate in the anonymous channel, sends a random subset of voters. Fake votes will be sent on their behalf identified by a randomly sampled credential and a randomly sampled vote. Then it executes the $\mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Vote}, \mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Cast}$ protocols to update the BB.

Ballots generated by chaffvote, will receive an invalid signature, as the probability that the credential selected, matches a valid credential for a voter is $\frac{n}{q}$, which is negligible in the security parameter.

Removal of duplicate authorization requests

The [JCJ05] coercion resistance scheme does not preclude that multiple ballots are cast with the same credential. As a result, before tallying begins *one ballot per credential* must be kept. This phase is of quadratic complexity in [JCJ05] and as we saw in section 4.4.2 there are many variations trying to improve efficiency.

In our scheme, the duplicate removal functionality is moved to the authorization phase (along with fake credential detection and marking). Since the ballots are not yet cast, the RA must remove duplicate authorization requests. The simplest way to detect them would involve the use of the PET primitive on the credential (cf. subsection 2.6.2). In more detail, if there are k requests for a particular voter i, the RA computes $\{\mathsf{PET}\ (\boldsymbol{C}_{i2}, \boldsymbol{C}_{j2})\}_{j=1}^{k}$. If a duplicate credential exists on the j^* request, then this will be found and can be presented as proof. In other words: $\pi_{\text{i,dup}} := \mathsf{PET}\ (\boldsymbol{C}_{i2}, \boldsymbol{C}_{j^*2})$. If no duplicate is found, then the RA can post all the checks performed i.e. $\{\pi_{\text{i,non-dup}} = \mathsf{PET}\ (\boldsymbol{C}_{i2}, \boldsymbol{C}_{j2})\}_{j=1}^{k}$). This procedure, however, is quadratic in the number of authorization requests, which means that we forego all the efficiency gains in the literature.

To our advantage, the method of [Smi05; WAB07] is safe to use during duplicate detection since the tagging attack of by [AFT07] does not apply as remarked by [Spy+12]. Indeed, if an attacker tries to tag a credential and check using the tag if this is later discarded, both credentials will pass duplicate detection. As a result, we can apply their method here: All encryptions of credentials $\boldsymbol{C}_{i2}$ are blinded using a common random factor α known to all members of the RA. Then they are decrypted to obtain $g_0^{\alpha\theta_i}$, which is sent to a hashtable. If there exists another copy the request is marked as duplicate. The members of the RA generate proofs of correct blinding $\pi_{\text{i,dup1}}$ (π_{CP} from section 2.4.1) and proof of correct decryption $\pi_{\text{i,dup2}}$ (π_{Dec} from section 2.4.1). These proofs can be checked to verify that a credential is duplicate or that it is not.

However, the duplicate removal phase, still, has a profound effect on the usability and perceived performance of the scheme. Since the ballots to be counted are unlinkable to the authorization requests (because of the blindness of OSPACBS), duplicate removal cannot be performed during tallying. If this were the case, we could apply any rule to decide which of the duplicate ballots to keep. If we were to do this during authorization, it would mean that the duplicated removal phase would have to be executed after *all* authorization requests have been submitted and before the signatures are generated. This would hurt usability, since each voter would have to wait until all other voters had cast their authorization requests (typically until a time limit imposed by the RA was reached). To avoid this situation, we sacrifice this flexibility and pick the regular (first-come, first-served) definition of duplicate ballots: If a ballot contains the same credential as a previously submitted ballot, then it is considered a duplicate.

The algorithm to detect requests with duplicate credentials is described in Algorithm 5.10.

After the authorization phase has ended, BB_{dup} will contain all duplicate credentials and $\mathsf{BB}_{\text{issued}}$ will contain requests with a unique credential. In order to verify this phase, a voter or interested party should recreate the hash table HT by retrieving all authorization requests *in the order they arrived*, verifying for each the proofs $\pi_{\text{i,dup1}}$, $\pi_{\text{i,dup2}}$ and checking if each blinded credential exists in HT. For each blinded credential $g_0^{\alpha\theta_i}$ in $\mathsf{BB}_{\text{issued}}$ the HT must return false, while for each $g_0^{\alpha\theta_i}$ in BB_{dup} the HT must return true and an earlier posted request must exist in $\mathsf{BB}_{\text{issued}}$. This process is better illustrated in Algorithm 5.10 which will be used in the verification phase as well.

5.2.4 Tally phase

After the voting period has expired, the TA executes the $\mathsf{VS}_{\mathsf{PACBS}}$.Tally functionality on all the ballots from $\mathsf{BB}_{\text{cast}}$. Initially, it checks all the proofs of correct candidate selection and encryption π_v and discards the ballots with invalid proofs and identical

Algorithm 5.9: Duplicate removal using hashtable

Input : $\mathsf{BB}_{\text{election}}, \mathsf{BB}_{\text{vote}}$ cleaned up for valid proofs
cf. Figure 5.4
Output: $\mathsf{BB}_{\text{issued}}, \mathsf{BB}_{\text{dup}}$

$\mathsf{HT} := \varnothing$
$\alpha \leftarrow_{\$} \mathbb{Z}_q$
foreach $(i, e_i, \boldsymbol{C}_{i2}, \pi_{\boldsymbol{C}_{i2}}) \in \mathsf{BB}_{vote}$ **do**
  $\boldsymbol{C}'_{i2} := \boldsymbol{C}^{\alpha}_{i2}$
  $g_0^{\alpha\theta_i} = \mathsf{Dec}_z(\boldsymbol{C}'_{i2})$
  $\pi_{\text{i,dup1}} := \mathsf{NIZK}\{(\mathsf{prms}, h, \boldsymbol{C}'_{i2}, \boldsymbol{C}_{i2}), (\alpha) : \boldsymbol{C}'_{i2} = \boldsymbol{C}^{\alpha}_{i2}\}$
  $\pi_{\text{i,dup2}} := \mathsf{NIZK}\{(\mathsf{prms}, h, \boldsymbol{C}'_{i2} = (c_{i21}, c_{i22}), g_0^{\alpha\theta_i}), (z) : g^z = h \text{ AND } c_{i22} g_0^{-\alpha\theta_i} = c_{i21}^z\}$
  if $g_0^{\alpha\theta_i} \in \mathsf{HT}$ **then**
    mark request $(i, e_i, \boldsymbol{C}_{i2}, \pi_{\boldsymbol{C}_{i2}})$ *as duplicate*
    $\mathsf{BB}_{dup} \Leftarrow (i, e_i, \boldsymbol{C}_{i2}, g_0^{\alpha\theta_i}, \pi_{i,dup1}, \pi_{i,dup2})$
  else
    Execute OSPACBS.BlindSign *to produce the blind signature* $(\overline{\beta}_i, \pi_{i,\mathsf{Sign}})$
    $\mathsf{HT} := \mathsf{HT} \cup g_0^{\alpha\theta_i}$
    $\mathsf{BB}_{issued} \Leftarrow (i, e_i, \boldsymbol{C}_{i1}, \boldsymbol{C}_{i2}, \overline{\beta}_i, \pi_{i,\mathsf{Sign}}, g_0^{\alpha\theta_i}, \pi_{i,dup1}, \pi_{i,dup2})$
  end
end

Algorithm 5.10: Verification of duplicate removal

Input : $\mathsf{BB}_{\text{issued}}, \mathsf{BB}_{\text{dup}}$
Output: $\{0, 1\}$
Sort authorization requests $(i, e_i, \boldsymbol{C}_{i2}, \pi_{\boldsymbol{C}_{i2}})$ by time of arrival
$\mathsf{HT} := \varnothing$
foreach $(i, e_i, \boldsymbol{C}_{i2}, \pi_{\boldsymbol{C}_{i2}}) \in \mathsf{BB}_{\mathsf{Vote}}$ **do**
  if $\pi_{i,dup1}$ *OR* $\pi_{i,dup2}$ *do not verify* **then**
    return 0
  end
  if $(i, e_i, \cdot, g_0^{\alpha\theta_i}, \cdot) \in \mathsf{BB}_{issued}$ AND $g_0^{\alpha\theta_i} \in \mathsf{HT}$ **then**
    return 0
  end
  if $(i, e_i, \cdot, g_0^{\alpha\theta_i}, \cdot) \in \mathsf{BB}_{dup}$ AND $g_0^{\alpha\theta_i} \notin \mathsf{HT}$ AND
  $(i, e_i, \cdot, g_0^{\alpha\theta_i}, \cdot) \notin \mathsf{BB}_{issued})$ **then**
    return 0
  end
end
return 1

contents. This action can be verified by any interested third party. Then the TA must act as the verifier in PACBS and count only the votes with a valid signature. For this reason, it will execute the PACBS.Verify algorithm of Algorithm 3.13 and write the result to the BB for verifiability. However, this will signal to the coercer that their vote will not be counted. In order to avoid this, the TA transforms the ballots before posting the results by sending them through a verifiable Shuffle . This means that the PACBS.Verify algorithm cannot be used as is, but must be split into a preparation step PACBS.Verify.Prepare (Algorithm 5.11) and a decryption step PACBS.Verify.Dec (Algorithm 5.12). In between the ballots are anonymized. The encrypted votes corresponding to ballots with valid signatures are transferred to the $\mathsf{BB}_{\text{final}}$ section of BB that contains only the votes that should be counted. Subsequently the votes can be decrypted and counted or homomorphically combined. The former method supports more elaborate voting rules, but both are supported. This is denoted by the Count functionality, which computes the tally $\mathtt{T}$ and returns it together with a proof of correct computation $\pi_{\mathtt{T}}$. For instance, if Count decrypts the votes then $\pi_{\mathtt{T}}$ will contain the plaintext of each vote along with proof of correct decryption.

Algorithm 5.11: PACBS.Verify preparation - PACBS.Verify.Prepare

Input : $\mathsf{prms}, \mathsf{pk}_{\mathsf{PACBS}}, \mathsf{sk}_{\mathsf{PACBS}}, m, \overline{\sigma} = (x^*, e^*, \overline{\sigma}_1, \sigma_2)$
Output: $\boldsymbol{R}, \pi_{\boldsymbol{R}}$

if $\mathsf{H}_2(m, x^*) \neq e^*$ **then**
  return $\perp$
end
$\gamma \leftarrow_{\$} \mathbb{Z}_q$
validity $:= x^* \cdot g_2^{-\sigma_2} \cdot v^{-e^*}$
$\boldsymbol{M} := \mathsf{Enc}_h(\text{validity}; r_1)$
$\boldsymbol{V} := \boldsymbol{M}^s$
$\boldsymbol{R} := \left(\frac{\boldsymbol{V}}{\overline{\sigma}_1}\right)^{\gamma}$
$\pi_1 \leftarrow \mathsf{NIZK}\{(h_1, h, \boldsymbol{M}, \text{validity}), (r_1) : \boldsymbol{M} = \mathsf{Enc}_h(\text{validity}; r_1)\}$
$\pi_2 \leftarrow \mathsf{NIZK}\{(\boldsymbol{V}, \boldsymbol{M}), (s) : \boldsymbol{V} = \boldsymbol{M}^s\}$
$\pi_3 \leftarrow \mathsf{NIZK}\{(\boldsymbol{V}, \overline{\boldsymbol{\sigma}}_{\boldsymbol{1}}, \boldsymbol{R}), (\gamma) : \boldsymbol{R} = \left(\frac{\boldsymbol{V}}{\overline{\sigma}_1}\right)^{\gamma}\}$
$\pi_R := (\boldsymbol{M}, \boldsymbol{V}, \boldsymbol{R}, \pi_1, \pi_2, \pi_3)$
return $(\boldsymbol{R}, \pi_{\boldsymbol{R}})$

5.2.5 Verification phase

The verification phase consists of invoking the functionalities $\mathsf{VS}_{\mathsf{PACBS}}$.VerifyBallot and $\mathsf{VS}_{\mathsf{PACBS}}$.Verify to check for individual and universal verifiability.

Algorithm 5.12: PACBS.Verify decryption - PACBS.Verify.Dec

Input : $\mathsf{prms}, \mathsf{pk}_{\mathsf{Enc}}, \mathsf{sk}_{\mathsf{Enc}}, R$
Output: $d \in \{0,1\}, \pi_{\mathsf{Dec}}$

result := $\mathsf{Dec}_z(\boldsymbol{R})$
$\pi_{\mathsf{Verify}} \leftarrow \mathsf{NIZK}\{(h_1, h, \text{result}, \boldsymbol{R}), (z) : \text{result} = \mathsf{Dec}_z(\boldsymbol{R})\}$
$d \leftarrow$ result
return $(d, \pi_{\mathsf{Verify}})$

Algorithm 5.13: $\mathsf{VS}_{\mathsf{PACBS}}$.Tally functionality

Input : $\mathsf{prms}, \mathsf{pk}_{\mathsf{TA}}, \mathsf{sk}_{\mathsf{PACBS}}, \mathsf{sk}_{\mathsf{Enc}}, \mathsf{BB}_{\text{cast}}$
Output: $\mathtt{T}, \pi_{\mathtt{T}}$

`/* Keep unique valid ballots */`
foreach $\mathsf{b}_i = (\boldsymbol{v}_i, \pi_{\boldsymbol{v}_i}, \overline{\sigma}_i) \in \mathsf{BB}_{\text{cast}}$ **do**
  if *$\boldsymbol{v}_i$ is unique and $\pi_{\boldsymbol{v},i}$ is valid* **then**
    $\mathsf{BB}_{\text{correct}} \Leftarrow (\boldsymbol{v}_i, \overline{\sigma}_i)$
  end
end
`/* For each correct ballot execute the first part of Verify */`
foreach $(\boldsymbol{v}_i, \overline{\sigma}_i) \in \mathsf{BB}_{\text{correct}}$ **do**
  $(\boldsymbol{R}_i, \pi_{\boldsymbol{R}_i}) := \mathsf{PACBS.Verify.Prepare}(\mathsf{prms}, \mathsf{pk}_{\mathsf{PACBS}}, \mathsf{sk}_{\mathsf{PACBS}}, \boldsymbol{v}_i, \overline{\sigma}_i)$
  $\mathsf{BB}_{\text{prepare}} \Leftarrow (\boldsymbol{v}_i, \boldsymbol{R}_i, \pi_{\boldsymbol{R}_i})$
  $\mathsf{BB}_{\text{unshuffled}} \Leftarrow (\boldsymbol{v}_i, \boldsymbol{R}_i)$ `/* Remove the proof */`
end
`/* Shuffle list */`
$(\mathsf{BB}_{\mathsf{Shuffle}}, \pi_{\mathsf{Shuffle}}) := \mathsf{Shuffle}\ (\mathsf{BB}_{\text{unshuffled}})$
`/* Execute second part of verify (i.e. decryption) */`
foreach $(\boldsymbol{v}'_l, \boldsymbol{R}'_l) \in \mathsf{BB}_{\text{shuffle}}$ **do**
  $(d_l, \pi_{l,\mathsf{Verify}}) := \mathsf{PACBS.Verify.Dec}(\mathsf{prms}, \mathsf{pk}_{\mathsf{Enc}}, \mathsf{sk}_{\mathsf{Enc}}, \boldsymbol{R}'_l)$
  $\mathsf{BB}_{\text{result}} \Leftarrow (\boldsymbol{v}'_l, d_l, \pi_{l,\mathsf{Verify}})$
  if $d_l = 1$ **then**
    $\mathsf{BB}_{\text{final}} \Leftarrow \boldsymbol{v}'_l$
  end
end
$(\mathtt{T}, \pi_{\mathtt{T}}) := \mathsf{Count}(\mathsf{BB}_{\text{final}})$

Individual Verifiability. The algorithm for $\mathsf{VS}_{\mathsf{PACBS}}$.VerifyBallot is specified in Algorithm 5.14. The receipt retrieved for the voting protocol consists of the randomness $r_i, r_{\mathtt{vt}_i}$ used in the encryption of the credential and the vote preference respectively. Each voter has to check if her ballot b_i appears in the $\mathsf{BB}_{\text{cast}}$ list and if both the credential and the vote have been correctly encrypted. Additionally, each voter must check if the authorization request has been correctly marked as a duplicate, by verifying if a similar request has been added to $\mathsf{BB}_{\text{issued}}$ and the duplicate proofs $\pi_{\text{dup1}}, \pi_{\text{dup2}}$. Finally, the voter must invoke the PACBS.AuditSign functionality, to check that the RA has considered the credentials for the creation of the blind signature.

These types of individual verifiability checks do not violate coercion resistance as while b_i might appear on the list only the voter knows if it will be counted or not and this cannot be transferred to the coercer.

Algorithm 5.14: VerifyBallot Algorithm

Input : $\mathsf{prms}, \mathsf{pk}_{\mathsf{RA}}, \mathsf{BB}, i, (r_i, r_{\mathtt{vt}_i}), (\mathsf{b}_i, e_i), (\mathtt{vt}_i, \theta_i)$
Output: $\{0, 1\}$

```
/* Parse ballot                                                    */
```
$(v_i, \pi_{v_i}, \overline{\sigma}_i) = \mathsf{b}_i$
if $\mathsf{b}_i \neq \mathsf{Enc}_h(\mathtt{vt}_i, r_{\mathtt{vt}_i})$ **then**
  return 0
end
if $\mathsf{b}_i \notin \mathsf{BB}_{cast}$ **then**
  return 0
end
```
/* Locate and verify authorization request                         */
```
if $(i, e_i, \cdot, \cdot) \notin \mathsf{BB}_{vote}$ **then**
  return 0
else
  retrieve $\boldsymbol{C}_{i2}, \pi_{\boldsymbol{C}_{i2}}$
end
if $\boldsymbol{C}_{i2} \neq \mathsf{Enc}_h(g^{\theta_i}, r_i)$ **then**
  return 0
end
if $\pi_{\boldsymbol{C}_{i2}}$ *does not verify* **then**
  return 0
end
if $e_i \neq \mathsf{OSPACBS.Blind}(\mathsf{prms}, \boldsymbol{C}_{i1}, \boldsymbol{C}_{i2}, v_i)$ **then**
  return 0
end
```
/* Verify duplicate removal using Algorithm 5.10                   */
```
if *duplicate removal does not verify* **then**
  return 0
end
return 1

Universal Verifiability. Every interested party must be able to verify that all the votes present in $\mathsf{BB}_{\mathsf{cast}}$ will be counted. In our scheme this can be performed in an indirect manner (similarly to individual verifiability), as the vote will be counted only if the signature is valid. Everybody can verify the proofs created by the EA during all phases of the scheme, which means that the EA followed the protocol, without disclosing which votes were counted. This will only be known to the interested voter.

The verification algorithm is presented in Algorithm 5.15. For a more clear presentation we split it Auth-Verify and Tally-Verify, both of which must be valid for the result to be accepted.

Algorithm 5.15: Verify Algorithm

Input : $\mathsf{prms}, \mathsf{pk}_{\mathsf{RA}}, \mathsf{BB}$
Output: $\{0, 1\}$

if $\mathsf{Auth\text{-}Verify}(\mathsf{prms}, \mathsf{pk}_{\mathsf{RA}}, \mathsf{BB}) = 1 \wedge \mathsf{Tally\text{-}Verify}(\mathsf{prms}, \mathsf{pk}_{\mathsf{RA}}, \mathsf{BB}) = 1$ **then**
  return 1
else
  return 0
end

Algorithm 5.16 Auth-Verify verifies the actions of the RA during the authorization phase. Firstly, it checks that invalid (i.e. with an incorrectly encrypted credential) authorization requests were not signed. Then it proceeds to verify that all valid authorization requests were either characterized as unique or duplicate. For the duplicate requests it checks the proofs and that there was another request in $\mathsf{BB}_{\mathsf{issued}}$ to justify this characterization. For the unique requests Auth-Verify also invokes the PACBS.AuditSign functionality of PACBS to check that the authorization signature was computed based on the actual encrypted credentials and not arbitrarily.

