



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

**ΨΗΦΟΦΟΡΙΕΣ ΚΑΤΑΤΑΞΗΣ ΣΕ ΟΜΟΜΟΡΦΙΚΑ
ΣΥΣΤΗΜΑΤΑ**

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΘΩΜΑΣ ΣΟΥΛΙΩΤΗΣ

Επιβλέπων: Αριστείδης Παγουρτζής
Αν. Καθηγητής Ε.Μ.Π

Αθήνα, Φεβρουάριος 2017



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

**ΨΗΦΟΦΟΡΙΕΣ ΚΑΤΑΤΑΞΗΣ ΣΕ ΟΜΟΜΟΡΦΙΚΑ
ΣΥΣΤΗΜΑΤΑ**

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΘΩΜΑΣ ΣΟΥΛΙΩΤΗΣ

Επιβλέπων: Αριστείδης Παγουρτζής
Αν. Καθηγητής Ε.Μ.Π

Εγκρίθηκε από την τριμελή επιτροπή την 22η Φεβρουαρίου 2017

.....
Αριστείδης Παγουρτζής
Αν. Καθηγητής Ε.Μ.Π

.....
Παναγιώτης Τσανάκας
Καθηγητής Ε.Μ.Π

.....
Άγγελος Κιαγιάς
Αν. Καθηγητής Ε.Κ.Π.Α

Αθήνα, Φεβρουάριος 2017

.....

Σουλιώτης Θωμάς

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π

© (2017) Εθνικό Μετσόβιο Πολυτεχνείο.

Περίληψη

Οι ψηφοφορίες αποτελούν ένα πολύ σημαντικό τμήμα της ζωής μας, καθώς μέσω αυτών διασφαλίζεται η δημοκρατικότητα με τον οποίο επιλέγεται η κατάλληλη σειρά ενεργειών μιας σύγχρονης κοινωνικής ομάδας. Πιο απλά, θα μπορούσαμε να πούμε ότι μέσω των ψηφοφοριών διασφαλίζεται ότι η τελική επιλογή σε ένα ζήτημα θα είναι η πιο δημοκρατική και αυτή που θα είναι η περισσότερη αποδεκτή βάση ενός συστήματος εκλογών.

Στα πλαίσια λοιπόν των ψηφοφοριών, και με την συνεχή εξέλιξη της τεχνολογίας, οι ηλεκτρονικές ψηφοφορίες είναι ξεκάθαρα το μέλλον των ψηφοφοριών. Έτσι, οι ηλεκτρονικές ψηφοφορίες έχουν εξελιχθεί πολύ τα τελευταία χρόνια. Εμείς σε αυτήν την διπλωματική θα μελετήσουμε τις ηλεκτρονικές ψηφοφορίες E2E(θα αναλυθούν αργότερα) οι οποίες θα διασφαλίζουν την ορθότητα, την ασφάλεια και την ιδιωτικότητα μέσω κρυπτογραφικών διαδικασιών.

Πιο συγκεκριμένα, αρχικά θα μελετήσουμε τα μαθηματικά και κρυπτογραφικά πρωτόκολλα τα οποία είναι απαραίτητα για την κατανόηση των μετέπειτα εννοιών. Στην συνέχεια, θα επικεντρωθούμε στις ομομορφικές ψηφοφορίες για τις οποίες γίνεται και ιδιαίτερη αναφορά. Στις ομομορφικές ψηφοφορίες λοιπόν, θα θεμελιώσουμε θεωρητικά, αρχικά, και πιο μετά σε πρακτικό επίπεδο, τις ψηφοφορίες κατάταξης/βαθμονόμησης. Τέλος, θα παρουσιαστεί η πρακτική υλοποίηση στο σύστημα Helios.

Η σημασία της παρούσας διπλωματικής, έγκειται στο ότι τα ομομορφικά συστήματα δεν επέτρεπαν μέχρι στιγμής ψηφοφορίες πέρα από τις απλές approval, λόγω των αποδείξεων ορθότητας που απαιτούνται για τις ψήφους. Συνεπώς, εμείς αρχικά αναλύουμε και προτείνουμε ένα νέο σύστημα αποδείξεων μηδενικής γνώσης που θα διασφαλίζουν την δυνατότητα να έχουμε ψηφοφορίες κατάταξης σε ομομορφικά συστήματα, ενώ τελικά υλοποιούμε πρακτικά το παραπάνω προτεινόμενο σύστημα.

Ευχαριστίες

Με την ολοκλήρωση αυτής της διπλωματικής, θα ήθελα να ευχαριστήσω τους καθηγητές της τριμελούς επιτροπής: κ. Άρη Παγουρτζή, κ. Παναγιώτη Τσανάκα και κ. Άγγελο Κιαγιά. Ιδιαίτερα, θα ήθελα να ευχαριστήσω τον κ. Παγουρτζή ο οποίος ήταν και ο επιβλέπων αυτής της εργασίας, όσο και τον κ. Παναγιώτη Γροντά, ως υποψήφιο διδάκτορα υπό την επίβλεψη του κ. Παγουρτζή, οι οποίοι με βοήθησαν αρχικά στην κατανόηση και εμβάθυνση σε όλες τις απαραίτητες εξειδικευμένες έννοιες, αλλά και στην μετέπειτα υλοποίηση και ολοκλήρωσης της διπλωματικής μου εργασίας. Ακόμη, θα ήθελα να ευχαριστήσω, όλους όσους με βοήθησαν στα πλαίσια αυτής της διπλωματικής εργασίας. Τέλος, θα ήθελα να ευχαριστήσω ιδιαίτερα την οικογένεια μου για τη διαρκή στήριξη τους και τους φίλους μου για όσα περάσαμε μαζί.

Περιεχόμενα

1	Εισαγωγή	3
1.1	Γενικά Περί Ψηφοφοριών	3
1.2	Εξέλιξη των Ψηφοφοριών και Ηλεκτρονικές Ψηφοφορίες	4
1.3	Σημασία της Κρυπτογραφίας και Σκοπός της Εργασίας	5
2	Απαραίτητες Μαθηματικές και Κρυπτογραφικές Έννοιες	7
2.1	Γενικά Περί Κρυπτογραφίας	7
2.2	Άλγεβρα	7
2.2.1	Θεωρία Ομάδων	7
2.2.2	Το Z_m και οι Ιδιότητες του	9
2.3	Θεωρία Αριθμών	9
2.3.1	Το πρόβλημα του Διακριτού Λογάριθμου (DLOG)	9
2.3.2	Το πρόβλημα Diffie-Hellman	10
2.3.3	Το πρόβλημα απόφασης Diffie-Hellman	10
2.4	Κρυπτογραφικά Συστήματα	10
2.4.1	RSA	11
2.4.2	Paillier	12
2.4.3	El Gamal	13
2.5	Ομομορφικά Συστήματα και Mixnets	13
2.5.1	Ομομορφικά Συστήματα	14
2.5.2	Mixnets	14
2.6	Ασφάλεια Κρυπτοσυστημάτων	15
2.6.1	Σημασιολογική Ασφάλεια	16
2.6.2	Ασφάλεια Επιλεγμένου Αρχικού Κειμένου IND-CPA	16
2.6.3	Ασφάλεια Επιλεγμένου Κρυπτοκειμένου IND-CCA	17
2.6.4	Ασφάλεια Τροποποιήσιμου Επιλεγμένου Κρυπτοκειμένου IND-CCA2	17
2.7	Σχήματα Δέσμευσης	18
2.7.1	Συναρτήσεις Hash	18
2.7.2	Ψηφιακές Υπογραφές	19
2.7.3	Τυφλές Υπογραφές	20
2.8	Αποδείξεις Μηδενικής Γνώσης	21
2.8.1	Γνώση του Διακριτού Λογάριθμου	22
2.8.2	Διαζευκτικές Αποδείξεις(OR-proofs)	23

2.8.3	Αποδείξεις Ανακατέματος(Shuffle-proofs)	24
2.8.4	Μη-Διαδραστικές Αποδείξεις	25
3	Το σύστημα Helios	27
3.1	Εισαγωγή στα End-to-end auditable voting systems	27
3.2	Εισαγωγή στο Helios	27
3.2.1	Πότε και Γιατί Χρησιμοποιούμε το Helios	28
3.2.2	Η Διαδικασία της Ψηφοφορίας	28
3.3	Κρυπτογραφικές Διαδικασίες	29
3.3.1	Παραγωγή της Ψήφου	29
3.3.2	Ορθότητα της Ψήφου	30
3.3.3	Διαδικασία Αποκρυπτογράφησης και Απόδειξη Ορθής Λειτουργίας	33
3.4	Η Συνολική Διαδικασία	35
3.5	Θετικά και Αρνητικά του Helios	36
4	Το Τροποποιημένο Σύστημα	38
4.1	Σκοπός του Τροποποιημένου Συστήματος	38
4.2	Πρώτες Ιδέες Υλοποίησης	39
4.3	Τελική Ιδέα Υλοποίησης	41
4.4	Ανάλυση του Νέου Συστήματος Αποδείξεων	41
4.4.1	Απόδειξη του $\sum_{h=1}^n A_{hi}A_{hj} = \delta_{ij}$	43
4.4.2	Απόδειξη του $\sum_{h=1}^n A_{hi}A_{hj}A_{hk} = \delta_{ijk}$	44
4.4.3	Η Συνολική Απόδειξη	45
4.4.4	Το Τελικό Σύστημα σε Θεωρητικό Επίπεδο	47
5	Πρακτική Υλοποίηση του Νέου Συστήματος	48
5.1	Εισαγωγή	48
5.2	Ένα Απλό Ολοκληρωμένο Πρακτικό Παράδειγμα	48
5.3	Κώδικες του Νέου Συστήματος	54
5.3.1	Οι Αλλαγές που Εφαρμόστηκαν	96
5.4	Απαραίτητες Προσθήκες για την Ορθή Λειτουργία	96
6	Μελλοντικές Ιδέες και Επίλογος	97
6.1	Βελτιώσεις του Τροποποιημένου Συστήματος	97
6.2	Επίλογος	98

Κεφάλαιο 1

Εισαγωγή

Σε αυτό το κεφάλαιο θα γίνει μία μικρή εισαγωγή στις ψηφοφορίες και πιο συγκεκριμένα θα επικεντρωθούμε στις ηλεκτρονικές ψηφοφορίες. Στην συνέχεια θα αναφερθούμε στην σημασία της κρυπτογραφίας στις ηλεκτρονικές ψηφοφορίες, και τελικά θα μιλήσουμε για τον σκοπό αυτής της διπλωματικής εργασίας.

1.1 Γενικά Περί Ψηφοφοριών

Τι είναι όμως μία ψηφοφορία; Μία ψηφοφορία είναι ένας τρόπος έτσι ώστε μία ομάδα ατόμων να πάρει μία απόφαση, να εκφράσει μία γνώμη ή πιο γενικά να εκλέξει ένα σύνολο από ιδέες ή/και άτομα. Τα άτομα τα οποία καλούνται να ψηφίσουν σε μία ψηφοφορία ονομάζονται ψηφοφόροι.

Η σημασία των ψηφοφοριών έχει κάνει την εμφάνιση της από την αρχαιότητα, ενώ η θεμελίωση της έγινε με την ύπαρξη των πρώτων δημοκρατικών κοινωνιών. Οι περισσότερες ψηφοφορίες πρέπει να ικανοποιούν ορισμένες προϋποθέσεις. Αυτές, οι προϋποθέσεις έχουν να κάνουν κυρίως με την μυστικότητα και την ατομικότητα της ψήφου, δηλαδή κανείς άλλος πέρα από τον εκάστοτε ψηφοφόρο δεν θα πρέπει να μαθαίνει πως αυτός ψήφισε. Ακόμη, μία σημαντική ιδιότητα των ψηφοφοριών, είναι η σωστή καταγραφή και καταμέτρηση των ψήφων όπως αυτές δημιουργήθηκαν και κατατέθηκαν από τους ψηφοφόρους, έτσι ώστε να μην υπάρχει αλλοίωση της βούλησης και της επιλογής των ψηφοφόρων. Προφανώς, όσο σημαντικότερη η ψηφοφορία τόσο σημαντικότερο είναι να τηρούνται κάποιοι βασικοί κανόνες.