Algorithm 5.17 Tally-Verify verifies the actions of the TA during the tally phase. At first, it checks that only the ballots with correct proofs are collected for tallying, by essentially repeating the $\mathsf{VS}_{\mathsf{PACBS}}$.Valid functionality. This reduces the required trust to the BB. Then it checks that all the valid ballots have been inputted correctly to the Shuffle functionality and that PACBS.Verify.Prepare has been correctly performed for each one, by verifying the proof $\pi_{\boldsymbol{R}}$. Then the shuffle proof π_{Shuffle} is checked, thus verifying that the anonymization was done according the protocol. Moreover, the proof π_{Dec} to check that the shuffle outputs were correctly decrypted, thus concluding the PACBS.AuditVrfy functionality of PACBS. Finally, Tally-Verify checks that only the ballots were the signature was valid (i.e. the ones were $d = 1$) are present in $\mathsf{BB}_{\mathsf{final}}$ and recomputes the result of Count from the ones with the correct format. These steps are dependent on the actual counting method used (i.e. if homomorphic

Algorithm 5.16: Verification of authorization phase Auth-Verify

Input : $\mathsf{prms}, \mathsf{pk}_{\mathsf{RA}}, \mathsf{BB}_{\mathrm{vote}}, \mathsf{BB}_{\mathrm{dup}}, \mathsf{BB}_{\mathrm{issued}}$
Output: $\{0,1\}$

```
/* Check that invalid authorization requests were not answered     */
```
foreach $(i, e_i, \boldsymbol{C}_{i2}, \pi_{\boldsymbol{C}_{i2}}) \in \mathsf{BB}_{vote}$ **do**
  if $\pi_{\boldsymbol{C}_{i2}}$ *is invalid* $(i, e_i, \cdot) \in \mathsf{BB}_{dup} \cup \mathsf{BB}_{issued}$ **then**
    return 0
  else
    `/* Check that were all valid request were handled */`
    if $(i, e_i, \cdot) \notin \mathsf{BB}_{dup} \cup \mathsf{BB}_{issued}$ **then**
      return 0
    end
  end
end
if $|\{(i, e_i, \boldsymbol{C}_{i2}, \pi_{\boldsymbol{C}_{i2}}) \in \mathsf{BB}_{vote} : \pi_{\boldsymbol{C}_{i2}}$ *is valid* $\}| \neq |\mathsf{BB}_{issued}| + |\mathsf{BB}_{dup}|$ **then**
  return 0
end
`/* Verify duplicate removal using Algorithm 5.10 */`
if *duplicate removal does not verify* **then**
  return 0
end
`/* Check that non-duplicate authorization requests are signed */`
foreach $(i, e_i, \boldsymbol{C}_{i1}, \boldsymbol{C}_{i2}, \overline{\beta}_i, \pi_{i,\mathsf{Sign}}, g_0^{\alpha\theta_i}, \pi_{i,dup1}, \pi_{i,dup2}) \in \mathsf{BB}_{issued}$ **do**
  if $\boldsymbol{C}_{i1} \notin \mathsf{BB}_{\mathsf{Register}}$ **then**
    return 0
  end
  if $\mathsf{PACBS.AuditSign}(\mathsf{prms}, \mathsf{pk}_{\mathsf{PACBS}}, \boldsymbol{C}_{i1}, \boldsymbol{C}_{i2}, e_i, \overline{\beta}_i, \pi_{i,\mathsf{Sign}}) = 0$ **then**
    return 0
  end
end
return 1

counting or decrypted ballots). A check of the lengths of all the outputted lists is also performed.

Algorithm 5.17: Verification of tally phase Tally-Verify

Input : $\mathsf{prms}, \mathsf{pk}_{\mathsf{TA}}, \mathsf{BB}$
Output: $\{0,1\}$

```
/* filter out invalid ballots                                    */
```
foreach $(v_i, \pi_{v_i}, \overline{\sigma}_i) \in \mathsf{BB}_{cast}$ *not marked as duplicate* **do**
  if $\pi_{v,i}$ *is valid* **then**
    check that $(v_i, \overline{\sigma}_i) \in \mathsf{BB}_{\text{correct}}$
  else
    check that $(v_i, \overline{\sigma}_i) \notin \mathsf{BB}_{\text{correct}}$
  end
end
foreach $(v, R_i, \pi_{R_i}) \in \mathsf{BB}_{prepare}$ **do**
  check that there exists the corresponding entry in $\mathsf{BB}_{\text{correct}}$
  verify π_{R_i}
end
check that $\mathsf{BB}_{\text{unshuffled}}$ is the same as $\mathsf{BB}_{\text{prepare}}$ without the proofs
verify π_{Shuffle}
foreach $(v'_l, d_l, \pi_{l,\mathsf{Verify}}) \in \mathsf{BB}_{result}$ **do**
  check that that there exists the corresponding entry in $\mathsf{BB}_{\mathsf{Shuffle}}$
  verify $\pi_{l,\mathsf{Verify}}$
end
foreach $v'_l \in \mathsf{BB}_{final}$ **do**
  check that there exists the corresponding entry $(v'_l, d_l, \pi_{l,\mathsf{Verify}})$ in $\mathsf{BB}_{\text{result}}$ and $d_l = 1$
end
check if $|\mathsf{BB}_{\text{correct}}| = |\mathsf{BB}_{\text{prepare}}| = |\mathsf{BB}_{\text{unshuffled}}| = |\mathsf{BB}_{\text{shuffled}}| = |\mathsf{BB}_{\text{result}}|$
if *all verifications are successful* **then**
  return 1
else
  return 0
end

5.2.6 Performance

We now compute the performance of our scheme in an election with n voters. We assume that the $\mathsf{EA} = (\mathsf{RA}, \mathsf{TA})$ is split into t members and we measure the number of modular exponentiations. We cover the worst-case scenario, where the ballot construction proof has a linear number of exponentiations to create or verify in the number m of candidates. This proof can be improved to a logarithmic number of steps. The performance of PACBS.Vote is dependent on the performance of PACBS from Table 3.1.

In the registration phase, the cost for each voter is 6 exponentiations, 3 for the credential encryption and 3 for the proof, while the RA performs 4 exponentiations per voter.

In the voting phase, each voter performs $12t + 4(m-1) + 17$ exponentiations to create the ballot, provide the proof of correct construction, initiate the PACBS.Sign protocol and audit π_{Sign}. To sign a single authorization request the RA performs $(12t + 10)$ exponentiations and $9t|\mathsf{BB}_{\mathsf{Vote}}|$ to check for duplicates.

In the tally phase the TA performs in total $|\mathsf{BB}_{\mathsf{Cast}}|(9t + 4m + 6)$ exponentiations, to verify the ballots, verify the duplicate proofs and execute PACBS.Verify. Some extra exponentiations will be performed by the Shuffle functionality that depend on the algorithm used.

For individual verifiability, each voter performs $8t + 1$ exponentiations for her own ballot and $8t|\mathsf{BB}_{\mathsf{Vote}}|$ to verify for duplicate elimination. Universal verifiability requires $(18 + 12t)|\mathsf{BB}_{\mathsf{Vote}}| + (8t + 4m + 1)|\mathsf{BB}_{\mathsf{Cast}}|$ exponentiations.

We now compare the performance of relevant functionalities from VS.PACBS with the schemes of CIVITAS [CCM08] and Selection [UH12]. We use the data of [UH12]. Note that the proof steps for Selections' ballot correctness need to be scaled by m to take into account all the candidates. We don't consider the soundness of the registration phase for simplicity.

Phase	Entity	Civitas	Selections	VS.PACBS
Registration	Voter	11	6	6
	RA	7	4	4
Casting	Voter	10	$(2\eta + 9)$	$12t + 4(m - 1) + 17$
	RA	-	-	$\|\mathsf{BB}_{\mathsf{Cast}}\|(12t + 10) + 9\|\mathsf{BB}_{\mathsf{Vote}}\|$
Tally	TA	$4\|\mathsf{BB}_{\mathsf{Vote}}\| + \mathcal{O}(t\|\mathsf{BB}_{\mathsf{Cast}}\|^2)$	$20\|\mathsf{BB}_{\mathsf{Vote}}\| + (16t + 8)\|\mathsf{BB}_{\mathsf{Cast}}\|$	$\|\mathsf{BB}_{\mathsf{Cast}}\|(9t + 4m + 6)$
Verify	TA	$4\|\mathsf{BB}_{\mathsf{Vote}}\| + \mathcal{O}(t\|\mathsf{BB}_{\mathsf{Cast}}\|^2))$	$20\|\mathsf{BB}_{\mathsf{Vote}}\| + (16t + 10)\|\mathsf{BB}_{\mathsf{Cast}}\|$	$(18 + 12t)\|\mathsf{BB}_{\mathsf{Vote}}\| + (8t + 4m + 1)\|\mathsf{BB}_{\mathsf{Cast}}\|$

Table 5.1: Performance comparison of our scheme with Civitas and Selections

Clearly, our scheme outperforms CIVITAS since the tallying phase is linear. VS.PACBS is more demanding for the authorities the Selections, but in the same order of magnitude. Clearly, the existence of the RA in the authorization phase is a bottleneck for our scheme. However, our scheme can be more efficient for the honest voter. In Selections the honest voter must create the anonymity set of size η, which can be computation-intensive for large values. In our case this is taken care of by the chaff votes, cast by third parties.

5.3 Security analysis

We analyze the PACBS voting protocol for verifiability, privacy, and coercion resistance.

5.3.1 Verifiability

Verifiability protects voters against a corrupted EA - in our case both against a fully corrupted RA and TA. In our protocol, this is intuitively achieved, because the EA is forced to provide evidence in the form of NIZK PoK. Everybody can audit these proofs. Their soundness proves that the EA did not deviate from the protocol. Combined with the private knowledge that the valid credential was used during authorization; each voter can be convinced that her vote was taken into account. More importantly, this private knowledge cannot be transferred to a coercer, who is confined to a public view.

The PACBS primitive was designed to implement this intuition. The OSPACBS.Sign and the OSPACBS.AuditSign functionality force the RA to follow the protocol and embed the predicate in a correct way inside the signature. Furthermore, the proofs generated by the OSPACBS.Verify functionality can make sure that the signature was indeed taken into consideration, when deciding which votes to count. PACBS is augmented with a verifiable shuffle and verifiable duplicate removal, to cover the rest of the protocol.

As a result, our scheme provides individual and universal verifiability and private eligibility verifiability. The details are provided in the following sections.

Individual Verifiability The voter V_i receives a 'receipt' that consists of the randomness $r_{\mathtt{vt}_i}$ used to encrypt the vote and r_i used to encrypt the credential. The latter can be used to check if the authorization request is in $\mathsf{BB}_{\text{vote}}$ and the former to check if the ballot is located in $\mathsf{BB}_{\text{cast}}$. Contrary to other voting systems such as Helios, the existence of the ballot in $\mathsf{BB}_{\text{cast}}$ does not mean that the vote will be counted, since this depends on the validity of the attached signature. As a result, this receipt alone does

not immediately provide individual verifiability (and as a result, cannot be used for vote selling or coercion as we will see in subsection 5.3.4). Combined however with the mental knowledge of whether the correct credential was used during the authorization request, this receipt will allow the voter to be convinced that a particular vote will be counted.

To reason about individual verifiability we need to examine two statements:

Firstly, that our voting protocol protects against clash attacks (cf. subsection 4.3.1), i.e. that each voter can uniquely pinpoint her ballot in the BB. We stress again, that this would suffice for systems like Helios, where all ballots in the BB are counted. In our case, the voter must also be convinced that the ballot will be counted. To do this, it suffices to show that the authorization request was not marked as duplicate and that the signature is valid.

These statements are proved in Theorem 5.4.

Theorem 5.1: PACBS voting is individually verifiable

Assuming that:

- The registration phase is sound.
- The BB is honest as it does not drop or inject or alter the order of submitted authorization requests.
- The voters contribute to the generation of the random coins for the encryption of the ballot.
- The PACB signature is publicly auditable.

The PACBS voting scheme provides individual verifiability according to Definition 4.2.

Proof. First, we adapt the game of Algorithm 4.2 for our scheme in the game in Algorithm 5.18. The adversary generates the cryptographic parameters and the election candidates. The challenger randomly selects credentials by skipping the ϕ function of Figure 5.2 and performs the registration phase against the $\mathcal{A}$ who then decides which voters to corrupt and selects two honest voters and two candidates. The challenger executes the vote authorization and casts the resulting ballots. The adversary wins if he can perform a clash attack or if it can render the resulting ballot invalid so as not to be counted.

Note that since the authorization request will be disassociated from the final ballot that will be sent into the tallying phase, we directly refer to the output of the $\mathsf{VS}_{\mathsf{PACBS}}$.Vote functionality.

Algorithm 5.18: $\mathsf{IndVer}^{\mathsf{int}}_{\mathcal{A},\mathsf{VS}_{\mathsf{PACBS}}}$ adapted for VS.PACBS

Input : security parameter λ
Output: $\{0,1\}$

$(\mathsf{prms}, \mathsf{pk}_{\mathsf{PACBS}}, \mathsf{sk}_{\mathsf{PACBS}}, \mathsf{sk}_{\mathsf{Enc}}, \mathsf{pk}_{\mathsf{Enc}}, \mathsf{CS}) \leftarrow \mathcal{A}(1^{\lambda})$
$\left\{\theta_i \leftarrow_{\$} \mathbb{Z}_q\right\}_{i=1}^{n}$
if $\exists (i,j) : \theta_i = \theta_j$ AND $i \neq j$ **then**
  return 0
end
$\left\{((\boldsymbol{C}_{i1}, \pi_{i1}), \theta_i) \leftarrow \mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Register}\langle \mathcal{A}(\mathsf{sk}_{\mathsf{PACBS}}), \mathsf{V}_i(\theta_i), \cdot \rangle\right\}_{i=1}^{n}$
$\mathsf{VR} := \{(i, \boldsymbol{C}_{i1})\}_{i=1}^{n}$
$V_{\mathtt{Corr}} \leftarrow \mathcal{A}(\textbf{corrupt})$
$(\mathtt{vt}_0, \mathtt{vt}_1, i, j) \leftarrow \mathcal{A}()$
if $i, j \in V_{\mathtt{Corr}}$ OR $i = j$ **then**
  return 0
end
$\mathsf{b}_i = (\boldsymbol{v_{0}}_i, \pi_{\boldsymbol{i},\boldsymbol{v_0}}, \overline{\sigma}_i) := \mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Vote}\langle \mathcal{A}(\mathsf{sk}_{\mathsf{RA}}), \mathsf{V}_i(\mathtt{vt}_0, \theta_i), \cdot \rangle$
$\mathsf{b}_j = (\boldsymbol{v_{1}}_j, \pi_{\boldsymbol{j},\boldsymbol{v_1}}, \overline{\sigma}_j) := \mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Vote}\langle \mathcal{A}(\mathsf{sk}_{\mathsf{RA}}), \mathsf{V}_j(\mathtt{vt}_1, \theta_j), \cdot \rangle()$
if $(\mathsf{b}_i = \mathsf{b}_j$ AND $\mathsf{b}_i \neq \perp)$ OR $\mathsf{b}_i = \perp$ **then**
  return 1
else
  return 0
end

Firstly, regarding defense against clash attacks, since the credentials are not created solely by the RA but the voter takes part as well, the RA cannot assign the same credential to two distinct voters. As a result, $\mathsf{b}_i = \mathsf{b}_j$. would mean that $\boldsymbol{v_{0}}_i = \boldsymbol{v_{1}}_j$, $\pi_{\boldsymbol{i},\boldsymbol{v_0}} = \pi_{\boldsymbol{j},\boldsymbol{v_1}}$, $\overline{\sigma}_i = \overline{\sigma}_j$.

$$\boldsymbol{v_{0}}_i = \boldsymbol{v_{1}}_j \Rightarrow (g^{r_{\mathtt{vt}_0}}, \mathtt{vt}_0 \cdot h^{r_{\mathtt{vt}_0}}) = (g^{r_{\mathtt{vt}_1}}, \mathtt{vt}_1 \cdot h^{r_{\mathtt{vt}_1}}) \Rightarrow r_{\mathtt{vt}_0} = r_{\mathtt{vt}_1} \text{ AND } \mathtt{vt}_0 = \mathtt{vt}_1$$

The assumption that the random coins $r_{\mathtt{vt}_0}, r_{\mathtt{vt}_1}$ are honestly generated, because of the contribution of randomness by the voters, makes the probability of the event that $r_{\mathtt{vt}_0} = r_{\mathtt{vt}_1}$ equal to $\epsilon_{\mathsf{Enc}} = \frac{1}{q}$.

Assume now that $\mathcal{A}$ manages, as a corrupted RA, to achieve that $\mathsf{b}_i = \perp$, i.e. invalidate the ballot b_i so that it will not be counted. This attack can be performed in two ways:

- By retrieving the voter credential $\boldsymbol{C}_{i1}$ from VR, decrypting it to retrieve $g_0^{\theta_i}$ and posting an authorization request for the particular voter. To do this, the $\mathcal{A}$ must either fake the proof of knowledge of θ_i, $\pi_{\boldsymbol{C}_{i2}}$, or extract it from $g_0^{\theta_i}$. This is achieved with probability $\epsilon_{\mathsf{NIZK}} + \epsilon_{\mathrm{DL}}$.
- By characterizing the corresponding authorization request as duplicate by posting a duplicate request $(i, e_i, \boldsymbol{C}_{i2}, g_0^{\alpha\theta_i}, \pi_{\mathrm{i,dup1}}, \pi_{\mathrm{i,dup2}})$ in $\mathsf{BB}_{\mathrm{dup}}$. While this can

be achieved with probability ϵ_{NIZK}, it won't pass verification as the BB is assumed honest and the HT is reconstructed by the voters.

– Alternatively, the $\mathcal{A}$ must provide an invalid $\overline{\sigma}$, even V_i supplied the correct credential. This can be achieved with probability $\epsilon_{\mathsf{PACBS}_{\text{audit}}}$.

As a result, the probability of the adversary succeeding in the game in Algorithm 5.18 is $\epsilon_{\mathrm{DL}} + \epsilon_{\mathsf{Enc}} + \epsilon_{\mathsf{PACBS}_{\text{audit}}} + \epsilon_{\mathsf{NIZK}}$ which under the assumption is negligible in the security parameter, a contradiction.

■

Universal Verifiability Our scheme is universally verifiable assuming a corrupted RA and a corrupted TA but an honest BB. Consequently, VS.PACBS satisfies the notion of strong verifiability of Algorithm 4.6. However, our construction is different from the one presented in [Cor+14] since the voters do not sign their votes themselves, but receive signatures from the RA. As a result, the techniques presented there cannot be used and strong universally verifiability will be proved directly.

Theorem 5.2: PACBS voting is universally verifiable

Assuming that:

- The L provided to SetupElection is authentic - contains only real voters.
- The BB is honest as it does not drop or inject or alter the order of submitted authorization requests.
- The Shuffle functionality is verifiable.
- The NIZK system used for the proofs is sound.
- The PACB signature is publicly auditable and unforgeable.

The PACBS voting scheme provides universal verifiability according to Definition 4.3.

Proof. First, we adapt the game of Algorithm 4.7, that is used for the definition of strong universal verifiability when the RA is corrupted and the BB is honest for our scheme. In the game in Algorithm 4.7, the $\mathcal{A}$ has access to 3 oracles, to control corrupted voters **Corrupt** and engage in the protocols for voting and casting. The RA is under the control of the adversary. In [Cor+14], where this game originates, the role of the RA is different: It solely generates and distributes the credentials on its own and takes no further action. In our case in Algorithm 5.19, however, the RA takes part in voting through the authorization phase. The winning condition is the same, however.

Algorithm 5.19: $\mathsf{UniVer}^{\mathsf{Strong}}_{\mathcal{A},\mathsf{VS}}$ with malicious RA adapted for VS.PACBS

Input : security parameter λ
Output: $\{0,1\}$

Oracle Corrupt(i)
| $V_{\mathtt{Corr}} \Leftarrow (i, \theta_i)$

Oracle Vote$(i, \mathtt{vt})$
| $\mathsf{b}_i := \mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Vote}\langle \mathcal{A}(\mathsf{sk}_{\mathsf{RA}}), \mathsf{V}_i(\mathtt{vt}, \theta_i), \cdot \rangle$

Oracle Cast(i, b_i)
| $BB_{\text{cast}} \Leftarrow \mathsf{b}_i$

$(\mathsf{prms}, \mathsf{pk}_{\mathsf{PACBS}}, \mathsf{sk}_{\mathsf{PACBS}}, \mathsf{pk}_{\mathsf{Enc}}, \mathsf{sk}_{\mathsf{Enc}}, \mathsf{CS}) \leftarrow \mathcal{A}(1^\lambda)$
$\{\theta_i \leftarrow_\$ \mathbb{Z}_q\}_{i=1}^n$
if $\exists (i,j) : \theta_i = \theta_j$ AND $i \neq j$ **then**
| return 0
end
$\{((\boldsymbol{C}_{i1}, \pi_{i1}), \theta_i) \leftarrow \mathsf{VS.Register}\langle \mathcal{A}(\mathsf{sk}_{\mathsf{RA}}), \mathsf{V}_i(\theta_i), \cdot \rangle\}_{i=1}^n$
$\mathsf{VR} := \{(i, \boldsymbol{C}_{i1})\}_{i=1}^n$
$V_{\mathtt{Corr}} \leftarrow \mathcal{A}^{\mathbf{Corrupt}}(\mathbf{corrupt})$
$V_{\mathtt{Hon}} := V \setminus V_{\mathtt{Corr}}$
$(\mathtt{T}_{\mathcal{A}}, \pi_{\mathtt{T}_{\mathcal{A}}}) \leftarrow \mathcal{A}^{\mathbf{Cast}}()$
if $\mathsf{VS.Verify}(\mathtt{T}_{\mathcal{A}}, \pi_{\mathtt{T}_{\mathcal{A}}}, \cdot) = 0$ OR $\mathtt{T}_{\mathcal{A}} = \perp$ **then**
| return 0
end
$\mathtt{Chck} = \{(i^{\mathtt{Chck}}, \mathtt{vt}_i^{\mathtt{Chck}}, \mathsf{b}_i^{\mathtt{Chck}})\}_{i=1}^{|\mathtt{Chck}|}$
if $\exists n_{\mathtt{Corr}} : 0 \leq n_{\mathtt{Corr}} \leq |V_{\mathtt{Corr}}|$ AND $\exists \{\mathtt{vt}_i^{\mathtt{Corr}} \in \mathsf{CS}\}_{i=1}^{n_{\mathtt{Corr}}}$
$\exists n' : 0 \leq n' \leq |\mathtt{Hon}| - |\mathtt{Chck}|$ AND $\exists \{\mathtt{vt}'_i\}_{i=1}^{n'}$ `// Honest voters that did not check`
$\mathtt{T}_{\mathcal{A}} = \mathsf{result}(\mathtt{vt}_i^{\mathtt{Corr}}) \oplus \mathsf{result}(\mathtt{vt}_i^{\mathtt{Chck}}) \oplus \mathsf{result}(\mathtt{vt}'_i)$ **then**
| return 0
else
| return 1
end

Note, as the challenger controls all the voters in $V_{\texttt{Hon}}$ (which includes $V_{\texttt{Chck}}$) it can internally maintain a list $(i, \texttt{vt}_i, r_{\texttt{vt}_i}, \theta_i, r_i)$ of their inputs to $\mathsf{VS}_{\mathsf{PACBS}}$.Vote functionality. As a result, it can produce the choices $\texttt{vt}'_i$ of the voters that did not check and therefore compute $\mathsf{result}(\texttt{vt}_i^{\texttt{Chck}}) \oplus \mathsf{result}(\texttt{vt}'_i)$ in the winning condition. Consequently, it can track their respective ballots in $\mathsf{BB}_{\text{cast}}$. The rest are ballots, that originate from $\mathcal{A}$. For the challenger to lose the game, it must be proved that $\mathcal{A}$ cannot modify or drop honest ballots and that $\mathcal{A}$ cannot add more ballots than $|V_{\texttt{Corr}}|$.

Assume, now that $\mathcal{A}$ manages to win the game in Algorithm 5.19 with non-negligible probability for some or all the following reasons:

- $\mathcal{A}$ has modified at least a ballot corresponding to the vote of an honest voter. This can happen in $\mathsf{VS}_{\mathsf{PACBS}}$.Cast and in $\mathsf{VS}_{\mathsf{PACBS}}$.Tally, as the $\mathcal{A}$ does not have access to the ballot itself in $\mathsf{VS}_{\mathsf{PACBS}}$.Vote. In $\mathsf{VS}_{\mathsf{PACBS}}$.Cast changing $\texttt{vt}$ from v in the tuple $(v, \pi_v, \overline{\sigma})$ without invalidating the proof π_v can happen with probability ϵ_{NIZK}. In $\mathsf{VS}_{\mathsf{PACBS}}$.Tally this can occur during the call to Shuffle with probability $\epsilon_{\mathsf{Shuffle}}$.
- $\mathcal{A}$ has removed at least a ballot corresponding to an honest voter. Since the BB is honest and does not drop messages, this can occur in the following ways:
  - By marking the authorization request as duplicate in $\mathsf{VS}_{\mathsf{PACBS}}$.Vote, or by explicitly issuing an invalid signature for a valid authorization request, by faking π_{Sign}. As we saw in the proof of Theorem 5.4 this can happen with probability $\epsilon_{\text{DL}} + \epsilon_{\mathsf{PACBS}_{\text{audit}}} + \epsilon_{\mathsf{NIZK}}$.
  - By disregarding a ballot during tallying despite having a valid signature. Again, this can happen with probability $\epsilon_{\mathsf{PACBS}_{\text{audit}}}$.
  - By dropping a ballot during shuffling, which can happen with probability $\epsilon_{\mathsf{Shuffle}}$.
- $\mathcal{A}$ has added at least a ballot corresponding to a duplicate of a valid one or to a user that does not exist in L.
  - In the former case, the adversary must produce fake proofs $\pi_{\text{dup1}}, \pi_{\text{dup2}}$ which can happen with probability ϵ_{NIZK}. But this attack does not pass verification.
  - In both cases, taking into account VR is public, the $\mathcal{A}$ can add the duplicated ballot only in $\mathsf{VS}_{\mathsf{PACBS}}$.Cast by forging the signature. This can happen with $\epsilon_{\mathsf{PACBS}_{\text{forge}}}$.

As a result, the probability of success of the $\mathcal{A}$ in the game in Algorithm 5.19 is $\epsilon_{\mathrm{DL}} + \epsilon_{\mathsf{NIZK}} + \epsilon_{\mathsf{Shuffle}} + \epsilon_{\mathsf{PACBS}_{\mathrm{audit}}} + \epsilon_{\mathsf{PACBS}_{\mathrm{forge}}}$ which is negligible according to the assumptions, leading to a contradiction. ■

Eligibility Verifiability VS.PACBS does not provide eligibility verifiability. As we saw in subsection 4.3.3, eligibility verifiability requires that anyone can verify that each tallied ballot was cast by a voter with the right to vote and that no voter cast more than two counted ballots. While the second part of this statements does indeed hold for our scheme as we saw in proof of Theorem 5.2 the first part is incompatible with coercion resistance. In VS.PACBS voters are identified by their credential. This means that if a voter is eligible, her ballot can be identified by a valid credential. This cannot be revealed to the public, as it would also notify the coercer. This is compatible with the analysis in section 4.7.

However, our scheme does provide *private* eligibility verifiability, where only the interested voter learns if her ballot was eligible. The reasoning is similar to Theorem 5.4.

5.3.2 Ballot secrecy

The PACBS voting scheme we defined in section 5.2 provides ballot secrecy. Importantly, this result holds under the assumption that the TA is corrupted and the use of the anonymous channel. Intuitively, this is achieved due to the blindness of the PACBS scheme. During the authorization request, where voter identifying information is present, the encrypted vote is blinded, which means that the ballot does not leak anything. During the voting phase the vote in the ballot is unblinded (thus merely encrypted). However, the ballot is unlinkable to the signing session.

After the casting phase, the BB contents are the same as Helios, except for the unblinded RA signature. Our protocol also incorporated defenses against Helios-related attacks, such as attacks against ballot independence of [CS13] because of the ballot weeding that occurs in the $\mathsf{VS}_{\mathsf{PACBS}}$.Valid (Algorithm 5.7) and $\mathsf{VS}_{\mathsf{PACBS}}$.Tally (Algorithm 5.13) functionalities. Additionally, all the NIZKPoK contain the full statement in the random oracle call, thus thwarting the attacks of [BPW12; CS13].

The ballot secrecy of our scheme is analyzed in Theorem 5.3 according to a modified BPRIV definition.

Theorem 5.3: PACBS voting is private

Assuming that:

- The PACB signature is blind.
- The PACB signature is publicly auditable.
- The BB is honest in that it accepts all entries.
- There is an anonymous channel during casting.

The PACBS voting scheme provides privacy according to U-BPRIV even against a corrupted TA.

Proof. Recall from section 4.5 that BPRIV essentially states that all the voting data released in the BB do not provide the privacy adversary any advantage in guessing the preference of an honest voter concerning what is provided by the election tally alone. Furthermore, it must be noted that in BPRIV it is assumed that there are 2 bulletin boards $\mathsf{BB}_0, \mathsf{BB}_1$. When the $\mathcal{A}$ casts a ballot, it is posted in both. However, an honest voter can cast a different vote $\mathtt{vt}_0$ for BB_0 and a different vote $\mathtt{vt}_1$ for BB_1. Both $\mathtt{vt}_0, \mathtt{vt}_1$ are selected by the $\mathcal{A}$.

We deviate from BPRIV in two ways: The first concerns the tallying phase. Since BPRIV considers the TA trusted, the tallying is performed by $\mathcal{C}$ always on BB_0 and a proof (or a simulation) is provided. In our case, the adversary performs the tallying on $\mathsf{BB}_\mathfrak{b}$. As a result, there is no need to define a BPRIV simulator nor a **Tally** oracle. Note that the same concept is expressed in U-BPRIV Algorithm 4.11, but since the RA in our case does not generate the keys on its own, we build on the definition from Algorithm 5.20 for simplicity. Secondly, in order to indicate the anonymous channel, the votes chosen by $\mathcal{A}$ are not sent to the specified BB, but their destination is chosen randomly, by a coin flipped by $\mathcal{C}$.

We begin by adjusting the description of the oracles of Algorithm 4.10 for our case. Note, that to avoid confusion on the names of the oracles and the respective functionalities, when we use the **Cast** oracle we mean that the adversary executes the complete procedure to cast a ballot and not only the $\mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Cast}$ algorithm. Furthermore, in our case, the ballot mentioned in the BPRIV definition contains all interactions of the voter with the EA: both the authorization request and the actual casting. In particular, the ballot of the BPRIV definition contains what the voter posts in $\mathsf{BB}_{\mathrm{vote}}, \mathsf{BB}_{\mathrm{cast}}$.

Since we don't want the adversary to trivially win the game, all the sets of inputs $\{\mathtt{vt}_0\}, \{\mathtt{vt}_1\}$ to the **Vote** oracle are assumed to be equal as multi-sets, or equivalently that they are permutations of each other. Since the adversary controls the tally, if it were not for this assumption then the challenger would not be able to swap the

Algorithm 5.20: Privacy definition for PACBS voting

Input : security parameter λ
Output: $\{0,1\}$

Oracle Vote$(i, \mathtt{vt}_0, \mathtt{vt}_1)$
- $b \leftarrow_\$ \{0,1\}$
- $(\perp, (i, e_b, \boldsymbol{C}_2, \pi_{\boldsymbol{C}_2})) := \mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Vote}\langle \mathcal{A}(\mathsf{sk}_{\mathsf{RA}}), \mathsf{V}_i(\mathtt{vt}_0, \theta_i), \cdot \rangle$
- $(\perp, (i, e_{1-b}, \boldsymbol{C}_2, \pi_{\boldsymbol{C}_2})) := \mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Vote}\langle \mathcal{A}(\mathsf{sk}_{\mathsf{RA}}), \mathsf{V}_i(\mathtt{vt}_1, \theta_i), \cdot \rangle$
- $\mathsf{BB}_{b,\mathrm{vote}} \Leftarrow (i, e_b, \boldsymbol{C}_2, \pi_{\boldsymbol{C}_2})$
- $\mathsf{BB}_{1-b,\mathrm{vote}} \Leftarrow (i, e_{1-b}, \boldsymbol{C}_2, \pi_{\boldsymbol{C}_2})$
- **if** $\mathsf{Valid}((\boldsymbol{v}_b, \pi_{b,\boldsymbol{v}}, \overline{\sigma}_b), \mathsf{BB}_b)$ AND $\mathsf{Valid}((\boldsymbol{v}_{1-b}, \pi_{1-b,\boldsymbol{v}}, \overline{\sigma}), \mathsf{BB}_{1-b})$ **then**
  - $\mathsf{BB}_{b,\mathrm{cast}} \Leftarrow (\boldsymbol{v}_b, \pi_{b,\boldsymbol{v}}, \overline{\sigma}_b)$
  - $\mathsf{BB}_{1-b,\mathrm{cast}} \Leftarrow (\boldsymbol{v}_{1-b}, \pi_{1-b,\boldsymbol{v}}, \overline{\sigma}_{1-b})$
- **else**
  - return $\perp$

Oracle Cast(i, b_i)
- `/* Parse ballot contents */`
- $((i, e_i, \boldsymbol{C}_{i2}, \pi_{\boldsymbol{C}_{i2}}), (\boldsymbol{v}_i, \pi_{\boldsymbol{v}_i}, \overline{\sigma}_i)) = \mathrm{b}_i$
- $\mathsf{BB}_{\beta,\mathrm{vote}} \Leftarrow (i, e_i, \boldsymbol{C}_{i2}, \pi_{\boldsymbol{C}_{i2}})$ for $\beta \in \{0,1\}$
- **if** $\mathsf{Valid}((\boldsymbol{v}_i, \pi_{\boldsymbol{v}_i}, \overline{\sigma}_i), \mathsf{BB}_\beta)$ *for* $\beta \in \{0,1\}$ **then**
  - $\mathsf{BB}_{b,\mathrm{cast}} \Leftarrow (\boldsymbol{v}_i, \pi_{\boldsymbol{v}_i}, \overline{\sigma}_i)$ for $\beta \in \{0,1\}$
- **else**
  - return $\perp$

$(\mathsf{prms}, \mathsf{pk}_{\mathsf{RA}}, \mathsf{sk}_{\mathsf{RA}}, \mathsf{pk}_{\mathsf{TA}}, \mathsf{sk}_{\mathsf{TA}}) \leftarrow \mathcal{A}(1^\lambda)$
$(\mathsf{V}_{\mathtt{El}}, \mathsf{CS}) \leftarrow \mathcal{A}()$
$\mathfrak{b}' \leftarrow \mathcal{A}^{\mathbf{Vote},\mathbf{Cast}}(\mathsf{prms}, \mathsf{pk}, \mathsf{BB}_{\mathfrak{b}})$
return $\mathfrak{b} = \mathfrak{b}'$

votes between bulleting boards, since the tally would change, and the adversary could trivially distinguish it. Additionally, we also assume that the authorization requests for $\{\mathtt{vt}_0\}, \{\mathtt{vt}_1\}$ are issued in random order by the challenger.

We define a sequence of games beginning from the adversary interacting with the challenger of BPRIV^0 and concluding to the adversary interacting with the challenger of BPRIV^1. The differences in each game are detected by the adversary with negligible probability.

- $Game_0$ is the BPRIV^0 game. Both bulletin boards are built through a series of calls to oracles **Vote**, **Cast**. This means that, for each tuple (i, b) posted by the challenger, where $\mathsf{b} = ((e, \boldsymbol{C}_2, \pi_{\boldsymbol{C}_2}),\ (\boldsymbol{v}, \pi_{\boldsymbol{v}}, \overline{\sigma}))$, it can internally maintain the tuple $(i, \mathsf{b}_0, \mathtt{vt}_0, \mathsf{b}_1, \mathtt{vt}_1)$ where b_0 is the ballot for $\mathtt{vt}_0$ in BB_0 and b_1 is the ballot for $\mathtt{vt}_1$ in BB_1. We observe that each ballot contains two parts i.e. $\mathsf{b}_i = (\mathsf{b}_i[0], \mathsf{b}_i[1])$ where $\mathsf{b}[0] = (e, \boldsymbol{C}_2, \pi_{\boldsymbol{C}_2})$ and $\mathsf{b}[1] = (\boldsymbol{v}, \pi_{\boldsymbol{v}}, \overline{\sigma})$.
- $\{Game_1^i\}_{i\in[n]}$. For each honest i, the challenger replaces the entry in the BB, to be tallied, with an entry with the same plaintext vote from the other BB. More formally, it swaps $(i, \mathsf{b}_\mathfrak{b}[1])$ in $BB_\mathfrak{b}$ with $(i', \mathsf{b}_{1-\mathfrak{b}}[1])$ from $BB_{1-\mathfrak{b}}$ by looking up an entry in the internal table of tuples it maintains from the calls to oracle **Vote**, such that $\mathtt{vt}_{i,\mathfrak{b}} = \mathtt{vt}_{i',1-\mathfrak{b}}$, where i' is another honest voter.

  This change is indistinguishable from the adversarial point of view, for the following reasons:

  * Since $\mathtt{vt}_{i,\mathfrak{b}} = \mathtt{vt}_{i',1-\mathfrak{b}}$ the tally does not change.
  * The signature for i' contained in $\mathsf{b}_{i'}[1]$ in is perfectly unlinkable to the authorization request for $\mathsf{b}_i[0]$.
  * The values $\boldsymbol{v}, \pi_{\boldsymbol{v}}$ are indistinguishable, or else the $\mathsf{Enc} + \mathsf{PoK}$ encryption scheme would not satisfy the NM-CPA property, which is not the case [BPW12].
  * Since both voters i, i' are honest, both signatures are valid assuming an honest RA. However, since the RA is adversarial, it could try to distinguish the two BB by providing a valid ballot for i in $\mathsf{BB}_\mathfrak{b}$ and an invalid ballot for i' in $\mathsf{BB}_{1-\mathfrak{b}}$, so that $\mathsf{BB}_\mathfrak{b}$ contains one less vote. The adversary cannot disregard the predicate and provide an invalid signature, as this will violate the public auditability of the PACBS scheme. Additionally, $\mathcal{A}$ could try to mark the request as duplicate, while it is not. This, however, would violate the soundness of $\pi_{\text{dup1}}, \pi_{\text{dup2}}$ in the duplicate detection phase.

It is easy to see that $Game_1^n$ is BPRIV^1. Since each game in the sequence is indistinguishable to the adversary, the initial and final games are also indistinguishable. As a result, even if an adversary fully controls the RA, it cannot distinguish between the contents of BB_b and BB_{1-b}. ■

5.3.3 Everlasting privacy

We will now analyze how our scheme fares against unbounded adversaries using the games WE-BPRIV, E-BPRIV, SE-BPRIV from Algorithm 4.12, Algorithm 4.13,Algorithm 4.14. Intuitively, VS.PACBS provides strong everlasting privacy since the ballots on the BB are information-theoretically protected and the casting phase is anonymous.