Επιπρόσθετα, αναφερθήκαμε στις ψηφοφορίες και τις ιδιότητες τους, αλλά δεν περιγράψαμε πως ακριβώς λαμβάνει χώρα μία ψηφοφορία. Κατά την ψηφοφορία, λοιπόν οι ψηφοφόροι καλούνται να λάβουν μία απόφαση. Μία συνηθισμένη ψηφοφορία είναι μία σειρά από ερωτήσεις, με κάθε ερώτηση να απαρτίζεται από συγκεκριμένες επιλογές. Αυτές οι επιλογές, λοιπόν, επιλέγονται και εν τέλει καταμετρούνται βάση του εκάστοτε συστήματος ψηφοφοριών που έχουμε. Πιο συγκεκριμένα υπάρχουν διάφορα συστήματα ψηφοφοριών που έχουν δημιουργηθεί από την αρχαιότητα μέχρι και σήμερα ώστε να είναι εφικτή η όσο το δυνατόν περισσότερο γίνεται η ορθή έκφραση των επιλογών από ένα σώμα ψηφοφόρων.

Εκτενέστερα, τα πρώτα συστήματα ψηφοφοριών που δημιουργήθηκαν ήταν τα συστήματα στα οποία ο ψηφοφόρος καλούνταν να επιλέξει μία επιλογή από ένα σύνολο επιλογών για μία ερώτηση. Συνέχεια αυτών των συστημάτων είναι τα συστήματα στα οποία ο ψηφοφόρος μπορούσε να επιλέξει ένα συγκεκριμένο υποσύνολο των αρχικών επιλογών με κάποιες προϋποθέσεις που έθετε η εκάστοτε αρχή. Μία διαφορετική προσέγγιση των παραπάνω αποτελούν τα συστήματα στα οποία ο ψηφοφόρος εφαρμόζει κάποιου είδους κατάταξη στους ψηφοφόρους είτε βαθμολογώντας τους είτε με κάποιον άλλον τρόπο, έχοντας έτσι τα συστήματα κατάταξης. Ακόμη, ένα σύστημα βαθμολογιών στο οποίο δεν είναι απαραίτητη κάποια κατάταξη (score) αποτελεί και αυτό ένα είδους συστήματος ψηφοφοριών. Τέλος, τα παραπάνω συστήματα διαχωρίζονται βάση του τρόπου με τον οποίο ο ψηφοφόρος διαλέγει τις επιλογές του, αλλά μπορεί να γίνει και διάκριση βάση του τρόπου με τον οποίο γίνεται η καταμέτρηση των ψήφων, αλλά και ο τρόπος με τον οποίο υπολογίζεται ο νικητής της κάθε ψηφοφορίας. Ενδεικτικά κάποια από τα σημαντικότερα συστήματα ψηφοφοριών είναι τα εξής: Approval, Borda count, Copeland, IRV, Kemeny-Young, Majority judgment, Minimax, Plurality, Range voting, Ranked pairs, Runoff voting, Schulze, Sortition, arbitrary winner...

1.2 Εξέλιξη των Ψηφοφοριών και Ηλεκτρονικές Ψηφοφορίες

Παραπάνω παρουσιάσαμε τα διάφορα συστήματα ψηφοφοριών που έχουν δημιουργηθεί κατά καιρούς ενώ αναλύσαμε και ορισμένες ιδιότητες τους. Βεβαίως, σε αυτό που δεν αναφερθήκαμε εκτενέστερα είναι η διαδικασία της ίδιας της ψηφοφορίας και το πως θα καταφέρουμε να προστατεύσουμε τις απαραίτητες προϋποθέσεις που πρέπει, συνήθως, να διασφαλίζει μία ψηφοφορία. Στο παραπάνω πλαίσιο αναφέρουμε την εξέλιξη των ψηφοφοριών διαχρονικά και τον τρόπο διασφάλισης των απαραίτητων συνθηκών.

Αρχικά, οι πρώτες ψηφοφορίες στην αρχαιότητα γίνονταν είτε δια βοής, όπου ο καθένας επέλεγε τις επιλογές του και τις παρουσίαζε, όπου προφανώς δεν τηρούνται οι απαραίτητες συνθήκες για ιδιωτικότητα της ψήφου, είτε μέσω καταγραφής της ψήφου σε κάποιου είδους πήλινου αντικειμένου -όστρακο- το οποίο ήταν και από τα πρώτα ψηφοδέλτια που χρησιμοποιήθηκε στην αρχαία Ελλάδα. Με την πάροδο των χρόνων και την εξέλιξη της τεχνολογίας, τα ψηφοδέλτια έγιναν ξύλινα και εν τέλει χάρτινα. Η διαδικασία και η λογική παρέμενε πάντα η ίδια, το άτομο επέλεγε τις επιλογές του μυστικά, την κατέγραφε στο ψηφοδέλτιό του και το ψηφοδέλτιο τοποθετούνταν στην κάλπη, η οποία ανοίγονταν στο τέλος της διαδικασίας και ιδανικά καταμετρούνται όλα τα ψηφοδέλτια χωρίς να υπάρχει γνώση του ποιος ψήφισε τι.

Μέχρι, πρόσφατα η φυσική υπόσταση του ψηφοδελτίου και της κάλπης ήταν κάτι το αυτονόητο, κάτι που άλλαξε ριζικά όμως, όταν μπήκαν στις ζωές μας οι ηλεκτρονικές ψηφοφορίες. Με την δημιουργία των πρώτων υπολογιστών, ήταν δυνατό να δημιουργηθούν ηλεκτρονικά ψηφοδέλτια και ο ψηφοφόρος να συμπλήρωνε ηλεκτρονικά την ψήφο του, ενώ τελικά η κάλπη θα υπολογίζονταν πολύ εύκολα

μιας και οι πράξεις θα γίνονταν ηλεκτρονικά και άρα πολύ πιο γρήγορα.

Τα προβλήματα των ηλεκτρονικών συστημάτων, όμως, γίνονται εμφανή όσον αφορά την ιδιωτικότητα και την ακεραιότητα της ψηφοφορίας. Φυσικά, αυτά τα προβλήματα υπάρχουν και όταν τα ψηφοδέλτια είναι για παράδειγμα χάρτινα και η κάλπη έχει φυσική υπόσταση, αλλά τότε, αν θεωρήσουμε ότι οι υπεύθυνοι για τον έλεγχο και την καταμέτρηση κάνουν σωστά την δουλειά τους, σε συνδυασμό με την παρουσία διάφορων ελεγκτών της ψηφοφορίας, είναι ιδιαίτερα δύσκολο να υπάρξει κάποιο πρόβλημα για την ψηφοφορία. Προφανώς, πάντα αν η αρχή είναι διεφθαρμένη, υπάρχει κίνδυνος νοθείας και μη ορθής διατήρησης των απαραίτητων προϋποθέσεων για την εύρυθμη λειτουργία της ψηφοφορίας, απλά πάντα σε αυτού του είδους τις ψηφοφορίες στηριζόμαστε στην σωστή διαχείριση και τον κατάλληλο έλεγχο από ανθρώπους.

Από την άλλη μεριά, όσον αφορά τις ηλεκτρονικές ψηφοφορίες, ο έλεγχος ξεφεύγει από τα χέρια του ανθρώπου και πρέπει να στηριχθούμε και στην ορθή λειτουργία μιας μηχανής. Ακόμη, οι δυνατότητες χειραγώγησης τέτοιων συστημάτων, θα ήταν περισσότερες μιας και από την στιγμή που μιλάμε για προγράμματα ενός υπολογιστή, πολύς κόσμος δεν ξέρει πως να προστατευθεί και ένας κακόβουλος θα μπορούσε να προσβάλλει σοβαρά ένα τέτοιο σύστημα. Τέλος, ακόμα και με μία πλήρη αντιστοιχία με τις κλασικές ψηφοφορίες, οι ηλεκτρονικές ψηφοφορίες θα ήταν επίσης χειραγωγήσιμες από την εκάστοτε αρχή, η οποία είναι υπεύθυνη για την ψηφοφορία. Έτσι, κρίθηκε αναγκαίο να δημιουργηθεί ένα σύστημα το οποίο θα διασφάλιζε την ορθή λειτουργία μιας ηλεκτρονικής ψηφοφορίας.

1.3 Σημασία της Κρυπτογραφίας και Σκοπός της Εργασίας

Ο κλάδος της κρυπτογραφίας ήρθε για να δώσει την πολυπόθητη λύση στα προβλήματα των ηλεκτρονικών ψηφοφοριών. Πιο συγκεκριμένα, διάφορα κρυπτογραφικά πρωτόκολλα δημιουργήθηκαν έτσι ώστε να διασφαλίζεται η ιδιωτικότητα και η ακεραιότητα των ψηφοφοριών. Αρχικά, πάντα υπήρχε μία αρχή, η οποία πάλι θα μπορούσε να επηρεάσει την ψηφοφορία αν είναι πλήρως διεφθαρμένη. Στην συνέχεια, όμως χάρις σε εξελιγμένα κρυπτογραφικά πρωτόκολλα δίνεται πλέον η δυνατότητα να έχουμε συστήματα ψηφοφοριών τα οποία προστατεύουν την ακεραιότητα της ψηφοφορίας και την ιδιωτικότητα του ψηφοφόρου σε βαθμό τέτοιο ώστε οι ηλεκτρονικές ψηφοφορίες να είναι πιο ασφαλής και από τις συμβατικές ψηφοφορίες.

Επιπλέον, είναι σαφές ότι οι ηλεκτρονικές ψηφοφορίες είναι το μέλλον των ψηφοφοριών. Πλέον, τα ηλεκτρονικά συστήματα έχουν κατακλύσει τα πάντα και βρίσκονται παντού τριγύρω μας, εξαιτίας της ακρίβειας και της ταχύτητάς τους. Σε κάθε περίπτωση, όμως, σε έναν τομέα τόσο ευαίσθητο όσο οι ψηφοφορίες οι άνθρωποι είναι ακόμη δύσπιστοι και πολλές φορές έχουν δίκιο. Ο λόγος της δυσπιστίας, υπάρχει εξαιτίας της μη κατανόησης της ακριβούς λειτουργίας ενός ηλεκτρονικού συστήματος ψηφοφοριών. Έτσι, για να μπορέσουμε να φτάσουμε στο σημείο, όλες οι ψηφοφορίες να είναι πλέον ηλεκτρονικές θα πρέπει πρώτα να υπάρξει η α-

παραίτητη εξοικείωση με τα ηλεκτρονικά συστήματα από τους ανθρώπους, ενώ θα πρέπει ταυτόχρονα να υπάρχουν και οι απαραίτητες αποδείξεις ότι η ηλεκτρονική ψηφοφορία θα είναι ασφαλής.

Σκοπός λοιπόν αυτής της εργασίας, είναι αρχικά η παρουσίαση των απαραίτητων μαθηματικών και κρυπτογραφικών μοντέλων, για να κατανοήσουμε την μετέπειτα παρουσίαση διαφόρων σύγχρονων κρυπτογραφικών συστημάτων. Στην συνέχεια, παρουσιάζεται εκτενέστερα μία μορφή ενός συστήματος ηλεκτρονικών ψηφοφοριών το οποία και αναλύουμε, με απώτερο σκοπό να θεμελιώσουμε και να υλοποιήσουμε αποδοτικά και με κρυπτογραφική ασφάλεια, ένα σύστημα ψηφοφοριών κατάταξης, σε αυτού του τύπου των ψηφοφοριών που μέχρι τώρα δεν ήταν διαθέσιμο.

Κεφάλαιο 2

Απαραίτητες Μαθηματικές και Κρυπτογραφικές Έννοιες

2.1 Γενικά Περί Κρυπτογραφίας

Όπως αναφέρθηκε και προηγουμένως η κρυπτογραφία αποτελεί αναπόσπαστο κομμάτι για την σωστή και εύρυθμη διεξαγωγή ηλεκτρονικών ψηφοφοριών. Λαμβάνοντας αυτό υπόψιν, είναι απαραίτητο να αναφέρουμε τα σημαντικότερα κρυπτογραφικά πρωτόκολλα, τα οποία βρίσκουν εφαρμογή στις ηλεκτρονικές ψηφοφορίες, και όχι μόνο. Βέβαια, για να γίνουν κατανοητά, θα πρέπει κάποιος να γνωρίζει κάποιες μαθηματικές έννοιες για να μπορέσει να κατανοήσει την λογική επαγωγή της κρυπτογραφικής ασφάλειας που προσφέρουν τα συγκεκριμένα πρωτόκολλα. Αυτές λοιπόν οι μαθηματικές έννοιες είναι κατά κύριο λόγο κάποιες αλγεβρικές γνώσεις σε συνδυασμό με κάποιες απαραίτητες γνώσεις από την θεωρία αριθμών.

2.2 Άλγεβρα

Με τον όρο άλγεβρα, εννοούμε όλες εκείνες τις απαραίτητες αλγεβρικές σχέσεις και ορισμούς που είναι απαραίτητες για την κατανόηση των κρυπτογραφικών μέσων που θα αναλυθούν αργότερα. Θα αναφερθούμε, επιγραμματικά στις παρακάτω υποενότητες στα χρήσιμα στοιχεία.

2.2.1 Θεωρία Ομάδων

Με τον όρο **ομάδα** εννοούμε, ένα σύνολο στοιχείων μαζί με μία πράξη, η οποία συνδυάζει δύο στοιχεία του συνόλου για να σχηματίσουν ένα τρίτο στοιχείο που ανήκει επίσης στο σύνολο, ικανοποιώντας ταυτόχρονα τέσσερις συνθήκες που

ονομάζονται αξιώματα της ομάδας και αναφορικά είναι η κλειστότητα, η προσεταιριστική ιδιότητα, η ταυτότητα και η αντιστρεψιμότητα. Πιο συγκεκριμένα ακολουθεί ο αυστηρός ορισμός της ομάδας.

Ορισμός 2.1 Ομάδα[18] είναι ένα σύνολο, G , μαζί με μία πράξη \bullet (νόμος της G) οι οποίες συνδυάζει οποιαδήποτε δύο στοιχεία a και b για να σχηματίσει ένα άλλο στοιχείο που συμβολίζεται με $a \bullet b$ ή απλά ab . Για να είναι ομάδα, το σύνολο και η πράξη, (G, \bullet) , πρέπει να ικανοποιούν τέσσερις ιδιότητες γνωστές ως αξιώματα των ομάδων[22]:

- **Κλειστότητα:** Για όλα τα a, b που ανήκουν στο G , το αποτέλεσμα της πράξης, $a \bullet b$, ανήκει επίσης στο G
- **Προσεταιριστική Ιδιότητα:** Για όλα τα a, b και c που ανήκουν στο G , ισχύει $(a \bullet b) \bullet c = a \bullet (b \bullet c)$.
- **Ουδέτερο στοιχείο:** Υπάρχει ένα στοιχείο e στο G , τέτοιο ώστε για κάθε στοιχείο a στο G , η εξίσωση $e \bullet a = a \bullet e = a$ να επαληθεύεται. Αυτό το στοιχείο είναι μοναδικό, και ως εκ τούτου αναφερόμαστε στο στοιχείο ταυτότητας.
- **Αντίστροφο στοιχείο:** Για κάθε a στο G , υπάρχει ένα στοιχείο a^{-1} στο G τέτοιο ώστε $a \bullet a^{-1} = a^{-1} \bullet a = e$.

Επιπλέον, αβελιανή θα καλείται η ομάδα στην οποία ισχύει πάντοτε η αντιμεταθετική ιδιότητα ($a \bullet b = b \bullet a$) Στη συνέχεια, θα παρουσιαστούν κάποιοι ορισμοί ακόμη για την εκτενέστερη κατανόηση των διαδικασιών που έπονται.

Ορισμός 2.2 Κυκλική ομάδα:[16] Μια ομάδα G θα ονομάζεται κυκλική αν υπάρχει ένα στοιχείο $g \in G$ τέτοιο ώστε κάθε στοιχείο του συνόλου G , έστω $a \in G$, να μπορεί να γραφτεί ως g^x για κάποιο $x \in \mathbf{Z}$

Ορισμός 2.3 Τάξη ομάδας: Ως τάξη μιας πεπερασμένης ομάδας G ορίζεται ο αριθμός ίσος με το πλήθος των στοιχείων της ομάδας και συμβολίζεται με $|G|$.

Ορισμός 2.4 Τάξη στοιχείου ομάδας: Η τάξη του στοιχείου a μιας πεπερασμένης ομάδας, δίνεται από το $ord(a)$, και είναι ο μικρότερος θετικός ακέραιος αριθμός x τέτοιος ώστε $a^x = 1$

Ορισμός 2.5 Υποομάδα: Για μία ομάδα G , ένα μη κενό υποσύνολο αυτής $H \subseteq G$ ονομάζεται υποομάδα του G , αν το H είναι ομάδα.

Θεώ 2.6 Το θεώρημα του Lagrange για την τάξη μιας ομάδας: Για κάθε ομάδα G και υποομάδα της H , η τάξη της H διαιρεί την τάξη της G .

Το τελευταίο θεώρημα μας δίνει ότι η τάξη κάθε στοιχείου μιας πεπερασμένης ομάδας διαιρεί την τάξη της ομάδας. Αν τώρα το g είναι γεννήτορας μιας κυκλικής ομάδας G τάξης m , τότε για $x \in \{1, \dots, m\}$, $b = g^x$ έχει τάξη $\frac{m}{gcd(m,x)}$. Έτσι, το b είναι γεννήτορας της G αν και μόνο αν $gcd(m, x) = 1$. Ως εκ τούτου, αν το m είναι πρώτος τότε κάθε στοιχείο διαφορετικό του 1, είναι γεννήτορας της G .

2.2.2 Το Z_m και οι Ιδιότητες του

Το σετ $Z_m := \{1, \dots, m-1\}$, σε συνδυασμό με την πράξη *modulo* m , φτιάχνει την αντιμεταθετική ομάδα τάξης m . Το σετ $Z_m^* := \{a \in Z_m \mid \gcd(m, a) = 1\}$ είναι το σετ των ακεραίων μεταξύ του 1 και του $m-1$ οι οποίοι είναι σχετικά πρώτοι στον m το οποίο σε συνδυασμό με τον πολλαπλασιασμό *modulo* m , σχηματίζει μία άλλη ομάδα. Στο σημείο αυτό ορίζουμε την συνάρτηση του Euler:

Ορισμός 2.7 Euler ϕ -συνάρτηση: Για θετικό ακέραιο n , το $\phi(n)$ ορίζεται ως ο αριθμός των μη αρνητικών ακεραίων οι οποίοι είναι μικρότεροι του n και σχετικά πρώτοι στον n :

$$\phi(n) = |\{a \mid 1 \leq a < n \text{ and } \gcd(a, n) = 1\}|.$$

Τα παραπάνω βρίσκουν άμεση εφαρμογή στην κρυπτογραφία ειδικά αν $m = p$ οπότε έχουμε κάποιες πολύ χρήσιμες ιδιότητες, όπως ότι κάθε στοιχείο της ομάδας Z_p^* είναι γεννήτορας της ομάδας.

2.3 Θεωρία Αριθμών

Σε αυτήν την ενότητα θα αναλυθούν τα σημαντικότερα προβλήματα της θεωρίας αριθμών, τα οποία βρίσκουν άμεση εφαρμογή στην κρυπτογραφική ασφάλεια των χρησιμοποιούμενων πρωτοκόλλων. Πιο συγκεκριμένα τα παρακάτω προβλήματα μας εξασφαλίζουν ότι η δυσκολία επίλυσης τους συνεπάγεται και κρυπτογραφική ασφάλεια.

2.3.1 Το πρόβλημα του Διακριτού Λογάριθμου (DLOG)

Το πρόβλημα Του Διακριτού Λογάριθμου (DLOG), και πιο συγκεκριμένα η δυσκολία υπολογισμού αυτού, είναι κάτι το θεμελιώδες για πολλά κρυπτογραφικά συστήματα και πρωτόκολλα. Τι είναι όμως ο Διακριτός λογάριθμος; Ακολουθεί ο επίσημος ορισμός του προβλήματος:

Ορισμός 2.8 Διακριτός Λογάριθμος[21]: Έστω ότι G μια πεπερασμένη κυκλική ομάδα και έστω g γεννήτορας αυτής. Ο διακριτός λογάριθμος ενός στοιχείου $a \in G$ δίνεται από το $\log_g a$, και είναι ο ακέραιος x , $0 \leq x < |G|$, τέτοιος ώστε $a = g^x$.

Βάση του παραπάνω ορισμού γίνεται εμφανές ότι το πρόβλημα του διακριτού λογαρίθμου είναι επί της ουσίας η δυσκολία να βρούμε το x ώστε να ισχύει ο παραπάνω ορισμός.

Γενικά αν έχουμε μια κυκλική ομάδα G με γεννήτορα g , τάξης m , και $a, b \in G$ τα παρακάτω ισχύουν ανάλογα:

- $\log_g(ab) \equiv \log_g a + \log_g b \pmod{m}$
- $\log_g(a^x) \equiv x \log_g a \pmod{m}$

- Αν h ένας άλλος γεννήτορας τότε, $\log_g a \equiv \log_h a * (\log_h g)^{-1} \pmod{m}$

Η χρησιμότητα του συγκεκριμένου προβλήματος και της δυσκολίας επίλυσης του υπό συγκεκριμένων συνθηκών, θα διαφανεί όταν αναφερθούμε εκτενέστερα στα κρυπτογραφικά συστήματα που στηρίζονται σε αυτό.

2.3.2 Το πρόβλημα Diffie-Hellman

Το πρόβλημα Diffie-Hellman ή πιο απλά DH αποτελεί επίσης ένα πρόβλημα βαρύνουσας σημασίας. Πιο συγκεκριμένα ο ακριβής ορισμός του είναι ο εξής[27]:

Ορισμός 2.9 Diffie-Hellman: Δεδομένου ενός στοιχείου g το οποίο αποτελεί γεννήτορα μιας κυκλικής ομάδας G , και με δοσμένες τις τιμές $g^u, g^v \in G$ να βρεθεί το $g^{uv} \in G$.

Βάση του παραπάνω ορισμού είναι εμφανής η σύνδεση του προβλήματος Diffie-Hellman με το αντίστοιχο του διακριτού λογαρίθμου. Πιο συγκεκριμένα αν το πρόβλημα του διακριτού λογαρίθμου είναι επιλύσιμο σε πολυωνυμικό χρόνο, τότε και το Diffie-Hellman θα είναι επίσης επιλύσιμο σε πολυωνυμικό χρόνο. Το τελευταίο είναι εμφανές αφού αν μπορούμε να υπολογίσουμε τον διακριτό λογάριθμο, τότε υπολογίζω $u = \log_g(g^u)$ και μετά υπολογίζω πολύ απλά το $(g^v)^u$.

2.3.3 Το πρόβλημα απόφασης Diffie-Hellman

Το πρόβλημα απόφασης Diffie-Hellman (DDH) μοιάζει με το κλασικό πρόβλημα και ορίζεται ως:

Ορισμός 2.10 Diffie-Hellman: Δεδομένου ενός στοιχείου g το οποίο αποτελεί γεννήτορα μιας κυκλικής ομάδας G , και με δοσμένες τις δύο παρακάτω τριάδες $g^u, g^v, g^{uv} \in G$ και $g^u, g^v, g^c \in G$ είναι υπολογιστικά μη διαχωρίσιμες.