In more detail, assuming there is an anonymous channel during casting and that the adversary cannot fully control it VS.PACBS provides everlasting privacy according to SE-BPRIV from Algorithm 4.14.

The reasoning behind this statement is that $\hat{\mathcal{A}}$, the unbounded adversary of Algorithm 4.14, will not be assisted by the data in the BB. Note that in our case the complete communication transcript is posted on the BB. As a result, the notions of E-BPRIV and WE-BPRIV are essentially the same. Furthermore, in Theorem 5.3 we proved that VS.PACBS satisfies U-BPRIV. Recall, that the contents of the BB after Vote and Cast are:

$$(i, e, \boldsymbol{C}_1, \pi_{\boldsymbol{C}_1}, \boldsymbol{C}_2, \pi_{\boldsymbol{C}_2}, \overline{\beta}, \pi_{\mathsf{Sign}}, g_0^{\alpha\theta_i}, \pi_{\mathrm{dup1}}, \pi_{\mathrm{dup2}}) \in \mathsf{BB}_{\mathsf{Vote}}$$
$$(\boldsymbol{v}, \pi_{\boldsymbol{v}}, \overline{\sigma}) \in \mathsf{BB}_{\mathsf{Cast}}$$

Since $\hat{\mathcal{A}}$ can decrypt the BB contents are equivalent to:

$$(i, e, \theta_1, \theta_2, \overline{\beta}) \in \mathsf{BB}_{\mathsf{Vote}}$$
$$(\mathtt{vt}, \overline{\sigma}) \in \mathsf{BB}_{\mathsf{Cast}}$$

The blindness of PACBS does not allow $\mathcal{A}$ to associate the identified authorization request in $\mathsf{BB}_{\mathsf{Vote}}$ with the respective vote in $\mathsf{BB}_{\mathsf{Cast}}$. As a result, the BB content alone cannot put an identity to a vote.

One way $\hat{\mathcal{A}}$ can bypass this, is by *tagging* the authorization and voting requests by attaching identifying information. Timing or network data could provide for such tags. To thwart this attack the anonymous channel is used. In the spirit of subsection 4.6.3 it is essential that the adversary cannot fully control it.

5.3.4 Coercion Resistance

Our voting protocol VS.PACBS provides coercion resistance according to the framework of [JCJ05] we analyzed in subsection 4.4.2.

Recall that the JCJ framework defines coercion resistance as receipt freeness, along with resistance to impersonation, random voting, and forced abstention attacks. Concerning the former two, if a coercer forces a voter V to reveal her credential, or vote randomly with it, then she can present a fake credential. During the moment of privacy, she can cast her real vote. Nobody (including the coercer) can tell if the credential is valid or not, since the conditional verifiability of PACBS discloses this information only to the designated verifier (the TA in this case) and the holder of the secret information, i.e. V. The generated proofs used in PACBS.AuditSign and PACBS.AuditVrfy will merely provide information that the protocol was executed successfully for verifiability.

To thwart the forced abstention attack, however, a further assumption should be made, namely that there exists an anonymous channel used by V during authorization and casting. In the former case, the anonymous channel is implemented with multiple authorization requests corresponding to the same voter ID. This prevents the coercer from verifying abstention by checking if a particular ID is missing from the BB. Such an assumption is common in previous JCJ-related schemes in the literature (e.g. [CH11; Sch+11]). These extra authorization requests are assumed to originate from interested third parties (e.g. pro-democracy non-governmental organizations) or other voters, and will not be counted as they will correspond to a valid credential with negligible probability.

Of course, for all these to hold, the coercer is assumed not to (fully) corrupt the election authorities RA, TA which is one of the assumptions made in the JCJ framework anyway.

It should also be noted that our scheme is impervious to the '1009' attack of [Smi05], where the coercer can check if the voter follows his directions by forcing him to cast a particular number of votes (e.g. 1009) and check if the group size is maintained in tallying. First of all the chaffvote functionality makes sure that it is improbable that the number of the '1009' authorization requests is maintained, as there will be chaff requests for that particular voter ID. As a result, grouping with that value does not make any sense and hence $\mathcal{A}$ cannot use it. If the attacker forces the voter to cast '1009' votes with a particular credential, the voter can do so and the duplicate marking functionality will keep one and mark the rest 1008 as duplicates without any impact to the group size. Additionally, the VS.Valid functionality discards all identical ballots, so performing these attacks in the casting phase (i.e. with the ballots instead

of authorization requests) will leave one for counting and the rest will be marked as invalid without again affecting the group size. More importantly, in the tallying phase of VS.PACBS there is no particular grouping of the votes, as all credential-related information is absent and the result is hidden inside the signature, protected by the conditional verifiability property.

In order to formally analyze coercion resistance, we adapt the two games from [JCJ05] to accommodate our protocol's functionality and the authorization phase in particular. Recall that we must prove that $\mathsf{Adv}^{\mathsf{cr}}_{\mathsf{VS}_{\mathsf{PACBS}},\mathcal{A}}(\lambda)$ is negligible in the security parameter, where:

$$\mathsf{Adv}^{\mathsf{cr}}_{\mathsf{VS}_{\mathsf{PACBS}},\mathcal{A}}(\lambda) = \Pr\Big[\mathbf{Game}^{\mathsf{cr}}_{\mathsf{VS}_{\mathsf{PACBS}},\mathcal{A}}(\lambda, n, m, |V_{\mathtt{Corr}}|, \mathcal{D}) = 1\Big] - \Pr\Big[\mathbf{Game}^{\mathsf{cr\text{-}ideal}}_{\mathsf{VS}_{\mathsf{PACBS}},\mathcal{A}_{\mathsf{ideal}}}(\lambda, n, m, |V_{\mathtt{Corr}}|, \mathcal{D}) = 1\Big]$$

As always, n denotes the total number of voters, who are partitioned in three sets: $V_{\mathtt{Corr}}$ are the corrupted voters, controlled by $\mathcal{A}$. We denote their number $|V_{\mathtt{Corr}}|$. The corruption is static, which means that the adversary selects them at the beginning of the game without the capacity to change them. $V_{\mathtt{Hon}}$ are the honest voters and $|V_{\mathtt{Hon}}|$ is their number. The behavior of the honest voters is governed by a distribution $\mathcal{D}$ [2] which aims to model the uncertainty of the adversary regarding their voting behavior. For instance, $\mathcal{D}$ determines whether an honest voter will abstain or not and how many chaff ballots except her real one will be cast on her behalf. The coerced voter is denoted by j and is neither honest nor corrupt. As a result: $n = |V_{\mathtt{Corr}}| + |V_{\mathtt{Hon}}| + 1$.

The coercion resistance game

The game $\mathbf{Game}^{\mathsf{cr}}_{\mathsf{VS}_{\mathsf{PACBS}},\mathcal{A}}(\lambda, n, m, |V_{\mathtt{Corr}}|, \mathcal{D})$ is described in Algorithm 5.21. In general, it does not deviate much from the JCJ workflow, except for the particular adaptations required by our scheme.

The $\mathsf{VS}_{\mathsf{PACBS}}$ challenger sets up the election system, randomly chooses the credentials and registers all the voters. In the end of this phase the encrypted credentials are posted in the BB. Again, for readability, in Algorithm 5.21 the exact data inserted to the BB are omitted, as they are specified in detail in section 5.2.

The coercer corrupts the voters in $V_{\mathtt{Corr}}$ and obtains their credentials. Subsequently, it picks a voter j to coerce, who must not be corrupt. A random coin $\mathfrak{b}$ is flipped to determine the behavior of a coerced voter. If $\mathfrak{b} = 0$ then she invokes the fakekey functionality to create a fake credential to present to the coercer. At a different time,

[2] We simplify notation by using $\mathcal{D}$ instead of $\mathcal{D}_{n,m}$

during the moment of privacy, she casts her real vote. If $\mathfrak{b} = 1$ the voter gives her real credential to the coercer, without utilizing the moment of privacy.

Afterward, all the honest voters vote, and $\mathcal{A}$ casts ballots on behalf of the corrupted voters and the coerced voter, in a manner that benefits him the most. Then the challenger executes the Tally functionality and computes the result $\mathtt{T}$ as well as a proof of correct computation. Finally, the adversary tries to guess if voter j followed his instructions or not. If he succeeds, he wins the game and 1 is returned from the experiment.

Note that if $\mathfrak{b} = 0$, there exists one more ballot than if $\mathfrak{b} = 1$. The reason for this is that in the former case there are two ballots for the coerced voter (one for the real and one for the fake credential), while in the latter there is only the fake ballot. This distinguishing factor is counterweighed by the distribution $\mathcal{D}$ that governs the behavior of honest voters.

Moreover, $\mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Tally}$ also produces a *side-channel* that consists of evidence with the role to check the correct operation as specified in Algorithm 5.13 and what is deduced by them. These include the number of unique valid ballots, the number of duplicate ballots, or the number of ballots with invalid proofs, the list of anonymized ballots, the proofs from PACBS.Verify.Prepare and PACBS.Verify.Dec which might provide extra information to $\mathcal{A}$, such as the number of ballots that contain invalid credentials. While these are already posted in the BB, we treat them in a special way as they can aid $\mathcal{A}$ in computing $\mathfrak{b}'$ and winning the experiment. As a result, we reserve a special argument Γ for them.

The ideal coercion resistance game

As we saw in section 4.4, a simplistic definition of coercion resistance would simply request from the adversary to distinguish the coerced voter's behavior for the different values of $\mathfrak{b}$ in Algorithm 5.21. However, this behavior can be deduced from the tally (e.g. in the case that the candidate preferred by the coercer receives no ballots). For this reason, as in [JCJ05] we must define an ideal version of the experiment of Algorithm 5.21 to express the maximal advantage that an adversary can obtain from the invariant characteristics of the voting system, such as the tally, without interacting with it. As a result, the ideal version, described in Algorithm 5.22 serves as a baseline for comparison.

In more detail, the differences of Algorithm 5.21 and Algorithm 5.22 are:

- The adversary is *not given* the credentials of corrupt voters. Consequently, he does not cast ballots for the corrupt voters, but merely pre-selects their votes

Algorithm 5.21: Real Coercion resistance game $\mathbf{Game}^{\mathsf{cr}}_{\mathsf{VS}_{\mathsf{PACBS}},\mathcal{A}}(\lambda, n, m, |V_{\mathtt{Corr}}|, \mathcal{D})$

$(\mathsf{prms}, \mathsf{pk}_{\mathsf{PACBS}}, \mathsf{sk}_{\mathsf{PACBS}}, \mathsf{pk}_{\mathsf{Enc}}, \mathsf{sk}_{\mathsf{Enc}}) := \mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Setup}(1^{\lambda})$
$\{\theta_i \leftarrow_{\$} \mathbb{Z}_q\}_{i=1}^{n}$
if $\exists (i,j) : \theta_i = \theta_j$ AND $i \neq j$ **then**
  return 0
end
$\{((\boldsymbol{C}_{i1}, \pi_{i1}), \theta_i) \leftarrow \mathsf{VS}.\mathsf{Register}\langle \mathsf{RA}(\mathsf{sk}_{\mathsf{RA}}), \mathsf{V}_i(\theta_i), \cdot\rangle\}_{i=1}^{n}$
$(\mathsf{V}_{\mathtt{El}}, \mathsf{CS}) := \mathsf{VS}.\mathsf{SetupElection}(n, m, \mathsf{prms}, \mathsf{L})$
$(V_{\mathtt{Corr}}, V_{\mathtt{Hon}}) \leftarrow \mathcal{A}(n, \textbf{corrupt})$
$\{\theta_i\}_{i=1}^{|V_{\mathtt{Corr}}|} \leftarrow \mathcal{A}(|V_{\mathtt{Corr}}|, \textbf{obtain})$
$(j, \mathtt{vt}_j) \leftarrow \mathcal{A}(\mathsf{V}_{\mathtt{El}}, \textbf{coerce})$
if $j \in \mathtt{Corr}$ AND $\mathtt{vt}_j \notin \mathsf{CS}$ **then**
  return $\perp$
end
$\mathfrak{b} \leftarrow_{\$} \{0,1\}$
if $\mathfrak{b} = 0$ **then**
  $\theta_j^* \leftarrow \mathsf{fakekey}(j)$ `// generate fake credential`
  `// In the moment of privacy cast the ballot for the real credential`
  $\mathsf{b}_j \Leftarrow \mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Vote}\langle \mathsf{RA}(\mathsf{sk}_{\mathsf{RA}}), \mathsf{V}_j(\mathtt{vt}_j, \theta_j), \cdot\rangle$
  $\mathsf{BB} \Leftarrow \mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Cast}(\mathsf{b}_j)$
else
  $\theta_j^* := \theta_j$ `// yield credential`
end
$\{\mathsf{b}_i \Leftarrow \mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Vote}\langle \mathsf{RA}(\mathsf{sk}_{\mathsf{RA}}), \mathsf{V}_i(\mathtt{vt}_i, \theta_i), \cdot\rangle\}_{i=1}^{|V_{\mathtt{Hon}}|, \mathcal{D}}$
$\{\mathsf{BB} \Leftarrow \mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Cast}(\mathsf{b}_i)\}_{i=1}^{|V_{\mathtt{Hon}}|, \mathcal{D}}$
$\{\mathsf{BB} \Leftarrow \mathcal{A}(\theta_i, \mathtt{vt}_i, \textbf{cast})\}_{i=1}^{|V_{\mathtt{Corr}}|}$
$\mathsf{BB} \Leftarrow \mathcal{A}(\theta_j^*, \mathtt{vt}_j, \textbf{cast})$
$(\mathtt{T}, \pi_{\mathtt{T}}, \Gamma) := \mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Tally}(\mathsf{BB}, \mathsf{sk}_{\mathsf{PACBS}}, \mathsf{sk}_{\mathsf{Enc}})$
$\mathfrak{b}' \leftarrow \mathcal{A}(\mathtt{T}, \pi_{\mathtt{T}}, \Gamma, \mathsf{BB}, \textbf{guess})$
return $\mathfrak{b} = \mathfrak{b}'$

and does not have access to the BB. The rationale for this is that in the ideal version, the adversary must not be aided by cryptographic material.

- For the same reason, the coerced voter does not use fakekey but always yields the real credential to $\mathcal{A}$.
- The functionality $\mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Vote}$ is replaced with $\mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Vote}^{\mathsf{ideal}}$ that performs the authorization part of the protocol by the RA in an ideal manner:
  - It gives only one valid signature (ballot) for each valid credential, i.e. performs duplicate ballot weeding.
  - For honest voters, it does not embed the PET of the credentials inside the signature. Instead, it extracts the underlying credentials (by decrypting them) from the authorization request. If they are equal it equips the ballot with a valid signature instead of embedding the credentials inside the signature, else it provides an invalid signature.
  - For the credential of the coerced voter the validity of the ballot is determined by the coin toss - i.e. if $\mathfrak{b} = 0$ (coercion resistance) the fake vote (which now carries the correct credential) is disregarded. Only the vote cast during the moment of privacy is counted. If $\mathfrak{b} = 1$ (coercion) the fake vote is treated normally, i.e. given a valid signature, as it contains the correct credential.
- Since the functionality $\mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Vote}^{\mathsf{ideal}}$ is the one that characterizes which votes will be counted, tally just follows the regular protocol. Unlike [JCJ05] in our case there is no need for an ideal tally.

We are now ready to prove that VS.PACBS is coercion resistant.

Theorem 5.4: PACBS voting is coercion resistant

Assuming that:
- The registration phase is sound.
- The RA, TA is honest (or that the $\mathcal{A}$ does not control the majority of their members).
- There is an anonymous channel during vote authorization and casting.
- The adversary is unaware of the exact behavior of honest voters.
- The voter has a moment of privacy.
- The PACB signature is conditionally verifiable.

The PACBS voting scheme provides coercion resistance according to the JCJ framework.

Algorithm 5.22: Ideal Coercion resistance game $\mathbf{Game}^{\mathsf{cr-ideal}}_{\mathsf{VS}_{\mathsf{PACBS}},\mathcal{A}_{\mathsf{ideal}}}(\lambda, n, m, |V_{\mathtt{Corr}}|, \mathcal{D})$

```
/* Registration proceeds as in Algorithm 5.21                    */
```
$(V_{\mathtt{Corr}}, V_{\mathtt{Hon}}) \leftarrow \mathcal{A}_{\mathsf{ideal}}(n, \textbf{corrupt})$
$\varnothing \leftarrow \mathcal{A}_{\mathsf{ideal}}(|V_{\mathtt{Corr}}|, \textbf{obtain})$ `// obtain credentials of corrupt voters`

$(j, \mathtt{vt}_j) \leftarrow \mathcal{A}_{\mathsf{ideal}}(\mathsf{V}_{\mathtt{El}}, \textbf{coerce})$
if $j \in \mathtt{Corr}$ AND $\mathsf{Vote}_j \notin \mathsf{CS}$ **then**
  return $\perp$
end
$\mathfrak{b} \leftarrow_{\$} \{0,1\}$
if $\mathfrak{b} = 0$ **then**
  $\mathsf{b}_j \Leftarrow \mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Vote}^{\mathsf{ideal}}\langle \mathsf{RA}(\mathsf{sk}_{\mathsf{RA}}), \mathsf{V}_j(\mathtt{vt}_j, \theta_j), \cdot\rangle$ `// moment of privacy: use real credential`

  $\mathsf{BB} \Leftarrow \mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Cast}(\mathsf{b}_j)$
end
$\theta^*_j := \theta_j$ `// Always yield real credential`
$\left\{\mathsf{b}_i \Leftarrow \mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Vote}^{\mathsf{ideal}}\langle \mathsf{RA}(\mathsf{sk}_{\mathsf{RA}}), \mathsf{V}_i(\mathtt{vt}_i, \theta_i), \cdot\rangle\right\}_{i=1}^{|V_{\mathtt{Hon}}|, \mathcal{D}}$
$\left\{\mathsf{BB} \Leftarrow \mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Cast}(\mathsf{b}_i)\right\}_{i=1}^{|V_{\mathtt{Hon}}|, \mathcal{D}}$
$\left\{\mathsf{BB} \Leftarrow \mathcal{A}(\_, \mathtt{vt}_i, \textbf{cast})\right\}_{i=1}^{|V_{\mathtt{Corr}}|}$
$\mathsf{BB} \Leftarrow \mathcal{A}(\theta^*_j, \mathtt{vt}_j, \textbf{cast})$
$(\mathtt{T}, \pi_{\mathtt{T}}, \Gamma) := \mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Tally}(\mathsf{BB}, \mathsf{sk}_{\mathsf{PACBS}}, \mathsf{sk}_{\mathsf{Enc}})$
$\mathfrak{b}' \leftarrow \mathcal{A}_{\mathsf{ideal}}(\mathtt{T}, \Gamma, \textbf{guess})$
return $\mathfrak{b} = \mathfrak{b}'$

Proof. In order to prove that our scheme is coercion resistant we construct a series of simulated games starting from $\mathbf{Game}^{\mathsf{cr}}_{\mathsf{VS}_{\mathsf{PACBS}},\mathcal{A}}$ and concluding to $\mathbf{Game}^{\mathsf{cr\text{-}ideal}}_{\mathsf{VS}_{\mathsf{PACBS}},\mathcal{A}}$ where the advantage of the adversary is negligible between each game. These modifications aim to show that the extra vote cast by the coerced voter and the use of the fakekey functionality if $\mathfrak{b} = 0$ as well as the behavior of the honest voters cannot essentially help the coercer to distinguish if he can win the game in a manner substantially different from the ideal case. Our detailed exposition follows the general framework of [JCJ05; UH12].