Το παραπάνω πρόβλημα συνδέεται και αυτό με το πρόβλημα του διακριτού λογαρίθμου, και για την ακρίβεια θεωρείται πιο ισχυρή συνθήκη καθώς υπάρχουν ομάδες όπου το DDH υπολογίζεται εύκολα ενώ το πρόβλημα του διακριτού λογαρίθμου παραμένει δύσκολο.

2.4 Κρυπτογραφικά Συστήματα

Τα κρυπτογραφικά συστήματα, έχουν κάνει την εμφάνισή τους στην ιστορία της ανθρωπότητας, από την αρχαιότητα. Επί της ουσίας, χρησιμοποιήθηκαν στην προσπάθεια των ανθρώπων να ανταλλάζουν κρυφά προς τρίτους μηνύματα, με στόχο την ανταλλαγή κρίσιμων πληροφοριών. Τα κρυπτοσυστήματα χωρίζονται σε δύο μεγάλες κατηγορίες, τα συμμετρικά και τα ασύμμετρα. Συμμετρικά είναι εκείνα τα κρυπτοσυστήματα τα οποία χρησιμοποιούν κατά την διαδικασία τόσο της κρυπτογράφησης όσο και της αποκρυπτογράφησης ένα κοινό κλειδί. Για να γίνει φυσικά αυτό, είναι απαραίτητη η ανταλλαγή μεταξύ των ατόμων που θέλουν να επικοινωνήσουν ενός κρυφού κοινού κλειδιού μέσα από ένα ασφαλές κανάλι επικοινωνίας

ή με κάποιον άλλον τρόπο. Σε κάθε περίπτωση, η δυσκολία αυτού του τύπου της κρυπτογράφησης έγκειται στο γεγονός, ότι είναι απαραίτητη η ύπαρξη αυτού του κοινού κλειδιού το οποίο πρέπει σε κάθε περίπτωση να παραμείνει και κρυφό προς τρίτους. Τα συμμετρικά κρυπτοσυστήματα με την σειρά τους χωρίζονται σε δύο ακόμη μεγάλες κατηγορίες ανάλογα με τον τρόπο κρυπτογράφησης, στους αλγόριθμους Δέσμης (Block Ciphers) και στους αλγόριθμους Ροής (Stream Ciphers). Από εκεί και πέρα υπάρχουν φυσικά και τα ασύμμετρα κρυπτοσυστήματα ή αλλιώς τα κρυπτοσυστήματα δημοσίου κλειδιού. Η δημιουργία τους επέτρεψε να παρακαμφθεί το δύσκολο μέρος της ανταλλαγής των κλειδιών που χρειάζονται τα συμμετρικά κρυπτοσυστήματα. Το κυριότερο χαρακτηριστικό αυτού του τύπου των κρυπτοσυστημάτων είναι η ύπαρξη τώρα δύο τύπων κλειδιού, ενός ιδιωτικού και ενός δημοσίου. Το δημόσιο κλειδί είναι διαθέσιμο προς όλους σε αντίθεση με το ιδιωτικό που είναι μυστικό. Η κρυπτογραφική λογική που εφαρμόζεται τώρα είναι ότι, οτιδήποτε κρυπτογραφείται με το ένα από τα δύο κλειδιά μπορεί να αποκρυπτογραφηθεί μόνο από το άλλο κλειδί. Έτσι μπορεί να διασφαλιστεί μια ασφαλής επικοινωνία μεταξύ του αποστολέα και του παραλήπτη του κρυπτογραφημένου μηνύματος. Τα περισσότερα σύγχρονα κρυπτοσυστήματα για την ανταλλαγή κρίσιμων πληροφοριών, είναι βεβαίως υβριδικά, δηλαδή χρησιμοποιούν ένα κατάλληλο συνδυασμό συμμετρικών και ασύμμετρων κρυπτοσυστημάτων με σκοπό την επίτευξη της καλύτερης δυνατής ιδιωτικότητας. Στη συνέχεια αναλύονται κάποιοι από τους σημαντικότερους αλγόριθμους ασύμμετρης κρυπτογράφησης οι οποίοι είναι χρήσιμοι για την κατανόηση του συστήματος μας αλλά και γενικότερα των ηλεκτρονικών ψηφοφοριών.

2.4.1 RSA

Το κρυπτοσύστημα RSA είναι από τα πρώτα πρακτικά κρυπτοσυστήματα δημοσίου κλειδιού. Χρησιμοποιείται ευρέως για ασφαλή ανταλλαγή πληροφοριών και όπως είναι φυσικό στηρίζεται στην ύπαρξη δύο κλειδιών, του ιδιωτικού και του δημοσίου. Η πρακτική δυσκολία του να σπάσει το RSA στηρίζεται στην πρακτική δυσκολία της παραγοντοποίησης δύο μεγάλων πρώτων αριθμών (προς το παρόν ισχύει ο παραπάνω ισχυρισμός, αλλά παραμένει ανοιχτό ερώτημα). Γενικά είναι σχετικά αργός αλγόριθμος και ως εκ τούτου δεν χρησιμοποιείται άμεσα για την κρυπτογράφηση δεδομένων αλλά πολλές φορές σε υβριδική μορφή. Πως όμως λειτουργεί ο αλγόριθμος RSA; Επί της ουσίας, ο αλγόριθμος περιλαμβάνει 4 κύρια βήματα, την δημιουργία των κλειδιών, την διάθεση των κλειδιών, την κρυπτογράφηση και την αποκρυπτογράφηση. Η διαδικασία επιγραμματικά λειτουργεί ως εξής[9]:

- **Δημιουργία των κλειδιών:** Διαλέγουμε τυχαία 2 πολύ μεγάλους πρώτους αριθμούς (πρακτικά της τάξης των 1024 - 4096 bits). Στην συνέχεια υπολογίζουμε το γινόμενο τους: $n = p * q$. Βάση του ορισμού που έχουμε δώσει στην συνάρτηση του Euler βρίσκουμε ότι $\phi(n) = (p - 1) * (q - 1)$ και διαλέγουμε κατάλληλο αριθμό $e > 1$ τέτοιο ώστε $e^{\phi(n)} \equiv 1 \pmod{n}$. Τέλος υπολογίζουμε τον d τέτοιο ώστε $d \equiv e^{-1} \pmod{\phi(n)}$. Άρα τα τελικά ζεύγη των κλειδιών μας είναι τα εξής το δημόσιο: (n, e) και το ιδιωτικό: (n, d)

- **Διάθεση των κλειδιών:** Σε αυτό το βήμα είναι αναγκαίο να στείλουμε στο άτομο ή στα άτομα με τα οποία θέλουμε να επικοινωνήσουμε μυστικά, το ζεύγος του δημοσίου κλειδιού μας (n, e) . Είναι απαραίτητο το ζεύγος αυτό να φτάσει σωστά, δηλαδή το κανάλι επικοινωνίας να είναι αξιόπιστο, αλλά δεν είναι απαραίτητο να είναι ιδιωτικό. Σε κάθε περίπτωση δεν πρέπει να αποστέλλουμε ποτέ το ιδιωτικό μας κλειδί.
- **Κρυπτογράφηση:** Μετά την παραλαβή του δημοσίου κλειδιού μας, όποιος το έχει μπορεί να μας στείλει ένα κρυπτοκείμενο. Αυτό το κρυπτοκείμενο θα παραχθεί με την κάτωθι διαδικασία. Αρχικά, ας υποθέσουμε ότι κάποιος θέλει να μας στείλει ένα μήνυμα M . Το πρώτο που θα κάνει είναι να αντιστοιχήσει το M σε έναν ακέραιο m μέσω ενός ευκόλως αντιστρέψιμου πρωτοκόλλου, με την μόνη προϋπόθεση το $\gcd(m, n) = 1$. Στην συνέχεια ο αποστολέας θα υπολογίσει το κρυπτοκείμενο c ως εξής: $c \equiv m^e \pmod{n}$, και θα το αποστείλει στον προορισμό του.
- **Αποκρυπτογράφηση:** Με την λήψη του c , μπορούμε πολύ εύκολα να εξάγουμε το αρχικό m υπολογίζοντας το $c^d \equiv (m^e)^d \equiv m \pmod{n}$. Τέλος με δεδομένο το m υπολογίζουμε εύκολα το M με χρήση της αρχικής αντιστρέψιμης συνάρτησης.

2.4.2 Paillier

Το κρυπτοσύστημα Paillier είναι ακόμη ένα κρυπτοσύστημα που στηρίζεται στην ασύμμετρη κρυπτογράφηση. Η δυσκολία αυτού του αλγορίθμου στηρίζεται στην δυσκολία υπολογισμού n -στων υπολοίπων (για την ακρίβεια του DCRA. Το σχήμα είναι ομομορφικό (θα αναφερθούμε αργότερα στο τι είναι ομομορφικό), και η διαδικασία που ακολουθείται στο κρυπτοσύστημα είναι η εξής [23]:

- **Δημιουργία των κλειδιών:** Διαλέγουμε τυχαία 2 πολύ μεγάλους πρώτους αριθμούς τέτοιους ώστε: $\gcd(pq, (p-1) * (q-1)) = 1$. Στην συνέχεια υπολογίζουμε το γινόμενο τους: $n = p * q$ και το $\lambda = \text{lcm}(p-1, q-1)$ (η προηγούμενη ιδιότητα διασφαλίζεται αν τα p, q έχουν ίσο μήκος). Στην συνέχεια διαλέγουμε ένα $g \in Z_{n^2}^*$, και διασφαλίζουμε ότι το n διαιρεί την τάξη του g (μέσα από τον εξής υπολογισμό: $\mu = \left(\frac{(g^\lambda \bmod n^2) - 1}{n}\right)^{-1} \bmod n$. Το τελικό δημόσιο κλειδί είναι το (n, g) και το ιδιωτικό το (λ, μ) . Οι παραπάνω υπολογισμοί απλοποιούνται κατά πολύ αν διαλέξουμε τα p, q να έχουν ίσο μήκος, τότε αρκεί να θέσουμε $g = n + 1, \lambda = \phi(n), \mu = \phi(n)^{-1} \bmod n$.
- **Κρυπτογράφηση:** Έστω ότι τώρα έχουμε να κρυπτογραφήσουμε ένα μήνυμα M . Τότε αντιστοιχούμε καταλλήλως αυτό το M σε $m \in Z_n$ και κρυπτογραφείται ως εξής: Διαλέγουμε ένα τυχαίο $r \in Z_n^*$ και στην συνέχεια υπολογίζουμε το $c = g^m * r^n \bmod n^2$
- **Αποκρυπτογράφηση:** Έστω τώρα ότι θέλω να αποκρυπτογραφήσω το $c \in Z_{n^2}^*$, τότε το αρχικό κείμενο παράγεται μέσα από τον υπολογισμό: $m = \frac{(c^\lambda \bmod n^2) - 1}{n} * \mu \bmod n$.

2.4.3 El Gamal

Το κρυπτοσύστημα El Gamal είναι το τελευταίο που θα μελετήσουμε, και αυτό το οποίο θα χρησιμοποιηθεί στα επόμενα μέρη της παρούσας διπλωματικής εργασίας (μια παραλλαγή του για την ακρίβεια). Είναι ακόμη ένα κρυπτοσύστημα ασύμμετρης κρυπτογράφησης και στηρίζεται στο Diffie-Hellman. Χρησιμοποιείται σε πάρα πολλά συστήματα είτε αυτό είτε μικρές παραλλαγές του, και η δυσκολία του σχετίζεται με την δυσκολία του υπολογισμού του διακριτού λογαρίθμου. Η διαδικασία κρυπτογράφησης έχει 3 στάδια, την παραγωγή του κλειδιού, τον αλγόριθμο κρυπτογράφησης και την αποκρυπτογράφηση. Στην συνέχεια παρατίθεται ο αλγόριθμος, με κάποιες μικρές τροποποιήσεις από τον αρχικό μαθηματικό ορισμό για ευκολότερη κατανόηση και καταλληλότερη υλοποίηση [26].