$\mathbf{Game}_0$ - Initial simulation

Initialization (Setup, Registration, SetupElection) The challenger $\mathcal{C}$ creates the parameters of the voting system, by running the PACBS.Gen algorithm. The public values $(q, \mathbb{G}, g_1, g_2, v, g)$ are selected uniformly at random and posted to the BB. The private values $s, z \in \mathbb{Z}_q$ are sampled and the corresponding public keys k, h are posted to the BB. During registration, for each voter i, $\mathcal{C}$ selects randomly a credential $\theta_i \leftarrow_\$ \mathbb{Z}_q$, stores it internally in a real credential table (i, θ_i) along with the voter identity. This means that the $\mathcal{C}$ knows the credentials for all voters (corrupt, honest, and coerced). Then all credentials are encrypted using h and rebased. In the end the BB contains encryptions of all the credentials $(i, \mathsf{Enc}_h(g_0^{\theta_i}))$. All the proofs are created normally, as $\mathcal{C}$ knows all the relevant values.

Corruption - Coercion - Coin Flip The adversary requests the credentials of the corrupt voters. The challenger reviews its internal table and retrieves them. $\mathcal{A}$ selects a voter to coerce and sends its id and the instructed vote to the $\mathcal{C}$, who checks for validity. $\mathcal{C}$ selects a random bit $\mathfrak{b} \leftarrow_\$ \{0, 1\}$. If $\mathfrak{b} = 0$ the challenger emulates the actions of the coerced voter, by casting the real vote and producing a fake credential. Since both of these actions will be modified, we describe them separately.

Coercion evasion - Real vote The challenger retrieves the real credential θ_j and computes a new encryption of $g_0^{\theta_j}$, $\boldsymbol{C}_{j2}$. It also retrieves the encryption $\boldsymbol{C}_{j1}$. Furthermore, it computes an encryption $\boldsymbol{v}_j$ of the vote $\mathtt{vt}_j$ selected by the coercer. Then $\mathsf{OSPACBS.Blind}(\boldsymbol{C}_{j1}, \boldsymbol{C}_{j2}, \boldsymbol{v}_j)$ is executed and the result e_j is posted to the BB along with the voter identity (i.e. $(j, e_j, \boldsymbol{C}_{j2}, \pi_{j2})$).

In order to answer the authorization request, he decrypts both credentials using its private key z. If there are no duplicates, it checks if the underlying credentials are the same. If this is the case, it creates a valid signature by using an encryption of 1 as $\boldsymbol{W}$, otherwise it encrypts a random element of $\mathbb{G}$. The proof $\pi_{j,\mathsf{Sign}}$ is produced

in the normal manner. Then it posts $(\boldsymbol{C}_{j1}, \boldsymbol{C}_{j2}, \overline{\beta}_j, \pi_{j,\mathsf{Sign}})$. Finally, it executes the OSPACBS.Unblind algorithm and posts the ballot $(\boldsymbol{v}_j, \pi_{v_j}, \overline{\sigma}_j)$ to the BB.

Coercion evasion - Fake credential The challenger executes fakekey and creates a fake credential θ_j that is provided to $\mathcal{A}$.

Coercion concession - Real credential If $\mathfrak{b} = 1$ the challenger retrieves the real credential for j and submits it to $\mathcal{A}$.

Honest voters vote The challenger posts the votes of the honest voters, and the same procedure that was followed for the real vote for voter j, is used to carry their authorization requests. As a result, the data that is posted on the BB are the 3 tuples:

$$((i, e_i, \boldsymbol{C}_{i2}, \pi_{i2}), (\boldsymbol{C}_{i1}, \boldsymbol{C}_{i2}, \overline{\beta}_i, \pi_{i,\mathsf{Sign}}), (\boldsymbol{v}_i, \pi_{v_i}, \overline{\sigma}_i))$$

Corrupt voters vote The adversary creates authorization requests and votes for all the corrupted voters. $\mathcal{A}$ does not necessarily follow the protocol: As a result, all types of votes (duplicates, with invalid proofs, with missing components, etc.) can be submitted.

Coerced voter votes $\mathcal{A}$ constructs a vote for the coerced voter.

Tallying The challenger filters the unique ballots and executes the PACBS.Verify functionality. Then it sends the votes with the partially decrypted signatures to an oracle Shuffle that performs shuffling. The signature in the result is then decrypted and if it is valid the corresponding vote will be sent for counting. It is easy to see, that all the votes submitted from honest voters as well as the real vote of the coerced voter will be counted in this way.

For convenience we denote by $\mathsf{Success}_i = \Pr\left[\mathbf{Game}^{\mathsf{cr}}_{i,\mathsf{VS}_{\mathsf{PACBS}},\mathcal{A}}(\lambda, n, m, |V_{\mathtt{Corr}}|, \mathcal{D}) = 1\right]$.
As a result, $\mathsf{Success}_0 = \Pr\left[\mathbf{Game}^{\mathsf{cr}}_{\mathsf{VS}_{\mathsf{PACBS}},\mathcal{A}}(\lambda, n, m, |V_{\mathtt{Corr}}|, \mathcal{D}) = 1\right]$.

$\mathbf{Game}_1$ - The real vote does not help the coercer Our first objective is to show that if $\mathfrak{b} = 0$ the vote cast during the moment of privacy for the coerced V_j does not aid the coercer. Note that this vote is present in both real (Algorithm 5.21) and ideal games (Algorithm 5.22). In the real game, this is the only vote with a valid credential, since if $\mathfrak{b} = 0$ the voter hands the result of fakekey to the adversary. In the ideal game, however, two votes with the real credential will be added if $\mathfrak{b} = 0$ (one by the voter and one by the adversary). This is taken care of, by the ideal version of the Vote

functionality which disregards the vote cast by $\mathcal{A}$. To conclude, an equal number of votes is cast in both games for this case.

To achieve our objective, we let $\mathcal{C}$ in $\mathbf{Game}_1$ choose a different credential $\hat{\theta}_j \neq \theta_j$ for V_j and cast the real vote with this one. As a result:

$$\begin{aligned} \mathsf{b}_j &\Leftarrow \mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Vote}\langle \mathsf{RA}(\mathsf{sk}_{\mathsf{RA}}), \mathsf{V}_j(\mathtt{vt}_j, \hat{\theta}_j), \cdot \rangle \\ \mathsf{BB} &\Leftarrow \mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Cast}(\mathsf{b}_j) \end{aligned}$$

where the $\mathsf{VS}_{\mathsf{PACBS}}.\mathsf{Vote}$ posts the tuples $(j, e_j, \hat{\boldsymbol{C}_{j2}}, \hat{\pi_{j2}}), (\boldsymbol{C}_{j1}, \boldsymbol{C}_{j2}, \hat{\overline{\beta}}_j, \pi_{j,\hat{\mathsf{Sign}}}), (\boldsymbol{v}_j, \pi_{\boldsymbol{v}_j}, \hat{\overline{\sigma}}_j))$to the BB. The values $j, e_j, \boldsymbol{C}_{j1}, \boldsymbol{v}_j, \pi_{\boldsymbol{v}_j}$ are the same both in $\mathbf{Game}_1$ and $\mathbf{Game}_0$ as $e, \boldsymbol{v}_j, \pi_{\boldsymbol{v}_j}$ depend only the voter choice which is the same in both games, while $\boldsymbol{C}_{j1}, \boldsymbol{v}_j$ depends on the registered credential which is the same again. As a result, $\mathcal{A}$ must use $\hat{\boldsymbol{C}_{j2}}, \hat{\pi_{j2}}, \hat{\overline{\beta}}_j, \pi_{j,\hat{\mathsf{Sign}}}, \hat{\overline{\sigma}}_j$ to distinguish between $\mathbf{Game}_1$ and $\mathbf{Game}_0$. Note that $\overline{\sigma}_j$ will be a valid signature, while $\hat{\overline{\sigma}}_j$ will be invalid, as the credentials no longer match. However, this is not distinguishable by the adversary by the conditional verifiability property of PACBS. Also, note, that the tally will not be a distinguishing fact as the $\mathcal{C}$, can monitor the choice $\mathtt{vt}_j$ of V_j during the various stages of the protocol, and add it to the final tally, despite having an invalid signature.

This means that:

$$|\mathsf{Success}_1 - \mathsf{Success}_0| \leq \epsilon_{\text{IND-CPA}} + \epsilon_{\mathsf{NIZK}} + \epsilon_{\text{CONDVER}} + \epsilon_{\text{dup}}$$

where $\epsilon_{\text{IND-CPA}}$ is the advantage of $\mathcal{A}$ to win the IND-CPA game, ϵ_{NIZK} is the probability that the system used leaks information on the witness, ϵ_{dup} is the probability that a duplicate vote has been cast for the credential $\hat{\theta}_j$ and $\epsilon_{\text{CONDVER}}$ is the probability that $\mathcal{A}$ wins the conditional verifiability game for PACBS.

$\mathbf{Game}_2$ - The fake credential does not help the coercer Continuing the case that $\mathfrak{b} = 0$, in the real game (Algorithm 5.21) $\mathcal{A}$ receives the fake credential, while in the ideal game $\mathcal{A}_{\mathsf{ideal}}$ receives the real one (Algorithm 5.22). The only way the two can be distinguished, is by using the encryption $\boldsymbol{C}_{i1}$ residing in the voter roll, as well as the relevant proof π_{i1} posted after the registration phase. The distinguishing advantage is:

$$|\mathsf{Success}_2 - \mathsf{Success}_1| \leq \epsilon_{\text{IND-CPA}} + \epsilon_{\mathsf{NIZK}}$$

Note that $\boldsymbol{C}_{i1}$ is reencrypted, which means that the voter cannot recreate it, as the randomness has changed.

$\mathbf{Game}_3$ - The ballots of the honest voters do not offer any advantage to the coercer If the $\mathcal{A}$ can determine the partial tally of the honest voters, then it can combine this fact with the votes of the corrupt voters and deduce the tally of the election minus the coerced voter. From there it is easy to see if the coercer obeyed or not. To examine this, we define a family of games $\{\mathbf{Game}_3^i\}_{i=1}^{|V_{\mathtt{Hon}}|}$ where in $\mathbf{Game}_3^i$ we change the behavior of $\mathsf{V}_i \in V_{\mathtt{Hon}}$ to use a credential $\hat{\theta}_i$ instead of θ_i. Following the same reasoning as in $\mathbf{Game}_1$, we can see that:

$$|\mathsf{Success}_3^i - \mathsf{Success}_3^{i-1}| \leq \epsilon_{\text{IND-CPA}} + \epsilon_{\mathsf{NIZK}} + \epsilon_{\text{CONDVER}} + \epsilon_{\text{dup}}$$

where $\mathsf{Success}_3^0$ is defined as $\mathsf{Success}_2$ and $\mathsf{Success}_3^{|V_{\mathtt{Hon}}|}$ is defined as $\mathsf{Success}_3$.

If $\mathcal{A}$ has advantage ϵ_{anon} to break the anonymity of the anonymous channel in the authorization and casting phase to win the forced abstention attack we get

$$|\mathsf{Success}_3 - \mathsf{Success}_0| \leq \epsilon_{\text{IND-CPA}} + \epsilon_{\mathsf{NIZK}} + \epsilon_{\text{CONDVER}} + \epsilon_{\text{dup}} + \epsilon_{\text{anon}}$$

which is negligible according to our assumptions.

Note that in $\mathbf{Game}_3^{|V_{\mathtt{Hon}}|}$ all the contents of the BB consist entirely of random values and tallying takes place by internally tracking the honest votes. As a result, $\mathcal{A}$ cannot be assisted by the cryptographic primitives but the only data he can use is the actual result. This means that $\mathbf{Game}_3^{|V_{\mathtt{Hon}}|}$ is $\mathbf{Game}^{\mathsf{cr-ideal}}$ and

$$\mathsf{Success}_3 = \Pr\left[\mathbf{Game}^{\mathsf{cr\text{-}ideal}}_{\mathsf{VS}_{\mathsf{PACBS}},\mathcal{A}_{\mathsf{ideal}}}(\lambda, n, m, |V_{\mathtt{Corr}}|, \mathcal{D}) = 1\right]$$

.

■

6 Conclusion

The end is the beginning is the end

Smashing Pumpkins

6.1 Summary

In this thesis, we presented two cryptographic primitives for the creation of a digital signature that is conditionally verifiable from a designated verifier. For the simpler variation, CBS, its validity is determined by a private input to the signer, which makes the signing process and its result not auditable. This problem is solved by PACBS where the validity, depends on publicly available but encrypted data, that is embedded inside the signature. The creation and verification functionalities must also emit non-interactive zero-knowledge proofs of knowledge, so that a corrupted entity is forced to follow the protocol. We created instantiations for these primitives and introduced security models to express and formally prove their security properties.

The introduction of CBS/PACBS was motivated by the need to implement efficient, private, coercion-resistant and verifiable electronic voting in the JCJ framework. Indeed, we created a voting protocol built around the properties of PACBS Conditional verifiability was used to provide coercion resistance, public auditability was used for universal verifiability and blindness was used for privacy against a corrupted signer. The intuition behind our protocol was to replace the public outputs of the public tests performed during tallying in the JCJ framework with private but verifiable ones. To provide efficiency the credential check was moved earlier in the protocol, during the authorization phase and its results were conveyed using the PACBS primitive. The combination of PACBS with other components assumed by the JCJ framework, like an anonymous channel, during the voting and casting phases, allowed us to provide ballot secrecy without a need to trust the talliers, an assumption common in the e-voting literature. We observed, that since this could amount to schemes that are resistant to more powerful adversaries, our voting schemes provide everlasting privacy.

To better study this phenomenon, we introduced security models for everlasting privacy. Our adversary has the strongest capabilities ever defined in the literature as he

is both active during the election by collecting data, as well as in the future where he can break the cryptographic schemes used. Based on this we defined three models of everlasting privacy. Our novel contribution was the modeling of the adversarial capabilities both in terms of computational power and in terms of information context. Using this model, we reasoned that a system based on commitments opened through private channels cannot provide the strongest sense of everlasting privacy, as an adversary with internal knowledge (such as a governmental agency) will have access to both the decommitments and network information. The use of an *independent* anonymous channel, however, will be able to thwart such an attempt. While such a channel is not currently practical, especially on a large scale, our model indicates that research for everlasting privacy will be assisted by its existence, as long as the other properties required by voting systems (e.g. integrity and election verifiability). Anonymous channels have the added benefit that they resemble the way traditional elections work and as a result, such a system will be more accessible to the voter.

While, anonymous channels are indeed a strong assumption, its use was prompted by the need to defend against the vote abstention attack, described in the JCJ framework. This also applies to other aspects of PACBS as well, as they were developed for the extremely adversarial environment of electronic voting. In an environment, with less strict security requirements, the logic behind CBS/PACBS could be applied more efficiently.

6.2 Future work

In general, the separation of the authorization and casting phases that characterize the scheme of [FOO92] can be easily applied to many scenarios. CBS/PACBS fits easily with this architecture and helps provide *certified anonymity* combined with verifiability. Apart from electronic voting, other applications of interest, could include anonymous surveys, anonymous usage of services for authorized users (e.g. adults). Such use cases have attracted attention in the previous years and there have been similar proposals [Hoh+14] that follow the [FOO92] architecture, differing on the use of primitives.

To make the use of our primitives in such usage scenarios clear, we generalize the voting use case from chapter 5. A functionality is meant to be used by a predetermined set of users, that must be authenticated before access is granted. However, the use of the service must be anonymous, concerning the identities as well as the inputs of the users. For instance, one can imagine a questionnaire about the evaluation of a university course, or the statistical processing of the financial results for a set of companies. To guarantee anonymity, the identities of the participants should be

hidden. However, this is not enough, as the answers to survey questions could leak identifying information. A possible solution could involve homomorphic encryption; however, this will limit the types of possible operations to process the data.

Our proposed protocol provides another solution. There is a publicly available list of encrypted tokens, representing the valid authentication information for the users of a service. The service provider has granted the users with the credential that corresponds to the token beforehand. Each time a user requests the use of the service, she provides an encryption of her credential and points to the token in the list for comparison. Moreover, she includes a blinded input to the service (e.g. the answers of the survey). The signer uses PACBS to sign the request, granting the service usage, if the user has the right to. PACBS hides the input to the service from the signer. From that moment on, the service can be used without any need for identity validation, as the (unblinded) signature 'carries' the authentication information. Consequently, the processing of the inputs can take place without the need for identifying information. The PACBS primitive makes all actions auditable. However, the inputs can still leak information. The fake credential mechanism can allow the users to obfuscate their submitted data, by posting misleading information accompanied with a fake credential. These inputs will be seemingly valid, without leaving any public trace of whether they were really considered. Only the user and the processor will know which items make it into the outcome.

Furthermore, this obfuscated, auditable but private credential checking mechanism can provide a solution to metadata anonymity. In many cases, the contents of exchanged messages can be protected by cryptography or other means. This does not apply, however, to other related information, such as the sender or the receiver or the frequency of exchanged messages. Such metadata have many uses, and are easier to leak, providing a rich information context. One way to protect them is to flood the set of genuine conversations with fake ones. During processing, they must be weeded out for the results to make sense. Both of our primitives can help in this direction.

More concrete avenues for future work also include:

Different constructions of PACBS could be investigated and built on top of blind signature schemes for instance the original RSA blind signatures of [Cha83] or the scheme of [Bol02]. Our aim with such constructions is to overcome the polylogarithmic bound on the number of concurrent sessions, inherent in the scheme of [Oka92] from which our proposals inherit. Such instantiations would implement the main ideas of CBS and PACBS but provide unforgeability against plain one-more forgery instead of strong one-more forgery, thus improving efficiency and scalability. Additionally, the emphasis could shift between blindness, designated verification, and

auditability to express more fine-grained trust models (e.g. a corrupted signer but an honest verifier and vice versa, or computational instead of perfect blindness). Our security model should also be extended to reason about the security of such constructions. Secondly, another idea would be to extend the secret information to more than a single bit, maybe to a complete program, the input of which is the encrypted public data. This program could be inside a signature and evaluated when the signature is verified. The design of more concrete protocols for applying PACBS in different usage scenarios is another direction for further research, following the general guidelines we described.

Regarding the e-voting part of this thesis, an implementation and application in small-scale elections can be attempted to study its usability and its acceptance from end-users. Particular attention should be given on the coercion resistance mechanisms and the use of panic passwords.

We also plan to explore how this distribution of control for anonymous channels we saw in subsection 4.6.3 can be applied to the various types of anonymous channels proposed in the literature and incorporated in known voting protocols in a usable manner. Our models can prove a useful means of establishing the success of such efforts. Guided by U-BPRIV, we also plan to further investigate the consequences of building electronic voting protocols that do not require trust in the tallier for secrecy. In our view, this is of independent interest, as in the majority of works in electronic voting the talliers are trusted for secrecy and not trusted for verifiability. While this is a common assumption in the e-voting literature, little research has been conducted on whether it is accepted by the voters. Our intuition, from recent reactions to e-voting attempts are that there is a discrepancy between formal models and voter perceiptions. However this requires further research.

6.2.1 Coercion resistance in decentralized and blockchain voting

A major other research direction is the application of PACBS to provide coercion resistance in the decentralized setting with the aim of later transferring such a protocol on a blockchain. A first such attempt was presented in [PBS20].

One of the most interesting methods made possible by remote electronic voting, *self-tallying* elections, were proposed in [KY02], where voters can conduct the elections themselves, without using or trusting tallying authorities. That initial idea received many revisions and improvements ([Gro04]), with the most efficient one being the Open Vote Network (OV-net)[HRZ10], which was implemented on top of the Ethereum [But14] blockchain in [MSH17]. However, smart contracts' limitations

restricted the number of voters to around fifty in that attempt. Recently [SGY20], scalability in the OVT was improved, at the expense of decentralization though. Instead of using smart contracts for self-tallying, [SGY20] delegates the vote-counting functionality to an untrusted authority that performs it off-chain, but provides the computation trace, so that the result can be verified by everybody. This untrusted authority does not rely on private keys and as a result, *any* entity with enough local processing power can play this role.

As far as security is concerned, decentralized voting schemes should have the following basic properties [Kha+12]: *Perfect ballot secrecy:* In order to learn a voter's choice all other participants must conspire. *Self-tallying:* All voters and interested third parties can tally the election result from published data. This property provides universal verifiability. *Dispute-freeness:* The protocol avoids situations were one party (rightly) blames another for breaking the protocol, without providing evidence to support it. This property is related to accountability [KTV10]. *Fairness:* No party can deduce partial results before the voting period has ended. *Robustness:* The voting protocol and result computation cannot be blocked by a corrupted party.

Of course, the properties of section 1.2 must also hold. However, their semantics might be different. For instance, coercion resistance in decentralized voting is not well researched. The reason is that in such protocols, especially self-tallying ones, a coercer can be present during vote counting to '*help*' its victim '*correctly*' count the votes and at the same time make sure that his attack succeeded - i.e. the coerced voter followed his instructions. In fact, [Che+10] proves that universal verifiability cannot coexist with receipt freeness - a weaker form of coercion resistance - unless private channels are available. This however leaves open what can be achieved with private or anonymous channels. A decentralised PACBS could be used to provide such a private channel and an application of ring signatures, like the one in [PS17] could provide anonymity.

The main idea of the scheme of [PBS20] can be shortly described as follows: Voters are arranged in rings and the votes of each ring are counted by a tallier who acts as the designated verifier for the ring. A sortition mechanism, like in [Gil+17] can be used to assign voters to rings and select the tallier at random. Alternatively, in the case where the protocol is executed over a Bitcoin-like blockchain [Nak08] the proof of work mechanism can be used. More specifically the participants can locally run an algorithm, until its output matches some predefined characteristics (e.g. number of zeros) of the proof of work target. Such mechanisms have the goal to deter participants from conspiring to create rings and select a designated verifier. During vote casting, the voter decides on her choice `vt` and signs it using a ring-based variation of PACBS. In particular, the vote is seen as coming from the ring as a whole. To prevent

double-voting the pseudoidentity mechanism of [PS17] can be used. If the voter is under coercion she does not use her regular private key, but a randomly selected one. In her moment of privacy, she uses her regular private key. As a result, the former signature will not count and the vote that is accompanied by it will be considered coerced and therefore not counted. Signature verification is not public, but strongly tied to a specific verifier identified by a key, who provides proofs of correct operation as in PACBS.

An implementation of such a scheme could also use an underlying blockchain as [MSH17]. In general, the fact that a blockchain like [Nak08] or [But14] shares many similarities with an e-voting BB has generated many proposals for blockchain voting. However, as analyzed in [GP19] using a blockchain as an e-voting BB solves few problems and creates a whole lot more. In more detail, the basic blockchain as a BB scenario [YN16] is very similar to the snapshot of a show of hands analogy that we repeatedly used in explaining the properties of voting systems. More specifically, each candidate is represented by a Bitcoin address. When a voter wants to cast a vote for a specific candidate, she sends a fixed small payment to the address of the candidate. This transaction is recorded on the blockchain. Consequently, the voters themselves need to be represented using addresses that function as pseudonyms. Both a permissioned and a permissionless blockchain could be used. When the voting period ends, everybody can check the blockchain and sum the amount of coins received by each candidate address and declare the winner. Since all the votes are represented as transactions in the publicly readable blockchain, everybody can audit them and verify the tally. If a public permissionless blockchain is used, the voting system is integrated into the everyday operation of the cryptocurrency, and as a result, the guarantees are even higher. However, there are things that need to be considered and as commonly cited, the devil is in the details.

Firstly, in order to enable eligibility verifiability, there must be a protocol that determines the mapping between voter addresses and their real identities. If it is executed by a registration authority, then a trusted third party is introduced to the voting system and the use of the blockchain resembles the permissioned case. If the elections are conducted in a small scale, then it is reasonable to assume that the voters can jointly agree to their eligibility, using commonly agreed upon information. On the other hand, in large scale elections, voters need to simply ‘vote and go’ and cannot be expected to run consensus protocols on their own. In fact, some aspects of real-world voting especially at the national scale are claimed to be inherently uncentralizable as argued in [Hei+18]. But eligibility verifiability requires an identity provider that must bind real-world identities to voting credentials granted only to the ones with the

right to vote. This means that a registration authority is required, but it must be prevented from associating voters with addresses. This can be done using cryptographic anonymity principles such as blind signatures and mixnets. While this binding can be made verifiable with tools such as PACBS the identities are still maintained in a centralized database operated by a nation-state – a trusted third party

A second drawback is that the basic blockchain voting scheme does not satisfy ballot secrecy. The use of pseudonyms provides minimum protection, but the real identities might leak or they might be deanonymized by using advanced analysis techniques [Mei+16]. A solution employs techniques used in homomorphic voting systems. The voters encrypt their choice of candidate and instead of sending transactions to each candidate, they send transactions to a single address owned by a tallier. There are many ways to embed the encrypted ballot inside the transaction, depending on the type of blockchain used: In the case of Bitcoin, encryption will be done outside the system and the hidden ballot along with the zero-knowledge proofs of validity will have to be stored on a separate data source and linked with the transaction with a construct like the OP_RETURN statement. This approach has the downside that the external data source becomes a trusted third party that has control over the actual data. Alternatively, the voters can self-tally the elections using the recorded data. For this a blockchain that supports smart contracts must be used. This is the idea implemented in the Open Vote Network [MSH17].

Except for secrecy, the basic blockchain voting scheme does not support fairness, as anybody can calculate intermediate results by monitoring the transactions broadcast on the blockchain. This can affect the choice of late-come voters [YN16]. This is easy to understand by recalling the show of hands analogy, where an initially generated momentum on a voting option is self-reinforced due to the transparency of the process.

The proposed blockchain solutions can be examined in many more aspects; for instance, if they are truly non-centralizable or whether they support large scale elections. An analysis of [DP18] finds that only the OpenVote network, is a functionally decentralized platform. However, as we said, it has scaling problems, as it is meant to be used only for small scale or boardroom elections (for a maximum number of 50 voters as the authors themselves claim). In general, all public blockchains suffer from efficiency issues, both in the number of transactions that are cleared per second as well as in the wait time for a transaction to be confirmed. These make them difficult to use in large-scale elections. It must also be stressed that, the decentralization arguments only hold for the application layer, i.e. the voting protocol itself. While the network layer – the blockchain – is considered decentralized, a closer look reveals that there exists concentration on mining power [Hei+18]. For example, a

recent work by [Gen+18] finds that 90% of mining power is in the hands of 16 miners in Bitcoin and 11 miners in Ethereum. This is a troubling result, because it implies that the voting protocols are designed on top of a leaky abstraction.

Finally, blockchain voting worsens a major problem faced by all electronic voting schemes especially cryptographic ones. Enfranchisement requires that the voter understands the process she participates in. This is difficult to do when the system is built on top of complex mathematical concepts that cannot be easily explained. Some claim that approaches like code-voting [Cha01] aim to hide abstract such difficulties away from the voter, however they have not proved their effectiveness. This situation is made worse by the probabilistic and the incentive-based nature of the security of many of the schemes we dealt with. This applies especially to blockchain voting. While their scientific analysis is sound, the average voter might not be confident with less than perfect solutions. Opponents of such systems might even take advantage of such misconceptions in order to cast doubt on these voting solutions.

To conclude, voting on the blockchain solves only a single part of the voting problem – how to reach consensus on the transcript of the protocol. However, this is only the tip of the iceberg. In order to be able to convince the loser candidates, as well as the electoral body and observers, a voting protocol must satisfy strict security requirements, for which the blockchain does little to help so far.

6.3 Epilogue

Electronic voting is not purely a cryptographic or technical problem. Even if the perfect protocol existed, the voters must willingly accept it in order to express their opinions through it. User acceptance requires understanding, first of all, posing a problem for all cryptographic voting systems. We cannot expect all users to know about homomorphic encryption or zero-knowledge proofs in order to be able to vote. For such users, who comprise the majority of the electorate, the use of such systems would be a black box. If they were forced to use an e-voting system they would feel that they were deprived of their freedom of expression. As a result, the introduction of e-voting protocols should be consensual and gradual, otherwise, it is doomed to fail.

To foster consensus, the authorities that set up the electronic elections must first educate the voters, train them using pilot projects and be frank about the relative advantages of electronic voting when compared against physical. In fact, a detailed comparison can prove that if one applies the ‘paranoid mindset’, usually reserved only for electronic voting to physical elections, many flaws and implicit trust assumptions can be revealed. This can aid in user acceptance, as long as it is honest. For the latter,

e-voting cannot be straightly applied to national elections. Smaller-scale elections should be conducted first in professional organizations, societies, worker unions, and even in schools. Such elections will allow users to be acquainted with electronic elections and related processes.

Additionally, an e-voting scheme should not be a monoculture, meaning that it must not only entail a particular system. Voters can vote with one application, verify their vote with a different one and check the result with a third. These applications must be built from different providers. As a result, they must not only be open - source but also support open application programming interfaces to facilitate the exchange of data.

Finally, e-voting, as cryptography itself, can be a force for good or bad. It can be used to improve our democratic processes or by (authoritarian) regimes to legitimize their views. This leaves a moral obligation to designers and operators of such (cryptographic) voting schemes to add an extra layer of security, by refusing to build or operate systems that go against the interests of the public.

Bibliography

[DH76] Whitfield Diffie and Martin E. Hellman. "New directions in cryptography". In: *IEEE Trans. Inf. Theory* 22.6 (1976), pages 644–654. DOI: 10.1109/TIT.1976.1055638. URL: https://doi.org/10.1109/TIT.1976.1055638.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". In: *Commun. ACM* 21.2 (1978), pages 120–126. DOI: 10.1145/359340.359342. URL: http://doi.acm.org/10.1145/359340.359342.

[Sha79] Adi Shamir. "How to Share a Secret". In: *Commun. ACM* 22.11 (1979), pages 612–613. DOI: 10.1145/359168.359176. URL: http://doi.acm.org/10.1145/359168.359176.

[Cha81] David Chaum. "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms". In: *Commun. ACM* (1981), pages 84–88.

[Cha82] David Chaum. "Blind Signatures for Untraceable Payments". In: *Advances in Cryptology: Proceedings of CRYPTO '82, Santa Barbara, California, USA, August 23-25, 1982.* Edited by David Chaum, Ronald L. Rivest, and Alan T. Sherman. Plenum Press, New York, 1982, pages 199–203.

[LSP82] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. "The Byzantine Generals Problem". In: *ACM Trans. Program. Lang. Syst.* 4.3 (1982), pages 382–401.

[Cha83] David Chaum. "Blind Signatures for Untraceable Payments". In: *CRYPTO '82.* Edited by D. Chaum, R.L. Rivest, and A.T. Sherman. 1983, pages 199–203.

[GM84] Shafi Goldwasser and Silvio Micali. "Probabilistic encryption". In: *Journal of computer and system sciences* 28.2 (1984), pages 270–299.

[Gam85] Taher El Gamal. "A public key cryptosystem and a signature scheme based on discrete logarithms". In: *IEEE Trans. Information Theory* 31.4 (1985), pages 469–472. DOI: 10.1109/TIT.1985.1057074. URL: https://doi.org/10.1109/TIT.1985.1057074.

[GMR85] S Goldwasser, S Micali, and C Rackoff. "The Knowledge Complexity of Interactive Proof-Systems". In: *Proceedings of the Seventeenth Annual ACM*

Symposium on Theory of Computing. STOC '85. Association for Computing Machinery, 1985, pages 291–304. ISBN: 0897911512. DOI: 10.1145/22145.22178. URL: https://doi.org/10.1145/22145.22178.

[FS86] Amos Fiat and Adi Shamir. "How to Prove Yourself: Practical Solutions to Identification and Signature Problems". In: *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*. Edited by Andrew M. Odlyzko. Volume 263. Lecture Notes in Computer Science. Springer, 1986, pages 186–194. DOI: 10.1007/3-540-47721-7_12. URL: https://doi.org/10.1007/3-540-47721-7_12.

[Ben87] Josh Benaloh. "Verifiable Secret-Ballot Elections". PhD thesis. 1987. URL: https://www.microsoft.com/en-us/research/publication/verifiable-secret-ballot-elections/.

[Fel87] Paul Feldman. "A Practical Scheme for Non-interactive Verifiable Secret Sharing". In: *FOCS*. IEEE Computer Society, 1987, pages 427–437.

[CA89] David Chaum and Hans Van Antwerpen. "Undeniable Signatures". In: *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*. Edited by Gilles Brassard. Volume 435. Lecture Notes in Computer Science. Springer, 1989, pages 212–216. DOI: 10.1007/0-387-34805-0_20. URL: https://doi.org/10.1007/0-387-34805-0_20.

[PP89] Birgit Pfitzmann and Andreas Pfitzmann. "How to Break the Direct RSA-Implementation of Mixes". In: *EUROCRYPT*. Volume 434. Lecture Notes in Computer Science. Springer, 1989, pages 373–381.

[Sch89] Claus-Peter Schnorr. "Efficient Identification and Signatures for Smart Cards". In: *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*. Edited by Gilles Brassard. Volume 435. Lecture Notes in Computer Science. Springer, 1989, pages 239–252. DOI: 10.1007/0-387-34805-0_22. URL: https://doi.org/10.1007/0-387-34805-0_22.

[CH91] David Chaum and Eugène van Heyst. "Group Signatures". In: *Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of of Cryptographic Techniques, Brighton, UK, April 8-11, 1991, Proceedings*. Edited by Donald W. Davies. Volume 547. Lecture Notes in Computer Science. Springer, 1991, pages 257–265. DOI: 10.1007/3-540-46416-6_22. URL: https://doi.org/10.1007/3-540-46416-6_22.

[DDN91] D Dolev DDN, C Dwork, and M Naor. "Non-malleable cryptography". In: *Proceedings of the 23rd Annual Symposium on the Theory of Computing, ACM*. 1991.

[Ped91] Torben P. Pedersen. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing". In: *CRYPTO*. Volume 576. Lecture Notes in Computer Science. Springer, 1991, pages 129–140.

[Sch91] Claus-Peter Schnorr. "Efficient Signature Generation by Smart Cards". In: *J. Cryptology* 4.3 (1991), pages 161–174. DOI: 10.1007/BF00196725. URL: https://doi.org/10.1007/BF00196725.

[CP92] David Chaum and Torben P. Pedersen. "Wallet Databases with Observers". In: *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings*. Edited by Ernest F. Brickell. Volume 740. Lecture Notes in Computer Science. Springer, 1992, pages 89–105. DOI: 10.1007/3-540-48071-4_7. URL: https://doi.org/10.1007/3-540-48071-4_7.

[FOO92] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. "A Practical Secret Voting Scheme for Large Scale Elections". In: *Advances in Cryptology - AUSCRYPT '92, Workshop on the Theory and Application of Cryptographic Techniques, Gold Coast, Queensland, Australia, December 13-16, 1992, Proceedings*. Edited by Jennifer Seberry and Yuliang Zheng. Volume 718. Lecture Notes in Computer Science. Springer, 1992, pages 244–251. DOI: 10.1007/3-540-57220-1_66. URL: https://doi.org/10.1007/3-540-57220-1_66.

[Oka92] Tatsuaki Okamoto. "Provably Secure and Practical Identification Schemes and Corresponding Signature Schemes". In: *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings*. Edited by Ernest F. Brickell. Volume 740. Lecture Notes in Computer Science. Springer, 1992, pages 31–53. DOI: 10.1007/3-540-48071-4_3. URL: https://doi.org/10.1007/3-540-48071-4_3.

[BR93] Mihir Bellare and Phillip Rogaway. "Random Oracles are Practical: A Paradigm for Designing Efficient Protocols". In: *ACM Conference on Computer and Communications Security*. ACM, 1993, pages 62–73.

[CP93] David Chaum and Torben P. Pedersen. "Wallet Databases with Observers". In: *CRYPTO '92*. Springer-Verlag, 1993, pages 89–105. ISBN: 3-540-57340-2. URL: http://dl.acm.org/citation.cfm?id=646757.705670.

[PIK93] Choonsik Park, Kazutomo Itoh, and Kaoru Kurosawa. "Efficient Anonymous Channel and All/Nothing Election Scheme". In: *EUROCRYPT*. Volume 765. Lecture Notes in Computer Science. Springer, 1993, pages 248–259.

[BT94] Josh Benaloh and Dwight Tuinstra. "Receipt-free secret-ballot elections (extended abstract)". In: *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing - STOC '94*. ACM Press, 1994, pages 544–553. ISBN: 0897916638. DOI: 10.1145/195058.195407. URL: http://portal.acm.org/citation.cfm?doid=195058.195407.

[Cha94] David Chaum. "Designated Confirmer Signatures". In: *Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings*. Edited by Alfredo De Santis. Volume 950. Lecture Notes in Computer Science. Springer, 1994, pages 86–91. DOI: 10.1007/BFb0053427. URL: https://doi.org/10.1007/BFb0053427.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. "Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols". In: *Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings*. Edited by Yvo Desmedt. Volume 839. Lecture Notes in Computer Science. Springer, 1994, pages 174–187. DOI: 10.1007/3-540-48658-5_19. URL: https://doi.org/10.1007/3-540-48658-5_19.

[Pfi94] Birgit Pfitzmann. "Breaking Efficient Anonymous Channel". In: *EUROCRYPT*. Volume 950. Lecture Notes in Computer Science. Springer, 1994, pages 332–340.

[SK95] Kazue Sako and Joe Kilian. "Receipt-Free Mix-Type Voting Scheme - A Practical Solution to the Implementation of a Voting Booth". In: *EUROCRYPT*. Volume 921. Lecture Notes in Computer Science. Springer, 1995, pages 393–403.

[Cra+96] Ronald Cramer, Matthew Franklin, Berry Schoenmakers, and Moti Yung. "Multi-Authority Secret-Ballot Elections with Linear Work". In: 1996, pages 72–83. DOI: 10.1007/3-540-68339-9_7. URL: http://link.springer.com/10.1007/3-540-68339-9_7.

[JSI96] Markus Jakobsson, Kazue Sako, and Russell Impagliazzo. "Designated Verifier Proofs and Their Applications". In: *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding*.

Edited by Ueli M. Maurer. Volume 1070. Lecture Notes in Computer Science. Springer, 1996, pages 143–154. DOI: 10.1007/3-540-68339-9_13. URL: https://doi.org/10.1007/3-540-68339-9_13.

[PS96] David Pointcheval and Jacques Stern. "Provably Secure Blind Signature Schemes". In: *Advances in Cryptology - ASIACRYPT '96, International Conference on the Theory and Applications of Cryptology and Information Security, Kyongju, Korea, November 3-7, 1996, Proceedings*. Edited by Kwangjo Kim and Tsutomu Matsumoto. Volume 1163. Lecture Notes in Computer Science. Springer, 1996, pages 252–265. DOI: 10.1007/BFb0034852. URL: https://doi.org/10.1007/BFb0034852.

[CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. "A secure and optimally efficient multi-authority election scheme". In: *Transactions on Emerging Telecommunications Technologies* (1997), pages 481–490.

[JLO97] Ari Juels, Michael Luby, and Rafail Ostrovsky. "Security of Blind Digital Signatures (Extended Abstract)". In: *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*. Edited by Burton S. Kaliski Jr. Volume 1294. Lecture Notes in Computer Science. Springer, 1997, pages 150–164. DOI: 10.1007/BFb0052233. URL: https://doi.org/10.1007/BFb0052233.