- **Δημιουργία των κλειδιών:** Διαλέγουμε δύο πολύ μεγάλους πρώτους αριθμούς p, q τέτοιους ώστε το q να διαιρεί το $p - 1$, και διαλέγουμε έναν γεννήτορα g της τάξεως- q υποομάδας του Z_p^* . Στην συνέχεια για την ολοκλήρωση της διαδικασίας παραγωγής του κλειδιού, διαλέγουμε ένα τυχαίο $x \in Z_q$. Υπολογίζουμε το $y = g^x \pmod{p}$. Τέλος το δημόσιο κλειδί είναι το y , το οποίο και δημοσιεύεται σε συνδυασμό με την περιγραφή της ομάδας G , και των κατάλληλων παραμέτρων της.
- **Κρυπτογράφηση:** Κατά την κρυπτογράφηση όποιος θέλει να κρυπτογραφήσει ένα μήνυμα αρχικά, μετασχηματίζει το προς κρυπτογράφηση μήνυμα έστω M σε ένα στοιχείο της κυκλικής ομάδας, έστω m . Στην συνέχεια ο αποστολέας του κρυπτομηνύματος, θα διαλέξει ένα τυχαίο $r \in Z_q$, και θα υπολογίσει τα $c_1 = g^r \pmod{p}$, $c_2 = m * y^r \pmod{p}$, τα οποία θα είναι και το τελικό κρυπτοκείμενο (c_1, c_2) .
- **Αποκρυπτογράφηση:** Για να αποκρυπτογραφήσει κάποιος ένα μήνυμα κάνει τα εξής: υπολογίζει αρχικά το $s = c_1^x$. Στην συνέχεια για να αποκτήσει κάποιος το τελικό μήνυμα πρέπει να υπολογίσει το $m = c_2 * s^{-1} \pmod{p}$, και τέλος αντιστοιχεί το m στο κατάλληλο M .

Περαιτέρω ανάλυση για την ασφάλεια του παραπάνω συστήματος, όσο και για την κατάλληλη παραλλαγή του για την εφαρμογή ηλεκτρονικών ψηφοφοριών, θα υπάρξει στην επόμενη θεματική ενότητα της εργασίας, όπου και θα αναλυθεί εκτενέστερα το σύστημα πάνω στο οποίο θα πραγματοποιηθεί η πρακτική υλοποίηση των ψηφοφοριών κατάταξης.

2.5 Ομομορφικά Συστήματα και Mixnets

Στις προηγούμενες ενότητες αναλύθηκαν διάφορα μαθηματικά και κρυπτογραφικά μοντέλα τα οποία είναι χρήσιμα στην κατανόηση της λειτουργίας των συστημάτων που υποστηρίζουν ηλεκτρονικές ψηφοφορίες. Τα συστήματα αυτά λοιπόν στην πλειονότητά τους είναι είτε ομομορφικά συστήματα ψηφοφοριών είτε συστήματα Mixnets. Σε αυτήν την ενότητα θα αναλυθούν τα παραπάνω δύο κύρια συστήματα ως προς τις ιδιότητές τους, για το τι προσφέρει το κάθε ένα αλλά και για τις ομοιότητες και διαφορές τους.

2.5.1 Ομομορφικά Συστήματα

Όταν μιλάμε για ομομορφικά συστήματα, εννοούμε ένα σύστημα[12] το οποίο μας επιτρέπει όταν κάνουμε συγκεκριμένες πράξεις μεταξύ των κρυπτοκειμένων και παράγουμε ένα τελικό αποτέλεσμα, τότε με την αποκρυπτογράφηση του παραπάνω αποτελέσματος η πράξη που έγινε μεταξύ των κρυπτοκειμένων να έχει μεταφερθεί στα αρχικά κείμενα. Επί της ουσίας δηλαδή η κυρίαρχη ιδιότητα που διακρίνει τα ομομορφικά συστήματα είναι η εξής: $Enc(m_1) \circ Enc(m_2) = Enc(m_1 \circ m_2)$, όπου $Enc()$ είναι η διαδικασία κρυπτογράφησης. Τα ομομορφικά συστήματα με την σειρά τους μπορούν να είναι πλήρως ομομορφικά, δηλαδή να είναι ομομορφικά για όλες τις πιθανές πράξεις, ή εν μέρει ομομορφικά, δηλαδή να είναι ομομορφικά μόνο ως προς μία(ή και περισσότερες αλλά όχι όλες) πράξη. Η σημασία των παραπάνω συστημάτων, ειδικά σε περιπτώσεις ψηφοφοριών, γίνεται εμφανής αν αναλογιστεί κάποιος ότι με χρήση αυτών των συστημάτων δεν είναι πλέον απαραίτητη η αποκρυπτογράφηση ένα προς ένα όλων των κρυπτοκειμένων. Αυτό διαφαίνεται από το γεγονός ότι σε περιπτώσεις ψηφοφοριών, αυτό που μας ενδιαφέρει είναι για παράδειγμα το άθροισμα των ψήφων και όχι μία προς μία όλες οι ψήφοι. Βεβαίως, για να υλοποιηθούν σωστά τα ομομορφικά συστήματα σε ηλεκτρονικές ψηφοφορίες είναι απαραίτητο να διασφαλίζονται κάποιες προϋποθέσεις, οι οποίες θα μελετηθούν εκτενέστερα στο κεφάλαιο που θα αναλυθεί το σύστημα ψηφοφοριών Helios. Προς το παρόν, αυτό που αρκεί να πούμε είναι ότι τα ομομορφικά συστήματα παρέχουν ασφάλεια ως προς την ταυτότητα των ψηφοφόρων, καθώς όπως προείπαμε δεν είναι απαραίτητο να αποκρυπτογραφήσουμε την ατομική ψήφο ενός ψηφοφόρου, αλλά αποκρυπτογραφούμε όλες τις ψήφους μαζί αφού τις έχουμε συνδυάσει μέσω των ομομορφικών πράξεων που μας παρέχει το σύστημα μας. Τέλος, σε σχέση με τα υπόλοιπα συστήματα τα ομομορφικά είναι αρκετά πιο γρήγορα αν σκεφτούμε ότι στην πραγματικότητα η πλειονότητα του χρόνου σπαταλάται κατά την διαδικασία της αποκρυπτογράφησης, άρα ως εκ τούτου βάση του παρόντος συστήματος, η αποκρυπτογράφηση των ψήφων θα γίνει μονάχα μία φορά, κατά το τέλος της ψηφοφορίας.

2.5.2 Mixnets

Πέρα από τα ομομορφικά συστήματα, το άλλο κυρίαρχο σύστημα στις ηλεκτρονικές ψηφοφορίες είναι τα Mixnets[15] ή αλλιώς τα Mix networks. Τα συγκεκριμένα συστήματα στηρίζονται σε ένα δίκτυο από πολλαπλά επίπεδα με σέρβερς, όπου το κάθε επίπεδο δέχεται σαν είσοδο την έξοδο του προηγούμενου επιπέδου σε τυχαία σειρά. Δηλαδή επί της ουσίας, ένα επίπεδο λαμβάνει σαν είσοδο κάποια μηνύματα τα οποία και ανακατεύει τυχαία. Στην συνέχεια τα επανακρυπτογραφεί και τα στέλνει στο επόμενο επίπεδο κοκ. Ένα απλό παράδειγμα φαίνεται στο παρακάτω σχήμα.

Γίνεται λοιπόν σαφής η χρησιμότητα του παραπάνω συστήματος στις ψηφοφορίες, αφού ο κάθε ψηφοφόρος αρχικά κρυπτογραφεί την ψήφο του, και στην συνέχεια την στέλνει στο δίκτυο, όπου επανακρυπτογραφείται και ταυτόχρονα παίρνει μια τυχαία θέση. Συνέπεια των Mixnets είναι ότι πλέον όταν γίνεται η αποκρυπτογράφηση δεν μπορούμε να αντιστοιχίσουμε την ψήφο στον ψηφοφόρο που την

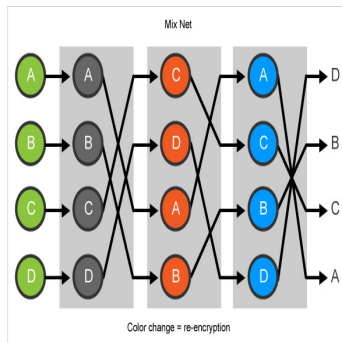


Figure 2.1: mixnet example

κατέθεσε. Βέβαια, τέτοια σχήματα είναι επιρρεπή σε επιθέσεις, αφού πολλές φορές αν δίνονται πολλές επιλογές μπορεί να εφαρμοστεί κάποιου είδους επιβολής του τι θα ψηφίσει ένας ψηφοφόρος και αν κατά το τέλος της διαδικασίας γίνουν γνωστές όλες οι ψήφοι, ο κακόβουλος μπορεί να ελέγξει αν ο ψηφοφόρος που υπέστη την επίθεση πράγματι ψήφισε αυτό που του επιβλήθη. Πέρα όμως από τις διάφορες επιθέσεις που μπορούν να εφαρμοστούν, πρέπει να αναφέρουμε ότι συγκριτικά με τα ομομορφικά συστήματα προσφέρουν μια αυξημένη ιδιωτικότητα, αλλά με αρκετό κόστος ως προς την ταχύτητα των υπολογισμών, καθώς στα Mixnets είναι απαραίτητη η αποκρυπτογράφηση όλων των ψήφων.

Ακόμη, η κρυπτογράφηση που εφαρμόζει ο κάθε σέρβερ είναι κάποιου είδους κρυπτογράφηση δημοσίου κλειδιού όπως έχουμε αναφέρει. Προφανώς, από σύστημα σε σύστημα διαφέρει η μέθοδος που γίνεται το ανακάτεμα των ψήφων όπως και η μέθοδος κρυπτογράφησης τους, αλλά σε όλα τα συστήματα που στηρίζονται σε Mixnets η λογική και οι βασικές αρχές παραμένουν οι ίδιες και είναι αυτές που αναφέρουμε προηγουμένως. Εντούτοις παρά την μορφή αυτών, υπάρχει επίσης μία τροποποιημένη έκδοση του Helios, το σύστημα Zeus[1], το οποίο έχει αναπτυχθεί από Ελληνική ομάδα, τροποποιώντας το ομομορφικό Helios σε σύστημα ψηφοφοριών με Mixnet. Τέλος, αναλυτικότερη περιγραφή μεθόδων μεταθέσεων/τυχαιοποίησης και ο τρόπος με τον οποίο γίνεται η επανακρυπτογράφηση θα αναφερθούν σε μετέπειτα κεφάλαιο της παρούσας διπλωματικής εργασίας (στο κεφάλαιο της νέας τροποποιημένης υλοποίησης), όπου και θα γίνει η χρήση ενός απλού Mixnet στα πλαίσια ενός ομομορφικού συστήματος για να αποδείξουμε κάποιες απαραίτητες σχέσεις.

2.6 Ασφάλεια Κρυπτοσυστημάτων

Αναφερθήκαμε προηγουμένως σε διάφορα κρυπτοσυστήματα, και στο πόσο δύσκολα ή εύκολα μπορεί να τα σπάσει κάποιος, αλλά ποτέ δεν δώσαμε ακριβή ορισμό για το τι σημαίνει ότι ένα κρυπτοσύστημα είναι ασφαλές. Σκοπός αυτής της ενότητας λοιπόν, είναι η θεμελίωση της έννοιας της ασφάλειας, μέσα από την παρουσίαση των πιθανών ειδών ασφάλειας που μπορεί να προσφέρει ένα κρυπτοσύστημα. Στην

συνέχεια παρουσιάζονται τα πιθανά είδη ασφαλείας.