[Oka97] Tatsuaki Okamoto. "Receipt-Free Electronic Voting Schemes for Large Scale Elections". In: *Security Protocols Workshop*. Volume 1361. Lecture Notes in Computer Science. Springer, 1997, pages 25–35.

[Abe98] Masayuki Abe. "Universally Verifiable Mix-net with Verification Work Indendent of the Number of Mix-servers". In: *EUROCRYPT*. Volume 1403. Lecture Notes in Computer Science. Springer, 1998, pages 437–447.

[Bel+98] Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. "Relations among notions of security for public-key encryption schemes". In: *Annual International Cryptology Conference*. Springer. 1998, pages 26–45.

[Bon98] Dan Boneh. "The Decision Diffie-Hellman Problem". In: *Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings*. Edited by Joe Buhler. Volume 1423. Lecture Notes in Computer Science. Springer, 1998, pages 48–63. DOI: 10.1007/BFb0054851. URL: https://doi.org/10.1007/BFb0054851.

[Ohk+99] Miyako Ohkubo, Fumiaki Miura, Masayuki Abe, Atsushi Fujioka, and Tatsuaki Okamoto. "An Improvement on a Practical Secret Voting Scheme".

English. In: *Information Security*. LNCS. 1999, pages 225–234. ISBN: 978-3-540-66695-0. DOI: 10.1007/3-540-47790-X_19. URL: http://dx.doi.org/10.1007/3-540-47790-X_19.

[Sch99] Berry Schoenmakers. "A Simple Publicly Verifiable Secret Sharing Scheme and Its Application to Electronic". In: *CRYPTO*. Volume 1666. Lecture Notes in Computer Science. Springer, 1999, pages 148–164.

[AO00] Masayuki Abe and Tatsuaki Okamoto. "Provably Secure Partially Blind Signatures". In: *Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings*. Edited by Mihir Bellare. Volume 1880. Lecture Notes in Computer Science. Springer, 2000, pages 271–286. DOI: 10.1007/3-540-44598-6_17. URL: https://doi.org/10.1007/3-540-44598-6_17.

[Ger+00] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. "Protecting Data Privacy in Private Information Retrieval Schemes". In: *J. Comput. Syst. Sci.* 60.3 (2000), pages 592–629. DOI: 10.1006/jcss.1999.1689. URL: https://doi.org/10.1006/jcss.1999.1689.

[HS00] Martin Hirt and Kazue Sako. "Efficient receipt-free voting based on homomorphic encryption". In: *Proceedings of the 19th international conference on Theory and application of cryptographic techniques*. EUROCRYPT'00. 2000, pages 539–556. ISBN: 3-540-67517-5. URL: http://dl.acm.org/citation.cfm?id=1756169.1756222.

[JJ00] Markus Jakobsson and Ari Juels. "Mix and Match: Secure Function Evaluation via Ciphertexts". In: *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings*. Edited by Tatsuaki Okamoto. Volume 1976. Lecture Notes in Computer Science. Springer, 2000, pages 162–177. DOI: 10.1007/3-540-44448-3_13. URL: https://doi.org/10.1007/3-540-44448-3_13.

[PS00] David Pointcheval and Jacques Stern. "Security Arguments for Digital Signatures and Blind Signatures". In: *J. Cryptology* 13.3 (2000), pages 361–396. DOI: 10.1007/s001450010003. URL: https://doi.org/10.1007/s001450010003.

[Can01] Ran Canetti. "Universally composable security: A new paradigm for cryptographic protocols". In: *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. IEEE. 2001, pages 136–145.

[Cha01] David Chaum. "Surevote: technical overview". In: *Proceedings of the workshop on trustworthy elections (WOTE'01)*. 2001.

[FS01] Jun Furukawa and Kazue Sako. "An Efficient Scheme for Proving a Shuffle". In: *CRYPTO*. Volume 2139. Lecture Notes in Computer Science. Springer, 2001, pages 368–387.

[Nef01] C. Andrew Neff. "A verifiable secret shuffle and its application to e-voting". In: *ACM Conference on Computer and Communications Security*. ACM, 2001, pages 116–125.

[RST01] Ronald L. Rivest, Adi Shamir, and Yael Tauman. "How to Leak a Secret". In: *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*. Edited by Colin Boyd. Volume 2248. Lecture Notes in Computer Science. Springer, 2001, pages 552–565. DOI: 10.1007/3-540-45682-1_32. URL: https://doi.org/10.1007/3-540-45682-1_32.

[Sch01] Claus-Peter Schnorr. "Security of Blind Discrete Log Signatures against Interactive Attacks". In: *Information and Communications Security, Third International Conference, ICICS 2001, Xian, China, November 13-16, 2001*. Edited by Sihan Qing, Tatsuaki Okamoto, and Jianying Zhou. Volume 2229. Lecture Notes in Computer Science. Springer, 2001, pages 1–12. DOI: 10.1007/3-540-45600-7_1. URL: https://doi.org/10.1007/3-540-45600-7_1.

[Bol02] Alexandra Boldyreva. "Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme". In: *Public Key Cryptography — PKC 2003*. Edited by Yvo G. Desmedt. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pages 31–46. ISBN: 978-3-540-36288-3.

[JJR02] Markus Jakobsson, Ari Juels, and Ronald L. Rivest. "Making Mix Nets Robust for Electronic Voting by Randomized Partial Checking". In: *USENIX Security Symposium*. USENIX, 2002, pages 339–353.

[KY02] Aggelos Kiayias and Moti Yung. "Self-tallying Elections and Perfect Ballot Secrecy". In: *Public Key Cryptography, 5th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2002, Paris, France, February 12-14, 2002, Proceedings*. 2002, pages 141–158. DOI: 10.1007/3-540-45664-3_10. URL: https://doi.org/10.1007/3-540-45664-3_10.

[Wag02] David Wagner. "A generalized birthday problem". In: *Annual International Cryptology Conference*. Springer. 2002, pages 288–304.

[Bel+03] Mihir Bellare, Chanathip Namprempre, David Pointcheval, and Michael Semanko. "The One-More-RSA-Inversion Problems and the Security of Chaum's Blind Signature Scheme." In: *Journal of Cryptology* 16.3 (2003).

[Cha04] David Chaum. "Secret-Ballot Receipts: True Voter-Verifiable Elections". In: *IEEE Secur. Priv.* 2.1 (2004), pages 38–47.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. "Tor: The Second-Generation Onion Router". In: *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*. SSYM'04. USENIX Association, 2004, page 21.

[Gro04] Jens Groth. "Efficient Maximal Privacy in Boardroom Voting and Anonymous Broadcast". In: *FC 2004*. Volume 3110. LNCS. Springer, 2004, pages 90–104.

[Nef04] C. Andrew Neff. *Practical High Certainty Intent Verification for Encrypted Votes*. 2004.

[Gro05] Jens Groth. "Non-interactive Zero-Knowledge Arguments for Voting". In: *Applied Cryptography and Network Security, Third International Conference, ACNS 2005, New York, NY, USA, June 7-10, 2005, Proceedings*. Edited by John Ioannidis, Angelos D. Keromytis, and Moti Yung. Volume 3531. Lecture Notes in Computer Science. 2005, pages 467–482. DOI: 10.1007/11496137_32. URL: https://doi.org/10.1007/11496137_32.

[JCJ05] Ari Juels, Dario Catalano, and Markus Jakobsson. "Coercion-resistant electronic elections". In: *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society, WPES 2005, Alexandria, VA, USA, November 7, 2005*. Edited by Vijay Atluri, Sabrina De Capitani di Vimercati, and Roger Dingledine. ACM, 2005, pages 61–70. DOI: 10.1145/1102199.1102213. URL: http://doi.acm.org/10.1145/1102199.1102213.

[LWB05] Helger Lipmaa, Guilin Wang, and Feng Bao. "Designated Verifier Signature Schemes: Attacks, New Security Notions and a New Construction". In: *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005, Proceedings*. Edited by Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung. Volume 3580. Lecture Notes in Computer Science. Springer, 2005, pages 459–471. DOI: 10.1007/11523468_38. URL: https://doi.org/10.1007/11523468_38.

[Smi05] Warren D. Smith. "New cryptographic voting scheme with best-known theoretical properties". In: *Frontiers in Electronic Elections (FEE 2005)*. 2005.

[Wik05] Douglas Wikström. "A sender verifiable mix-net and a new proof of a shuffle". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Volume 3788 LNCS. 2005, pages 273–292. ISBN: 3540306846.

[AN06] Ben Adida and C. Andrew Neff. "Ballot Casting Assurance". In: *Electronic Voting Technology Workshop, EVT'06, Vancouver, BC, Canada, August 1,*

2006. Edited by Dan S. Wallach and Ronald L. Rivest. USENIX Association, 2006. URL: https://www.usenix.org/conference/evt-06/ballot-casting-assurance.

[Ben06] J. Benaloh. "Simple verifiable elections". In: *Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop* (2006). ISSN: 03029743. DOI: 10.1007/978-3-642-28641-4_7.

[JV06] Hugo L Jonker and Erik P de Vink. "Formalising receipt-freeness". In: *International Conference on Information Security*. Springer. 2006, pages 476–488.

[MM06] Ülle Madise and Tarvi Martens. "E-voting in Estonia 2005. The first practice of country-wide binding Internet voting in the world". In: *Electronic Voting 2006 - 2nd International Workshop*. 2006.

[MN06] Tal Moran and Moni Naor. "Receipt-Free Universally-Verifiable Voting with Everlasting Privacy". In: 2006, pages 373–392. DOI: 10.1007/11818175_22. URL: http://link.springer.com/10.1007/11818175_22.

[AFT07] Roberto Araújo, Sébastien Foulle, and Jacques Traoré. "A practical and secure coercion resistant scheme for remote elections". In: *Frontiers of Electronic Voting*. 2007. URL: http://drops.dagstuhl.de/opus/volltexte/2008/1295.

[Li+07] Yong Li, Willy Susilo, Yi Mu, and Dingyi Pei. "Designated Verifier Signature: Definition, Framework and New Constructions". In: *Ubiquitous Intelligence and Computing, 4th International Conference, UIC 2007, Hong Kong, China, July 11-13, 2007, Proceedings*. Edited by Jadwiga Indulska, Jianhua Ma, Laurence Tianruo Yang, Theo Ungerer, and Jiannong Cao. Volume 4611. Lecture Notes in Computer Science. Springer, 2007, pages 1191–1200. DOI: 10.1007/978-3-540-73549-6_116. URL: https://doi.org/10.1007/978-3-540-73549-6_116.

[WAB07] Stefan G. Weber, Roberto Araujo, and Johannes Buchmann. "On Coercion-Resistant Electronic Elections with Linear Work." In: *ARES*. IEEE, 2007, pages 908–916. URL: http://dblp.uni-trier.de/db/conf/IEEEares/ares2007.html#WeberAB07.

[Adi08] Ben Adida. "Helios: web-based open-audit voting". In: *Proceedings of the 17th conference on Security symposium*. USENIX Association, 2008, pages 335–348. URL: http://dl.acm.org/citation.cfm?id=1496711.1496734.

[CH08] Jeremy Clark and Urs Hengartner. "Panic Passwords: Authenticating under Duress". In: *3rd USENIX Workshop on Hot Topics in Security, HotSec'08, San Jose, CA, USA, July 29, 2008, Proceedings*. Edited by Niels Provos.

USENIX Association, 2008. URL: http://www.usenix.org/events/hotsec08/tech/full_papers/clark/clark.pdf.

[CCM08] Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. "Civitas: Toward a Secure Voting System." In: *IEEE Security and Privacy Symposium*. May 19, 2008. URL: http://dblp.uni-trier.de/db/conf/sp/sp2008.html#ClarksonCM08.

[Nak08] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. 2008. URL: http://bitcoin.org/bitcoin.pdf.

[Riv08] Ronald L Rivest. "On the notion of 'software independence' in voting systems". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 366.1881 (2008), pages 3759–3767. ISSN: 1364503X. DOI: 10.1098/rsta.2008.0149.

[MPQ09] Olivier de Marneffe, Olivier Pereira, and Jean-Jacques Quisquater. "Electing a University President Using Open-Audit Voting: Analysis of Real-World Use of Helios". In: *2009 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '09, Montreal, Canada, August 10-11, 2009*. Edited by David Jefferson, Joseph Lorenzo Hall, and Tal Moran. USENIX Association, 2009. URL: https://www.usenix.org/conference/evtwote-09/electing-university-president-using-open-audit-voting-analysis-real-world-use.

[Wik09] Douglas Wikström. "A commitment-consistent proof of a shuffle". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2009. ISBN: 3642026192. DOI: 10.1007/978-3-642-02620-1_28.

[Ara+10] Roberto Araújo, Narjes Ben Rajeb, Riadh Robbana, Jacques Traoré, and Souheib Yousfi. "Towards Practical and Secure Coercion-Resistant Electronic Elections". In: *CANS*. 2010, pages 278–297.

[Che+10] Benoît Chevallier-Mames, Pierre-Alain Fouque, David Pointcheval, Julien Stern, and Jacques Traoré. "On Some Incompatible Properties of Voting Schemes". In: *Towards Trustworthy Elections, New Directions in Electronic Voting*. 2010, pages 191–199. DOI: 10.1007/978-3-642-12980-3_11. URL: http://dx.doi.org/10.1007/978-3-642-12980-3_11.

[ED10] Saghar Estehghari and Yvo Desmedt. "Exploiting the Client Vulnerabilities in Internet E-voting Systems: Hacking Helios 2.0 as an Example". In: *EVT/WOTE*. USENIX Association, 2010.

[HRZ10] Feng Hao, Peter Y. A. Ryan, and Piotr Zielinski. "Anonymous voting by two-round public discussion". In: *IET Information Security* 4.2 (2010), pages 62–67.

[KRS10] Steve Kremer, Mark Ryan, and Ben Smyth. "Election Verifiability in Electronic Voting Protocols". In: *ESORICS*. Volume 6345. Lecture Notes in Computer Science. Springer, 2010, pages 389–404.

[KTV10] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. "Accountability: Definition and relationship to verifiability". In: *Proceedings of the ACM Conference on Computer and Communications Security* (2010), pages 526–535. ISSN: 15437221. DOI: 10.1145/1866307.1866366.

[MN10] Tal Moran and Moni Naor. "Split-ballot voting". In: *ACM Transactions on Information and System Security* 13.2 (2010), pages 1–43. ISSN: 10949224. DOI: 10.1145/1698750.1698756. URL: http://portal.acm.org/citation.cfm?doid=1698750.1698756.

[TW10] Björn Terelius and Douglas Wikström. "Proofs of restricted shuffles". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2010. ISBN: 3642126774. DOI: 10.1007/978-3-642-12678-9_7.

[Ber+11] David Bernhard, Véronique Cortier, Olivier Pereira, Ben Smyth, and Bogdan Warinschi. "Adapting Helios for Provable Ballot Privacy". In: *ESORICS*. Volume 6879. Lecture Notes in Computer Science. Springer, 2011, pages 335–354.

[Bla+11] Olivier Blazy, Georg Fuchsbauer, David Pointcheval, and Damien Vergnaud. "Signatures on randomizable ciphertexts". In: *International Workshop on Public Key Cryptography*. Springer. 2011, pages 403–422.

[CH11] Jeremy Clark and Urs Hengartner. "Selections: Internet Voting with Over-the-Shoulder Coercion-Resistance". In: *Financial Cryptography and Data Security - 15th International Conference, FC 2011, Gros Islet, St. Lucia, February 28 - March 4, 2011, Revised Selected Papers*. Edited by George Danezis. Volume 7035. Lecture Notes in Computer Science. Springer, 2011, pages 47–61. DOI: 10.1007/978-3-642-27576-0_4. URL: https://doi.org/10.1007/978-3-642-27576-0_4.

[KHF11] Reto E. Koenig, Rolf Haenni, and Stephan Fischli. "Preventing Board Flooding Attacks in Coercion-Resistant Electronic Voting Schemes". In: *SEC*. 2011, pages 116–127.

[KTV11] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. "Verifiability, privacy, and coercion-resistance: New insights from a case study". In: *2011 IEEE Symposium on Security and Privacy*. IEEE. 2011, pages 538–553.

[Sch+11] Michael Schläpfer, Rolf Haenni, Reto E. Koenig, and Oliver Spycher. "Efficient Vote Authorization in Coercion-Resistant Internet Voting". In: *E-Voting and Identity - Third International Conference, VoteID 2011, Tallinn,*

Estonia, September 28-30, 2011, Revised Selected Papers. Edited by Aggelos Kiayias and Helger Lipmaa. Volume 7187. Lecture Notes in Computer Science. Springer, 2011, pages 71–88. DOI: 10.1007/978-3-642-32747-6_5. URL: https://doi.org/10.1007/978-3-642-32747-6_5.

[BG12] Stephanie Bayer and Jens Groth. "Efficient Zero-Knowledge Argument for Correctness of a Shuffle". In: *EUROCRYPT 2012*. 2012, pages 263–280.

[BPW12] David Bernhard, Olivier Pereira, and Bogdan Warinschi. "How Not to Prove Yourself: Pitfalls of the Fiat-Shamir Heuristic and Applications to Helios". In: *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*. Edited by Xiaoyun Wang and Kazue Sako. Volume 7658. Lecture Notes in Computer Science. Springer, 2012, pages 626–643. DOI: 10.1007/978-3-642-34961-4_38. URL: https://doi.org/10.1007/978-3-642-34961-4_38.

[DGA12] Denise Demirel, J Van De Graaf, and R Araújo. "Improving Helios with Everlasting Privacy Towards the Public". In: *EVT/WOTE'12 Proceedings of the 2012 international conference on Electronic Voting Technology/Workshop on Trustworthy Elections* (2012).

[Kha+12] Dalia Khader, Ben Smyth, Peter Y. A. Ryan, and Feng Hao. "A Fair and Robust Voting System by Broadcast". In: *5th International Conference on Electronic Voting 2012, (EVOTE 2012)*. Volume P-205. LNI. GI, 2012, pages 285–299.

[KTV12a] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. "A game-based definition of coercion resistance and its applications". In: *Journal of Computer Security* 20.6 (2012), pages 709–764.

[KTV12b] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. "Clash Attacks on the Verifiability of E-Voting Systems". In: *IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA*. IEEE Computer Society, 2012, pages 395–409. DOI: 10.1109/SP.2012.32. URL: https://doi.org/10.1109/SP.2012.32.

[LS12] Mark Lindeman and Philip B. Stark. *A gentle introduction to risk-limiting audits*. 2012. DOI: 10.1109/MSP.2012.56.

[SU12] Dominique Schröder and Dominique Unruh. "Security of Blind Signatures Revisited". In: *Public Key Cryptography*. Volume 7293. Lecture Notes in Computer Science. Springer, 2012, pages 662–679.

[Spy+12] Oliver Spycher, Reto Koenig, Rolf Haenni, and Michael Schlapfer. "A new approach towards coercion-resistant remote e-voting in linear time". In:

FC 2011. 2012. ISBN: 978-3-642-27575-3. URL: http://dx.doi.org/10.1007/978-3-642-27576-0_15.

[UH12] Jeremy Clark Urs and Hengartner. "Selections: Internet Voting with Over-the-Shoulder Coercion-Resistance". In: *FC 2011*. 2012. ISBN: 978-3-642-27576-0. DOI: 10.1007/978-3-642-27576-0_4. URL: http://dx.doi.org/10.1007/978-3-642-27576-0_4".

[Ara+13] Myrto Arapinis, Véronique Cortier, Steve Kremer, and Mark Ryan. "Practical everlasting privacy". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Volume 7796 LNCS. 2013, pages 21–40. ISBN: 9783642368295. DOI: 10.1007/978-3-642-36830-1_2. URL: http://link.springer.com/10.1007/978-3-642-36830-1_2.