2.6.1 Σημασιολογική Ασφάλεια

Όταν λέμε ότι ένα κρυπτοσύστημα διαθέτει σημασιολογική ασφάλεια (σ. semantic security [25]) εννοούμε ότι αν διαθέτουμε ένα πιθανοτικό, πολυωνυμικού χρόνου αλγόριθμο, τότε δεδομένου ενός κρυπτοκειμένου ενός συγκεκριμένου μηνύματος m , και το μήκος του μηνύματος αυτού, τότε δεν μπορούμε να εξάγουμε καμμία πληροφορία για το μήνυμα με πιθανότητα μη-αμέλητα μεγαλύτερης από οποιονδήποτε άλλο πιθανοτικό, πολυωνυμικού χρόνου αλγόριθμο ο οποίος διαθέτει πρόσβαση μόνο στο μήκος του μηνύματος. Επί της ουσίας, αυτό που μας δίνει η σημασιολογική ασφάλεια είναι ότι η γνώση του κρυπτοκειμένου και του μήκους ενός μηνύματος δεν αποκαλύπτει καμμία επιπλέον πληροφορία για το μήνυμα η οποία μπορεί να εξαχθεί εύκολα. Προσοχή, ο παραπάνω ορισμός δεν σημαίνει ότι το κρυπτοκείμενο δεν αποκαλύπτει καμμία απολύτως πληροφορία για το αρχικό κείμενο, αυτό που σημαίνει είναι ότι η πληροφορίες που αποκαλύπτει δεν μπορούν να εξαχθούν με εφικτό τρόπο.

Στις επόμενες υποενότητες η ασφάλεια θα λάβει την μορφή της μη-διαχωρισιμότητας [13], δηλαδή θα θεωρούμε ένα σύστημα ασφαλές αν και μόνο αν, κανένας αντίπαλος δεν θα μπορεί να βρει το αρχικό κείμενο με δεδομένο ένα κρυπτοκείμενο, με πιθανότητα (αρκετά) μεγαλύτερη από έναν αντίπαλο που επιλέγει τυχαία το αρχικό κείμενο

2.6.2 Ασφάλεια Επιλεγμένου Αρχικού Κειμένου IND-CPA

Με τον όρο CPA εννοούμε ένα μοντέλο κρυπταναλυτικής επίθεσης κατά το οποίο ο επιτιθέμενος μπορεί να αποκτήσει κρυπτοκείμενα για τυχαία αρχικά κείμενα, με σκοπό να αποκτήσει πληροφορίες που μειώνουν την ασφάλεια του κρυπτογραφικού σχήματος. Η σημασία του παραπάνω σχήματος γίνεται εμφανής σε κρυπτοσυστήματα δημοσίου κλειδιού όπου το δημόσιο κλειδί γίνεται γνωστό και οποισδήποτε μπορεί να κρυπτογραφήσει ό,τι θέλει. Για τα παραπάνω, η ασφάλεια IND-CPA σημαίνει ότι με δεδομένο έναν αλγόριθμο ασύμμετρης κρυπτογράφησης ορίζεται από το ακόλουθο παιχνίδι μεταξύ ενός αντιπάλου και ενός προκαλών, με τον αντίπαλο να είναι μια πολυωνυμικού χρόνου μηχανή Turing.

Θεωρούμε ότι $Enc(p_k, m)$ είναι η κρυπτογράφηση του μηνύματος m με δημόσιο κλειδί p_k και το παιχνίδι έχει ως εξής:

- Ο προκαλών παράγει το ζεύγος (p_k, s_k) , δημοσιεύει το p_k και διατηρεί κρυφό το s_k .
- Ο αντίπαλος στην συνέχεια μπορεί να πραγματοποιήσει πολυωνυμικού αριθμού κρυπτογραφήσεις, ή άλλων πράξεων που επιθυμεί, και με το πέρας αυτών διαλέγει τυχαία 2 μηνύματα (ίδιου μήκους) m_0, m_1 και τα στέλνει στον διεκδικητή
- Ο προκαλών με την σειρά του διαλέγει τυχαία ένα $m_i \in \{m_0, m_1\}$, το

κρυπτογραφεί($Enc(p_k, m_i)$) και στέλνει πίσω το αποτέλεσμα στον αντίπαλο.

- Τέλος ο αντίπαλος μπορεί να κάνει επιπλέον πράξεις ή κρυπτογραφήσεις που αυτός επιθυμεί, και μετά δίνει μια τιμή για το $i \in \{0, 1\}$

Λέμε ότι το σύστημα διαθέτει IND-CPA αν οποιοσδήποτε πιθανοτικός πολυωνυμικού χρόνου αντίπαλος μπορεί να επιλέξει την σωστή απάντηση με αμελητέα μεγαλύτερη πιθανότητα από κάποιον που επιλέγει τυχαία. Με τον όρο αμελητέα εννοούμε $\epsilon(k)$ τέτοιο ώστε $\epsilon(k) < \frac{1}{\text{poly}(k)}$, $k > k_0$. Το παραπάνω σχήμα μπορεί να μετασχηματιστεί και για συμμετρικές κρυπτογράφησης αλλά κάτι τέτοιο δεν μας ενδιαφέρει στα πλαίσια αυτής της εργασίας.

2.6.3 Ασφάλεια Επιλεγμένου Κρυπτοκειμένου IND-CCA

Σε περίπτωση επίθεσης CCA ένας επιτιθέμενος μπορεί να αποκτήσει αρχικά κείμενα, μέσα από κάποια επιλεγμένα κρυπτοκείμενα. Πιο συγκεκριμένα ο επιτιθέμενος μπορεί να εισάγει ένα ή και περισσότερα κρυπτοκείμενα και να πάρει τα αποτελέσματα που επιθυμεί. Τώρα όταν μιλάμε για ασφάλεια IND-CCA ο ορισμός είναι παρόμοιος με εκείνον για IND-CPA, μόνο που τώρα ο αντίπαλος έχει και πρόσβαση και σε ένα μαντείο αποκρυπτογραφήσεων όπου μπορεί να αποκρυπτογραφεί τυχαία κρυπτοκείμενα. Το παιχνίδι είναι πάλι το εξής:

- Ο προκαλών παράγει το ζεύγος (p_k, s_k) , δημοσιεύει το p_k και διατηρεί κρυφό το s_k .
- Ο αντίπαλος στην συνέχεια μπορεί να πραγματοποιήσει πολυωνυμικού αριθμού κρυπτογραφήσεις, ή άλλων πράξεων που επιθυμεί, αλλά τώρα μπορεί επιπλέον να επικαλείται και το μαντείο αποκρυπτογραφήσεων που διαθέτει για τυχόν αποκρυπτογραφήσεις που επιθυμεί, και με το πέρας αυτών διαλέγει τυχαία 2 μηνύματα(ίδιου μήκους) m_0, m_1 και τα στέλνει στον διεκδικητή
- Ο προκαλών με την σειρά του διαλέγει τυχαία ένα $m_i \in \{m_0, m_1\}$, το κρυπτογραφεί($Enc(p_k, m_i)$) και στέλνει πίσω το αποτέλεσμα στον αντίπαλο.
- Τέλος ο αντίπαλος μπορεί να κάνει επιπλέον πράξεις ή κρυπτογραφήσεις που αυτός επιθυμεί, και μετά δίνει μια τιμή για το $i \in \{0, 1\}$

Λέμε ότι το σύστημα διαθέτει IND-CCA αν οποιοσδήποτε πιθανοτικός πολυωνυμικού χρόνου αντίπαλος μπορεί να επιλέξει την σωστή απάντηση με αμελητέα μεγαλύτερη πιθανότητα από κάποιον που επιλέγει τυχαία.

2.6.4 Ασφάλεια Τροποποιήσιμου Επιλεγμένου Κρυπτοκειμένου IND-CCA2

Ισχύει ότι ακριβώς και με το πρότυπο IND-CCA με την διαφορά ότι τώρα ο επιτιθέμενος αφού λάβει την κρυπτογράφηση του m_i έχει την δυνατότητα να κάνει

επιπλέον αποκρυπτογραφήσεις κρυπτοκειμένων με την μόνη προϋπόθεση προφανώς να μην αποκρυπτογραφήσει το ίδιο το κρυπτοκειμένο που του εστάλη για να μαντέψει το αρχικό κείμενο.

2.7 Σχήματα Δέσμευσης

Ένα σχήμα δέσμευσης μας επιτρέπει να δεσμευτούμε σε μία συγκεκριμένη τιμή (ή κατάσταση) ενώ διατηρούμε κρυφή την τιμή αυτή προς τους υπολοίπους, και εν τέλει να δίνεται η δυνατότητα στην πλευρά που δεσμεύτηκε να αποκαλύπτει την τιμή και στους υπολοίπους να πραγματοποιούν τον έλεγχο αν πράγματι αυτή η τιμή είχε επιλεγθεί. Προφανώς είναι απαραίτητο, αφενός η πλευρά που δεσμεύτηκε να μην μπορεί να αλλάξει την τιμή στην οποία δεσμεύτηκε και αφετέρου να μην υπάρχει διαρροή πληροφορίας για την τιμή στην οποία υπήρξε η δέσμευση. Στην κρυπτογραφία τα σχήματα δέσμευσης είναι ιδιαίτερα σημαντικά σε διάφορους τομείς ενώ και στα πλαίσια των ηλεκτρονικών ψηφοφοριών η σημασία τους είναι εξαιρετική. Τέλος, ένα σχήμα δέσμευσης αποτελείται από δύο κυρίαρχες φάσεις, από την φάση της δέσμευσης στην συγκεκριμένη τιμή, όπου αποφασίζουμε να δεσμευτούμε σε μία τιμή η οποία δεν θα αλλάξει, και την φάση της αποκάλυψης, όπου και αποκαλύπτεται η τιμή στην οποία υπήρξε η δέσμευση και ελέγχεται από τους υπόλοιπους ενδιαφερόμενους. Στην συνέχεια παρουσιάζονται τα κυριότερα σχήματα δέσμευσης.

2.7.1 Συναρτήσεις Hash

Μία συνάρτηση κατακερματισμού (Hash function) είναι μία συνάρτηση η οποία μπορεί να αντιστοιχεί δεδομένα τυχαίου μεγέθους σε δεδομένα σταθερού μεγέθους. Οι συναρτήσεις κατακερματισμού έχουν μεγάλη χρησιμότητα σε πολλές εφαρμογές όπως για παράδειγμα οι πίνακες κατακερματισμού, οι οποίοι χρησιμοποιούνται για ταχεία αναζήτηση δεδομένων. Βέβαια, εμάς στα πλαίσια αυτής της εργασίας μας ενδιαφέρουν οι συναρτήσεις κατακερματισμού στα πλαίσια της κρυπτογραφίας. Η κυριότερη ιδιότητα μίας συνάρτησης κατακερματισμού είναι, όπως αναφέρθηκε η αντιστοίχιση δεδομένων τυχαίου μεγέθους σε δεδομένα σταθερού μεγέθους, μέσω μίας συνάρτησης η οποία να είναι μη-αντιστρέψιμη. Όταν λέμε μια συνάρτηση (έστω H) μη-αντιστρέψιμη εννοούμε ότι εννοούμε ότι ενώ μπορούμε με δεδομένο x να υπολογίσουμε το $H(x)$, να είναι αδύνατο με δεδομένο το $H(x)$ να υπολογίσουμε το x . Ακόμη όπως φαίνεται, θα πρέπει ο υπολογισμός και του $H(x)$ να είναι σχετικά απλός και γρήγορος για να έχουμε μια επιθυμητή συνάρτηση κατακερματισμού. Για την ακρίβεια η τέλεια συνάρτηση κατακερματισμού θα είχε τα εξής χαρακτηριστικά:

- Ταχύτητα στον υπολογισμό της τιμής κατακερματισμού δηλαδή του $H(x)$
- Να είναι αδύνατο (πρακτικά πολύ δύσκολο) να υπολογιστεί η αντίστροφη της H (εκτός προφανώς και αν δοκιμασθούν όλες οι πιθανές τιμές)
- Μικρές αλλαγές στο μήνυμα θα πρέπει να αλλάζουν πολύ την τιμή που προκύπτει από την συνάρτηση κατακερματισμού για τις δύο περιπτώσεις

- Να είναι αδύνατο (πρακτικά πολύ δύσκολο) να βρεις δύο διαφορετικά μηνύματα έστω (x_1, x_2) τέτοια ώστε $H(x_1) = H(x_2)$

Βάση των παραπάνω έχουν καταφέρει να αναπτυχθούν διάφορες πολύ αποδοτικές συνάρτησης κατακερματισμού (πχ SHA1, SHA2 κοκ). Τέλος, οι συναρτήσεις κατακερματισμού πέρα από την απευθείας εφαρμογή τους σε πολλά συστήματα (ακόμη και στο δικό μας), βρίσκουν εφαρμογή ως συστατικά μέρη για να φτιαχτούν κάποια πιο εξελιγμένα κρυπτογραφικά εργαλεία πχ. HMAC , μηχανές τυχαίων αριθμών και πολλά άλλα.