[AT13] Roberto Araújo and Jacques Traoré. "A Practical Coercion Resistant Voting Scheme Revisited". In: *VOTE-ID*. 2013, pages 193–209.

[Ben13] Josh Benaloh. "Rethinking Voter Coercion: The Realities Imposed by Technology". In: *USENIX Journal of Election Technology and Systems (JETS)* (2013).

[BDV13] Johannes Buchmann, Denise Demirel, and Jeroen Van De Graaf. "Towards a publicly-verifiable mix-net providing everlasting privacy". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Volume 7859 LNCS. 2013, pages 197–204. ISBN: 9783642398834. DOI: 10.1007/978-3-642-39884-1_16. URL: http://link.springer.com/10.1007/978-3-642-39884-1_16.

[CS13] Véronique Cortier and Ben Smyth. "Attacking and fixing Helios: An analysis of ballot secrecy". In: *J. Comput. Secur.* 21.1 (2013), pages 89–148.

[CPP13] Edouard Cuvelier, Olivier Pereira, and Thomas Peters. "Election Verifiability or Ballot Privacy: Do We Need to Choose?" In: *ESORICS 2013*. 2013, pages 481–498. DOI: 10.1007/978-3-642-40203-6_27. URL: http://dx.doi.org/10.1007/978-3-642-40203-6_27.

[JMP13] Hugo Jonker, Sjouke Mauw, and Jun Pang. "Privacy and verifiability in voting systems: Methods, developments and trends". In: *Comput. Sci. Rev.* 10 (2013), pages 1–30. DOI: 10.1016/j.cosrev.2013.08.002. URL: https://doi.org/10.1016/j.cosrev.2013.08.002.

[Tso+13] Georgios Tsoukalas, Kostas Papadimitriou, Panos Louridas, and Panayiotis Tsanakas. "From Helios to Zeus". In: *USENIX Journal of Election Technology and Systems (JETS)* 1.1 (2013), pages 1–17. URL: https://www.usenix.org/system/files/conference/evtwote13/jets-0101-tsoukalas.pdf.

[But14] Vitalik Buterin. *Ethereum: A next-generation smart contract and decentralized application platform.* 2014. URL: https://github.com/ethereum/wiki/wiki/White-Paper.

[Cor+14] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachène. "Election Verifiability for Helios under Weaker Trust Assumptions". In: *ESORICS (2)*. Volume 8713. Lecture Notes in Computer Science. Springer, 2014, pages 327–344.

[CS14] Chris Culnane and Steve A. Schneider. "A Peered Bulletin Board for Robust Use in Verifiable Voting Systems". In: *CSF*. IEEE Computer Society, 2014, pages 169–183.

[Gro14] Panagiotis Grontas. "Secure multi party computations for electronic voting". Master's thesis. University of Athens, 2014. URL: {https://mpla.math.uoa.gr/media/theses/msc/P.\%20Grontas.pdf}.

[Hoh+14] Susan Hohenberger, Steven Myers, Rafael Pass, and Abhi Shelat. "ANONIZE: A Large-Scale Anonymous Survey System". In: *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*. IEEE Computer Society, 2014, pages 375–389. DOI: 10.1109/SP.2014.31. URL: https://doi.org/10.1109/SP.2014.31.

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. 2nd. Chapman & Hall/CRC, 2014. ISBN: 1466570261.

[Sri+14] Sriramkrishnan Srinivasan, Chris Culnane, James Heather, Steve A. Schneider, and Zhe Xia. "Countering Ballot Stuffing and Incorporating Eligibility Verifiability in Helios". In: *NSS*. Volume 8792. Lecture Notes in Computer Science. Springer, 2014, pages 335–348.

[Ach+15] Dirk Achenbach, Carmen Kempka, Bernhard Löwe, and Jörn Müller-Quade. "Improved coercion-resistant electronic elections through deniable revoting". In: {*USENIX*} *Journal of Election Technology and Systems ({JETS})* 3 (2015), pages 26–45.

[Alw+15] Joël Alwen, Rafail Ostrovsky, Hong-Sheng Zhou, and Vassilis Zikas. "Incoercible Multi-party Computation and Universally Composable Receipt-Free Voting". In: *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*. Edited by Rosario Gennaro and Matthew Robshaw. Volume 9216. Lecture Notes in Computer Science. Springer, 2015, pages 763–780. DOI: 10.1007/978-3-662-48000-7_37. URL: https://doi.org/10.1007/978-3-662-48000-7_37.

[Ben+15] Josh Benaloh, Ronald L. Rivest, Peter Y. A. Ryan, Philip B. Stark, Vanessa Teague, and Poorvi L. Vora. "End-to-end verifiability". In: *CoRR* abs/1504.03778 (2015). arXiv: 1504.03778. URL: http://arxiv.org/abs/1504.03778.

[Ber+15] David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. "SoK: A Comprehensive Analysis of Game-Based Ballot Privacy Definitions". In: *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*. IEEE Computer Society, 2015, pages 499–516. DOI: 10.1109/SP.2015.37. URL: https://doi.org/10.1109/SP.2015.37.

[GPZ15] Pagourtzis Aristeidis Grontas Panagiotis and Efstathios Zachos. *Computational Cryptography*. Available Online at: http://hdl.handle.net/11419/5439. thens:Hellenic Academic Libraries Link, 2015. ISBN: 978-960-603-276-9.

[KZZ15a] Aggelos Kiayias, Thomas Zacharias, and Bingsheng Zhang. "End-to-End Verifiable Elections in the Standard Model". In: *EUROCRYPT 2015*. 2015, pages 468–498. DOI: 10.1007/978-3-662-46803-6_16. URL: https://doi.org/10.1007/978-3-662-46803-6_16.

[KZZ15b] Aggelos Kiayias, Thomas Zacharias, and Bingsheng Zhang. "On the Necessity of Auditing for Election Privacy in e-Voting Systems". In: *E-Democracy - Citizen Rights in the World of the New Computing Paradigms - 6th International Conference, E-Democracy 2015, Athens, Greece, December 10-11, 2015, Proceedings*. Edited by Sokratis K. Katsikas and Alexander B. Sideridis. Volume 570. Communications in Computer and Information Science. Springer, 2015, pages 3–17. DOI: 10.1007/978-3-319-27164-4_1. URL: https://doi.org/10.1007/978-3-319-27164-4_1.

[KTV15] Oksana Kulyk, Vanessa Teague, and Melanie Volkamer. "Extending Helios Towards Private Eligibility Verifiability". In: *VoteID*. Volume 9269. Lecture Notes in Computer Science. Springer, 2015, pages 57–73.

[LH15] Philipp Locher and Rolf Haenni. "Verifiable Internet Elections with Everlasting Privacy and Minimal Trust". In: *E-Voting and Identity - 5th International Conference, VoteID 2015, Bern, Switzerland, September 2-4, 2015, Proceedings*. Edited by Rolf Haenni, Reto E. Koenig, and Douglas Wikström. Volume 9269. Lecture Notes in Computer Science. Springer, 2015, pages 74–91. DOI: 10.1007/978-3-319-22270-7_5. URL: https://doi.org/10.1007/978-3-319-22270-7_5.

[SFC15] Ben Smyth, Steven Frink, and Michael R. Clarkson. "Computational Election Verifiability: Definitions and an Analysis of Helios and JCJ". In: *IACR Cryptol. ePrint Arch.* 2015 (2015), page 233. URL: http://eprint.iacr.org/2015/233.

[Cha+16] Pyrros Chaidos, Véronique Cortier, Georg Fuchsbauer, and David Galindo. "Beleniosrf: A non-interactive receipt-free electronic voting scheme". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 2016, pages 1614–1625.

[CE16] Nicholas Chang-Fong and Aleksander Essex. "The cloudier side of cryptographic end-to-end verifiable voting: a security analysis of Helios". In: *ACSAC*. ACM, 2016, pages 324–335.

[Cor+16] V Cortier, D Galindo, R Küsters, J Müller, and T Truderung. "SoK: Verifiability Notions for E-Voting Protocols". In: *IEEE Security and Privacy Symposium*. 2016, pages 779–798.

[Küs+16] Ralf Küsters, Johannes Müller, Enrico Scapin, and Tomasz Truderung. "sElect: A Lightweight Verifiable Remote Voting System". In: *IEEE 29th Computer Security Foundations Symposium, CSF 2016, Lisbon, Portugal, June 27 - July 1, 2016*. IEEE Computer Society, 2016, pages 341–354. DOI: 10.1109/CSF.2016.31. URL: https://doi.org/10.1109/CSF.2016.31.

[LHK16] Philipp Locher, Rolf Haenni, and Reto E. Koenig. "Coercion-Resistant Internet Voting with Everlasting Privacy". In: *FC'16 Workshops, BITCOIN,VOTING,WAHC*. 2016. DOI: 10.1007/978-3-662-53357-4_11. URL: http://dx.doi.org/10.1007/978-3-662-53357-4_11.

[Mei+16] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. "A Fistful of Bitcoins: Characterizing Payments among Men with No Names". In: *Commun. ACM* 59.4 (Mar. 2016), pages 86–93. ISSN: 0001-0782. DOI: 10.1145/2896384. URL: https://doi.org/10.1145/2896384.

[YN16] Jeremy Clark Peter YA Ryan Yomna Nasser Chidinma Okoye. *Blockchains and voting: somewhere between hype and a panacea*. https://users.encs.concordia.ca/~clark/papers/draft_voting.pdf. 2016.

[BKV17] David Bernhard, Oksana Kulyk, and Melanie Volkamer. "Security proofs for participation privacy, receipt-freeness and ballot privacy for the helios voting scheme". In: *Proceedings of the 12th International Conference on Availability, Reliability and Security*. 2017, pages 1–10.

[Ber+17] Matthew Bernhard, Josh Benaloh, J. Alex Halderman, Ronald L. Rivest, Peter Y. A. Ryan, Philip B. Stark, Vanessa Teague, Poorvi L. Vora, and Dan S. Wallach. "Public Evidence from Secret Ballots". In: *E-VOTE-ID*. Volume 10615. Lecture Notes in Computer Science. Springer, 2017, pages 84–109.

[Gil+17] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. "Algorand: Scaling Byzantine Agreements for Cryptocurrencies". In: *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017, pages 51–68.

[GPZ17] Panagiotis Grontas, Aris Pagourtzis, and Alexandros Zacharakis. "Coercion Resistance in a Practical Secret Voting Scheme for Large Scale Elections". In: *14th International Symposium on Pervasive Systems, Algorithms and Networks & 11th International Conference on Frontier of Computer Science and Technology & Third International Symposium of Creative Computing, ISPAN-FCST-ISCC 2017, Exeter, United Kingdom, June 21-23, 2017*. IEEE Computer Society, 2017, pages 514–519. DOI: 10.1109/ISPAN-FCST-ISCC.2017.79. URL: https://doi.org/10.1109/ISPAN-FCST-ISCC.2017.79.

[Iov+17] Vincenzo Iovino, Alfredo Rial, Peter B Roenne, and Peter Ryan. "Using Selene to Verify your Vote in JCJ". In: *FC'17 Workshops, BITCOIN, VOTING, WAHC*. 2017. URL: http://fc17.ifca.ai/voting/papers/voting17_MainJCJ-Selene.pdf.

[MSH17] Patrick McCorry, Siamak F. Shahandashti, and Feng Hao. "A Smart Contract for Boardroom Voting with Maximum Voter Privacy". In: *FC 2017*. Volume 10322. LNCS. Springer, 2017, pages 357–375.

[PS17] Stefan Patachi and Carsten Schürmann. "Eos a Universal Verifiable and Coercion Resistant Voting Protocol". In: *E-Vote-ID 2017*. Volume 10615. LNCS. Springer, 2017, pages 210–227.

[Wil17] Jan Willemson. "Bits or Paper: Which Should Get to Carry Your Vote?" In: *E-VOTE-ID*. Volume 10615. Lecture Notes in Computer Science. Springer, 2017, pages 292–305.

[ZGP17] Alexandros Zacharakis, Panagiotis Grontas, and Aris Pagourtzis. "Conditional Blind Signatures". In: *Short version presented in 7th International Conference on Algebraic Informatics - CAI 2017* 2017 (2017), page 682. URL: http://eprint.iacr.org/2017/682.

[Boy+18] Elette Boyle, Saleet Klein, Alon Rosen, and Gil Segev. "Securing Abe's Mix-Net Against Malicious Verifiers via Witness Indistinguishability". In: *SCN*. Volume 11035. Lecture Notes in Computer Science. Springer, 2018, pages 274–291.

[CL18] Véronique Cortier and Joseph Lallemand. "Voting: You can't have privacy without individual verifiability". In: *Proceedings of the ACM Conference on Computer and Communications Security* (2018), pages 53–66. ISSN: 15437221. DOI: 10.1145/3243734.3243762.

[DP18] Lionel Dricot and Olivier Pereira. "SoK: Uncentralisable Ledgers and their Impact on Voting Systems". In: *CoRR* abs/1801.08064 (2018). arXiv: 1801.08064. URL: http://arxiv.org/abs/1801.08064.

[Gen+18] Adem Efe Gencer, Soumya Basu, Ittay Eyal, Robbert van Renesse, and Emin Gün Sirer. "Decentralization in Bitcoin and Ethereum Networks".

In: *CoRR* abs/1801.03998 (2018). arXiv: 1801.03998. URL: http://arxiv.org/abs/1801.03998.

[Gro+18] Panagiotis Grontas, Aris Pagourtzis, Alexandros Zacharakis, and Bingsheng Zhang. "Towards Everlasting Privacy and Efficient Coercion Resistance in Remote Electronic Voting". In: *Financial Cryptography and Data Security - FC 2018 International Workshops, BITCOIN, VOTING, and WTSC, Nieuwpoort, Curaçao, March 2, 2018, Revised Selected Papers*. Edited by Aviv Zohar, Ittay Eyal, Vanessa Teague, Jeremy Clark, Andrea Bracciali, Federico Pintore, and Massimiliano Sala. Volume 10958. Lecture Notes in Computer Science. Springer, 2018, pages 210–231. DOI: 10.1007/978-3-662-58820-8_15. URL: https://doi.org/10.1007/978-3-662-58820-8_15.

[Hei+18] Sven Heiberg, Ivo Kubjas, Janno Siim, and Jan Willemson. *On Trade-offs of Applying Block Chains for Electronic Voting Bulletin Boards*. Cryptology ePrint Archive, Report 2018/685. https://eprint.iacr.org/2018/685. 2018.

[Kia+18] Aggelos Kiayias, Annabell Kuldmaa, Helger Lipmaa, Janno Siim, and Thomas Zacharias. "On the Security Properties of e-Voting Bulletin Boards". In: *SCN*. Volume 11035. Lecture Notes in Computer Science. Springer, 2018, pages 505–523.

[Sch] *Securing Elections*. https://www.schneier.com/blog/archives/2018/04/securing_electi_1.html. Accessed: 2020-04-29. 2018.

[Zac18] Alexandros G. Zacharakis. "Verifiable remote electronic elections with strong privacy guarantees". Master's thesis. University of Athens, 2018.

[CGG19] Véronique Cortier, Pierrick Gaudry, and Stéphane Glondu. "Belenios: A Simple Private and Verifiable Electronic Voting System". In: *Foundations of Security, Protocols, and Equational Reasoning*. Volume 11565. Lecture Notes in Computer Science. Springer, 2019, pages 214–238.

[Cul+19] Chris Culnane, Aleksander Essex, Sarah Jamie Lewis, Olivier Pereira, and Vanessa Teague. "Knights and Knaves Run Elections: Internet Voting and Undetectable Electoral Fraud". In: *IEEE Secur. Priv.* 17.4 (2019), pages 62–70.

[FQS19] Ashley Fraser, Elizabeth A Quaglia, and Ben Smyth. "A critique of game-based definitions of receipt-freeness for voting". In: *International Conference on Provable Security*. Springer. 2019, pages 189–205.

[GP19] Panagiotis Grontas and Aris Pagourtzis. "Blockchain, consensus, and cryptography in electronic voting". In: *Homo Virtualis* 2.1 (2019), pages 79–100. ISSN: 2585-3899. DOI: 10.12681/homvir.20289. URL: https://

ejournals.epublishing.ekt.gr/index.php/homvir/article/view/20289.

[GPZ19] Panagiotis Grontas, Aris Pagourtzis, and Alexandros Zacharakis. "Security models for everlasting privacy". In: *E-Vote-ID 2019* (2019), page 140.

[KWV19] Kristjan Krips, Jan Willemson, and Sebastian Varv. "Is your vote overheard? A new scalable side-channel attack against paper voting". In: *Proceedings - 4th IEEE European Symposium on Security and Privacy, EURO S and P 2019*. 2019, pages 621–634. ISBN: 9781728111476. DOI: 10.1109/EuroSP.2019.00051.

[Sti19] Douglas Stinson. *Cryptography: Theory and Practice,Second Edition*. 4th. CRC/C&H, 2019. ISBN: 1584882069.

[Ben+20] Fabrice Benhamouda, Tancrède Lepoint, Michele Orrù, and Mariana Raykova. *On the (in)security of ROS*. Cryptology ePrint Archive, Report 2020/945. https://eprint.iacr.org/2020/945. 2020.

[BS20] Dan Boneh and Victor Shoup. *A Graduate Course in Applied Cryptography*. 2020. URL: https://toc.cryptobook.us/.

[FPS20] Georg Fuchsbauer, Antoine Plouviez, and Yannick Seurin. "Blind Schnorr Signatures and Signed ElGamal Encryption in the Algebraic Group Model". In: *EUROCRYPT (2)*. Volume 12106. Lecture Notes in Computer Science. Springer, 2020, pages 63–95.

[GPZ20] Panagiotis Grontas, Aris Pagourtzis, and Alexandros Zacharakis. "Security Models for everlasting privacy in electronic voting". In: *under submission to the International Journal of Information Security* (2020).

[Gro+20] Panagiotis Grontas, Aris Pagourtzis, Alexandros Zacharakis, and Bingsheng Zhang. "Publicly Auditable Conditional Blind Signatures". In: *under submission to the Journal of Computer Security* (2020).

[HM20] Thomas Haines and Johannes Mueller. *SoK: Techniques for Verifiable Mix Nets*. Cryptology ePrint Archive, Report 2020/490. https://eprint.iacr.org/2020/490. 2020.

[LQAT20] Wouter Lueks, Iñigo Querejeta-Azurmendi, and Carmela Troncoso. "VoteAgain: A scalable coercion-resistant voting system". In: *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pages 1553–1570. ISBN: 978-1-939133-17-5. URL: https://www.usenix.org/conference/usenixsecurity20/presentation/lueks.

[MPT20] Eleanor McMurtry, Olivier Pereira, and Vanessa Teague. *When is a test not a proof?* Cryptology ePrint Archive, Report 2020/909. https://eprint.iacr.org/2020/909. 2020.

[PBS20] Aris Pagourtzis Pourandokht Behrouz Panagiotis Grontas and Marianna Spyrakou. *On coercion resistance in decentralized voting (Short paper)*. 1st

International Workshop on Foundations of Consensus and Distributed Ledgers (FOCODILE 2020). https://www.discotec.org/2020/focodile. 2020.

[Sch20] Berry Schoenmaker. *Lecture Notes on Cryptographic Protocols*. 2020. URL: https://www.win.tue.nl/~berry/2DMI00/.

[SGY20] Mohamed Seifelnasr, Hisham S. Galal, and Amr M. Youssef. "Scalable Open-Vote Network on Ethereum". In: *IACR Cryptology ePrint Archive* 2020 (2020), page 33.

Index