2.7.2 Ψηφιακές Υπογραφές

Η ψηφιακή υπογραφή είναι ένα τρόπος απόδειξης της αυθεντικότητας ενός μηνύματος. Μια σωστή ψηφιακή υπογραφή δίνει σε αυτόν που θα λάβει το μήνυμα την δυνατότητα και την σιγουριά να πιστεύει ότι το μήνυμα δημιουργήθηκε από αυτόν τον αποστολέα που περιμένουμε, ότι ο αποστολέας δεν μπορεί να αρνηθεί ότι έστειλε το μήνυμα και τέλος ότι το μήνυμα δεν αλλάχθηκε. Όπως είναι φανερό ο ψηφιακές υπογραφές βρίσκουν εφαρμογή σε κρυπτογραφικά σχήματα καθώς προσφέρουν αποδείξεις γνησιότητας και αυθεντικότητας. Πιο φορμαλιστικά μια ψηφιακή υπογραφή είναι ένα σχήμα το οποίο περιλαμβάνει τρεις αλγόριθμους:

- Έναν αλγόριθμο παραγωγής κλειδιού, ο οποίος διαλέγει ένα ιδιωτικό κλειδί (s_k) από ένα εύρος πιθανόν κλειδιών, ενώ ταυτόχρονα παράγεται και ένα δημόσιο κλειδί (p_k).
- Έναν αλγόριθμο υπογραφής, ο οποίος δεδομένου ενός μηνύματος και ενός δημοσίου κλειδιού παράγει την υπογραφή.
- Έναν αλγόριθμο επαλήθευσης υπογραφής, ο οποίος δεδομένου ενός μηνύματος, ενός δημοσίου κλειδιού και μιας υπογραφής, μπορεί να αποδείξει ότι η υπογραφή είναι είτε έγκυρη είτε άκυρη.

Βάση των παραπάνω μπορεί κανείς εύκολα να αντιληφθεί τον τρόπο με τον οποίο λειτουργούν οι ψηφιακές υπογραφές. Η γενική ιδέα λοιπόν με τον οποίο λειτουργούν οι ψηφιακές υπογραφές είναι η κάτωθι. Αρχικά, με χρήση ενός ασύμμετρου κρυπτογραφικού μοντέλου (πχ El gamal, RSA κτλ.) παράγουμε τα δύο κλειδιά που χρειάζεται το πρώτο στάδιο. Στην συνέχεια κρυπτογραφούμε με το δικό μας ιδιωτικό κλειδί που παράξαμε παραπάνω, το κείμενο και αυτό που προκύπτει είναι η υπογραφή. Για την ακρίβεια, προτιμούμε να μην κρυπτογραφήσουμε το ίδιο το κείμενο με την μία, αλλά πρώτα περνάμε το κείμενο από μία συνάρτηση κατακερματισμού και κρυπτογραφούμε το αποτέλεσμα της. Ο λόγος που συμβαίνει το τελευταίο είναι πολύ απλά γιατί δεν θέλουμε πάντα, όποιος έχει πρόσβαση στο δημόσιο κλειδί μας να έχει πρόσβαση και στο κείμενο, και με αυτόν τον τρόπο (κρυπτογραφώντας δηλαδή το αποτέλεσμα της συνάρτησης κατακερματισμού) πετυχαίνουμε πρακτικά το ίδιο αποτέλεσμα, δηλαδή όποιος έλαβε το κείμενο να μπορεί να εξακριβώσει την γνησιότητα του, σε συνδυασμό με το ότι κανείς κακόβουλος να μην έχει πρόσβαση στο αρχικό κείμενο. Ένας ακόμη λόγος που μας βολεύει να κρυπτογραφούμε την τιμή που προκύπτει από την εφαρμογή της

συνάρτησης κατακερματισμού στο κείμενο και όχι το ίδιο το κείμενο, ακόμη και αν το κείμενο είναι γενικά γνωστό και δεν μας ενδιαφέρει η ιδιωτικότητα, είναι ότι οι συναρτήσεις κατακερματισμού παράγουν σταθερού μεγέθους αντικείμενα τα οποία είναι και πολύ πιο εύκολα διαχειρίσιμα(κρυπτογραφούνται/αποκρυπτογραφούνται πιο εύκολα) από το να αναγκάζομαστε να εφαρμόζουμε πράξεις πάνω σε κείμενα τυχαίου και συχνά μεγάλου μεγέθους.

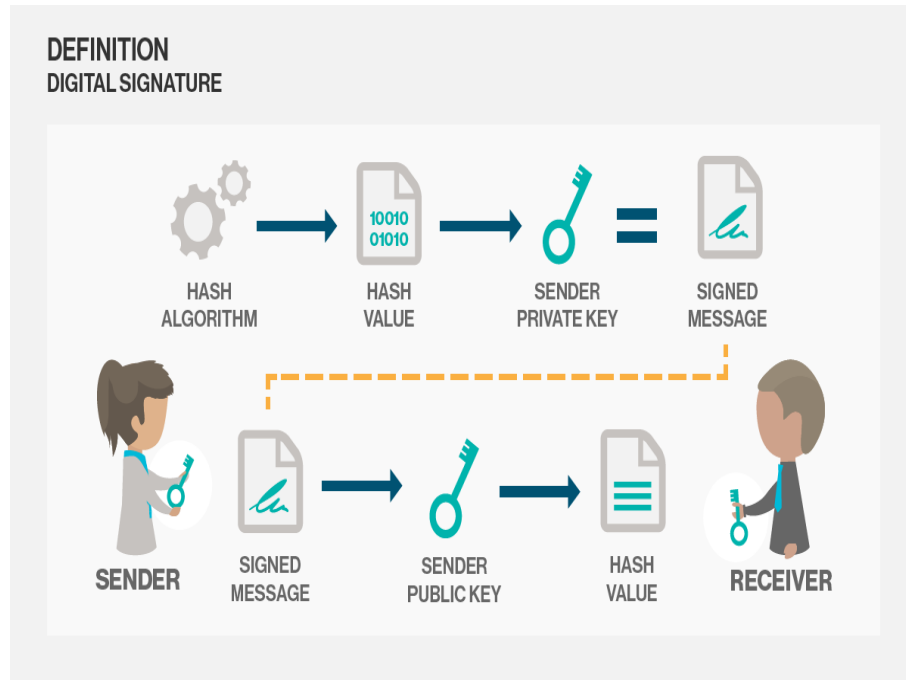


Figure 2.2: digital signature example

2.7.3 Τυφλές Υπογραφές

Οι τυφλές υπογραφές αναφερθηκαν πρώτη φορά από τον Chaum[14] και είναι το ανάλογο του να υπογράψεις κάτι με το χέρι με κλειστά μάτια, δηλαδή να μην βλέπεις τι υπογράφεις. Επί της ουσίας λοιπόν επιτρέπουν σε κάποιον να λάβει υπογραφή για ένα μήνυμα, υπογραφή η οποία θα γίνεται αποδεκτή από τον οποιοδήποτε τρίτο, αλλά ο υπογράφων δεν θα έχει δει το κείμενο που υπογράφει. Βεβαίως κάτι τέτοιο, ενώ φαίνεται υλοποιήσιμο στον πραγματικό κόσμο αν απλώς κλείσεις τα μάτια σου και υπογράψεις ό,τι σου δοθεί, στον ψηφιακό κόσμο προφανώς δεν είναι πραγματοποιήσιμο αφού για να κάνεις τους υπολογισμούς πάνω στο κείμενο. Για να επιτευχθεί λοιπόν αυτό, αντί να δίνεται στον υπογράφων το κείμενο προς υπογραφή, του δίνεται ένα κείμενο το οποίο έχει πρώτα συνδυαστεί με έναν παράγοντα 'τυφλότητας', και στην συνέχεια ο υπογράφων υπογράφει το συγκεκριμένο μετασχηματισμένο κείμενο κατά τα γνωστά, και παράγεται η υπογραφή. Η υπογραφή

λοιπόν αυτή σε συνδυασμό με τον παραπάνω παράγοντα, αποτελούν μια έγκυρη υπογραφή την οποία μπορεί να ελέγξει όποιος έχει πρόσβαση στις απαραίτητες πληροφορίες.

Η σημασία των τυφλών υπογραφών, είναι ιδιαίτερα σημαντική στις ηλεκτρονικές ψηφοφορίες που μελετάμε στην παρούσα διπλωματική. Αυτό συμβαίνει γιατί στις ψηφοφορίες, είναι αναγκαίο να έχουμε ψηφίσει σωστά, και η ψήφος μας να είναι μυστική. Συνεπώς, με ένα σχήμα τυφλών υπογραφών, κάτι τέτοιο είναι θεωρητικά(αλλά και πρακτικά) επιτεύξιμο με τις κατάλληλες βέβαια προσαρμογές.

2.8 Αποδείξεις Μηδενικής Γνώσης

Παραπάνω αναφέρθηκαν διάφορα συστήματα τα οποία μπορούν να διασφαλίσουν τις συνθήκες οι οποίες εξασφαλίζουν ότι ένας ψηφοφόρος ψήφισε ορθώς, σε συνδυασμό με την διατήρηση της ατομικότητας και της ιδιωτικότητας της ψήφου. Υπό το πρίσμα λοιπόν αυτής της ανάγκης, δηλαδή την ανάγκη απόδειξης ενός ισχυρισμού χωρίς να αποκαλύπτεις τον ίδιο τον ισχυρισμό δημιουργήθηκαν οι αποδείξεις μηδενικής γνώσης. Πιο φορμαλιστικά, θα λέμε ότι ένα διαδραστικό πρωτόκολλο διαθέτει (τέλεια ή στατιστική-υπολογιστική) μηδενική γνώση, αν για κάθε πολυωνυμικού χρόνου επαληθευτής V , υπάρχει ένας πολυωνυμικού χρόνου προσομοιωτής S_V τέτοιος ώστε τα αποτελέσματα του S_V και τα αποτελέσματα από την εφαρμογή του πρωτοκόλλου από έναν P (όπου P θα είναι αυτός που θέλει να αποδείξει την πρόταση) και έναν V να είναι μη διαχωρίσιμα. Το παραπάνω σημαίνει ότι ο επαληθευτής V δεν θα μπορεί να εξάγει καμμία πληροφορία για την τιμή/πρόταση κατά ή μετά την εκτέλεση του διαδραστικού πρωτοκόλλου. Σε κάθε περίπτωση θα πρέπει να έχουμε διαδραστικό μοντέλο μεταξύ ενός που θέλει να αποδείξει έναν ισχυρισμό, και ενός επαληθευτή, με απαραίτητη προϋπόθεση αν αυτός ο ισχυρισμός περιλαμβάνει την γνώση ενός αγνώστου παράγοντα, η απόδειξη να μην είναι αναπαράξιμη από κανέναν πέρα από αυτόν που γνωρίζει το μυστικό(προφανώς δηλαδή και ούτε από τον επαληθευτή). Βεβαίως, πολλές φορές χρειάζονται οι αποδείξεις να μην είναι διαδραστικές, αλλά και για αυτό υπάρχει λύση μέσω της εφαρμογής κάποιων μετασχηματισμών, και υπό την ισχύ ορισμένων συνθηκών, οι διαδραστικές αποδείξεις δύναται να αντικατασταθούν από μη-διαδραστικές.

Στην συνέχεια είναι χρήσιμο να αναφέρουμε τι ακριβώς πρέπει να διασφαλίζουν οι αποδείξεις μηδενικής γνώσης καθώς και επίσης την γενική μορφή τους. Οι αποδείξεις μηδενικής γνώσης πρέπει να ικανοποιούν τρεις ιδιότητες[8]:

- **Πληρότητα:** Αυτό σημαίνει ότι αν η πρόταση είναι αληθής, τότε ένας ειλικρινής αποδεικτής P θα πείθει έναν ειλικρινή επαληθευτή V με πολύ μεγάλη πιθανότητα για την αλήθεια της πρότασης.
- **Ορθότητα:** Σε περίπτωση ψευδούς πρότασης, κανένας κακόβουλος αποδεικτής P δεν θα μπορεί να πείσει έναν ειλικρινή επαληθευτή V , ότι η πρόταση είναι σωστή, παρά μόνο με μικρή πιθανότητα.
- **Μηδενική γνώση:** Αν η πρόταση είναι ορθή κανένας κακόβουλος επαληθευτής V δεν θα μπορεί να εξάγει καμμία πληροφορία για τον ισχυρισμό πέρα από το ότι ο ισχυρισμός στέχει.

Στα πλαίσια λοιπόν της μηδενικής γνώσης εφαρμόζεται το πρωτόκολλο με τον προσομοιωτή που αναφέρθηκε παραπάνω. Βάση τώρα της μηδενικής γνώσης που προσφέρει το εκάστοτε πρωτόκολλο ανάλογα με τις συνθήκες και τον τρόπο εφαρμογής υπάρχουν τα εξής επίπεδα μηδενικής γνώσης όπως φαίνεται και στον αρχικό ορισμό που δόθηκε στην αρχή:

- **Τέλεια μηδενική γνώση:** Τα πραγματικά και τα προσομοιωμένα αποτελέσματα, είναι μη-διαχωρίσιμα από οποιονδήποτε τρίτο όση υπολογιστική δύναμη και να διαθέτει, δηλαδή επί της ουσίας έχουν ακριβώς ίδια κατανομή
- **Υπολογιστική μηδενική γνώση:** Τα πραγματικά και τα προσομοιωμένα αποτελέσματα, είναι μη-διαχωρίσιμα από οποιονδήποτε τρίτο ο οποίος έχει υπολογιστική δύναμη ίση με πιθανότικους πολυωνυμικού χρόνου αλγόριθμους.
- **Στατιστική μηδενική γνώση:** Τα πραγματικά και τα προσομοιωμένα αποτελέσματα, ακολουθούν κατανομές με πολύ μικρές (αμελητέες) στατιστικές διαφορές
- **Μηδενική γνώση ειλικρινή επαληθευτή HVZK:** Ο επαληθευτής είναι ειλικρινής, με την δυνατότητα να κάνει περαιτέρω υπολογισμούς πέρα από τους απαραίτητους για το πρωτόκολλο.

Στην συνέχεια παρουσιάζονται τα κυριότερα σχήματα αποδείξεων μηδενικής γνώσης, ενώ τέλος αναφέρουμε και τον μετασχηματισμό για να έχουμε μη διαδραστικό πρωτόκολλο.

2.8.1 Γνώση του Διακριτού Λογάριθμου

Όπως αναφέρθηκε σε μία από τις πρώτες ενότητες αυτού του κεφαλαίου, το πρόβλημα του διακριτού λογαρίθμου είναι ένα δύσκολο υπολογιστικά πρόβλημα, δηλαδή ένας πιθανοτικός πολυωνυμικού χρόνου αντίπαλος δεν μπορεί να το λύσει αποδοτικά. Βάση λοιπόν αυτής της λογικής, το πρόβλημα του διακριτού λογαρίθμου μπορεί να μας βοηθήσει να μετασχηματίσουμε ένα πρόβλημα μηδενικής γνώσης ώστε να κρύβουμε την τιμή που θέλουμε να δείξουμε ότι κατέχουμε χωρίς όμως να αποκαλύψουμε την τιμή αυτή. Ένα απλό παράδειγμα είναι όταν έχουμε μια δεδομένη τιμή y , έναν μεγάλο πρώτο p και έναν γεννήτορα g μιας κυκλικής ομάδας, τότε θέλουμε να αποδείξουμε ότι ξέρουμε ένα x τέτοιο ώστε $y = g^x \pmod p$. Με βάση λοιπόν του παραπάνω έχουν δημιουργηθεί τα πρωτόκολλα Schnorr.

Το πρωτόκολλο Schnorr[4] είναι ένα πρωτόκολλο απόδειξης γνώσης του διακριτού λογαρίθμου. Για την εφαρμογή του απαιτούνται κατά τα γνωστά μία κυκλική ομάδα G τάξης q , και ένας γεννήτορας της g , και σκοπός είναι η απόδειξη γνώσης ενός x τέτοιου ώστε $x = \log_g y$. Το πρωτόκολλο είναι διαδραστικό όπως έχουμε πει, και οι αλληλεπιδράσεις μεταξύ του αποδεικτή P και του επαληθευτή V είναι οι εξής τρεις:

- Αρχικά ο P δεσμεύεται σε μία τυχαιότητα, έστω r , και στην συνέχεια στέλνει μήνυμα $t = g^r$ στον V .

- Ο V στην συνέχεια διαλέγει τυχαία πρόκληση, έστω $c \in Z_q$, και την στέλνει στον P
- Ο P αφού λάβει το c στέλνει την απάντηση του, έστω s , με $s = r + cx$. Ο V (και όποιος έχει γνώση του c γενικώς) δέχεται αν $g^s = ty^c$.

Πρωτόκολλα σαν και το παραπάνω (δηλαδή με τρία μέρη: δέσμευση, πρόκληση, απάντηση) ονομάζονται Σίγμα-πρωτόκολλα και συμβολίζονται με Σ . Τα Σ -πρωτόκολλα εξυπηρετούν ανάγκες απόδειξης διακριτών λογαρίθμων αλλά και άλλων μορφών. Μια μικρή παραλλαγή του πρωτοκόλλου Schnorr είναι το πρωτόκολλο Chaum-Pedersen το οποίο χρησιμοποιείται για την απόδειξη ισότητας μεταξύ δύο διακριτών λογαρίθμων. Σε αναλογία με το προηγούμενο θα έχω πάλι τρία βήματα, απλά τώρα θα έχω δύο γεννήτορες της ομάδας τους g_1, g_2 και θα θέλω να δείξω ότι $y_1 = g_1^x, y_2 = g_2^x$:

- Αρχικά ο P δεσμεύεται σε μία τυχαιότητα, έστω r , και στην συνέχεια στέλνει μήνυμα $t_1 = g_1^r, t_2 = g_2^r$ στον V .
- Ο V στην συνέχεια διαλέγει τυχαία πρόκληση, έστω $c \in Z_q$, και την στέλνει στον P
- Ο P αφού λάβει το c στέλνει την απάντηση του, έστω s , με $s = r + cx$. Ο V (και όποιος έχει γνώση του c γενικώς) δέχεται αν $g_1^s = t_1 y_1^c$ και $g_2^s = t_2 y_2^c$.

Στην συνέχεια αναφερόμαστε σε κάποιες πιο εξειδικευμένες αποδείξεις μηδενικής γνώσης που θα χρησιμοποιηθούν στο σύστημα μας.

2.8.2 Διαζευκτικές Αποδείξεις (OR-proofs)

Στηρίζομενοι στο πρωτόκολλο Schnorr μπορούμε να παράξουμε διάφορες αποδείξεις αυτές οι αποδείξεις είναι οι αποδείξεις AND που είναι αποδείξεις ισότητας, όπως το Chaum-Pedersen, οι αποδείξεις EQ όπου με δεδομένα y_1, y_2 εμείς θέλουμε να δείξουμε ότι $y_1 = g^x, y_2 = g^x$ και φυσικά οι αποδείξεις OR με τις οποίες θα ασχοληθούμε και εκτενέστερα μιας και είναι απαραίτητη η κατανόηση τους για την μελέτη του συστήματος Helios.

Για τις αποδείξεις OR ισχύουν οι κλασικές προϋποθέσεις όπως και πριν, μόνο που τώρα δεδομένων y_1 ή y_2 θέλουμε να δείξουμε ότι διαθέτουμε γνώση είτε του $x_1 = \log_g y_1$ είτε του $x_2 = \log_g y_2$ ή πιθανά και των δύο. Για να επιτευχθεί κάτι τέτοιο πρέπει ο P να 'κλέψει' και να χρησιμοποιήσει τον προσομοιωτή S για να παράξει μία απόδειξη για την τιμή που δεν ξέρει και θα παράξει κανονικά την δεύτερη απόδειξη όπως στο κλασικό πρωτόκολλο. Ο τρόπος με τον οποίο θα γίνει αυτό, και οι μετασχηματισμοί του πρωτοκόλλου θα φανούν όταν γράψουμε τον ακριβή τρόπο με τον οποίο λειτουργεί το νέο πρωτόκολλο παρακάτω, εξηγώντας παράλληλα ό,τι χρήζει προσοχής. Αυτό που χρειάζεται να πούμε προς το παρόν είναι ότι η πρόκληση, τώρα θα σπάσει σε δύο προκλήσεις, και οι απααντήσεις θα είναι και αυτές δύο με την σειρά τους. Η μία πρόκληση και η αντίστοιχη απάντηση της θα είναι προσομοιωμένη, ενώ η άλλη θα παράγεται κανονικά. Τέλος θα πρέπει το άθροισμα των προκλήσεων να ισούται με την συνολική πρόκληση από την μεριά του V για να μην μπορεί να κλέψει ο P . Τα νέα βήματα λοιπόν είναι τα εξής[7]:

- Θα θεωρήσουμε ότι ο P έχει γνώση για το $x_1 = \log_g y_1$. Η απόδειξη είναι εντελώς ανάλογη αν γνωρίζει το $x_2 = \log_g y_2$. Αρχικά ο P δεσμεύεται σε μία τυχαιότητα, έστω r , και παράγει τα $t_1 = g_1^r$ κατά τα γνωστά, στέλνοντας το στον V . Αυτό που αλλάζει τώρα είναι ο τρόπος παραγωγής του t_2 . Για την παραγωγή του θα γίνει προσομοίωση, αφού θα θεωρήσουμε εμείς τυχαία την πρόκληση c_2 και την αντίστοιχη απάντηση της r_2 , με $t_2 = g^{r_2} y_2^{-c_2}$.
- Ο V στην συνέχεια διαλέγει τυχαία πρόκληση, έστω $c \in Z_q$, και την στέλνει στον P , χωρίς να αλλάζει τίποτα στο γνωστό πρωτόκολλο.
- Ο P αφού λάβει το c στέλνει την απάντηση του, θα σπάσει αυτήν την πρόκληση σε c_1, c_2 με την c_2 να είναι ήδη γνωστή και την $c_1 = c - c_2$. Οι απαντήσεις με την σειρά τους θα είναι η s_2 που έχει προσομοιωθεί και η s_1 , με $s_1 = r + c_1 x_1$. Ο V (και όποιος έχει γνώση του c γενικώς) δέχεται αν $g^{s_1} = t_1 y_1^{c_1}, g^{s_2} = t_2 y_2^{c_2}$ και $c_1 + c_2 = c$.

(Στο παραπάνω πρωτόκολλο λειτουργούμε σε πράξεις $\text{mod } p$)

Το παραπάνω πρωτόκολλο είναι πλήρες γιατί: $c_1 + c_2 = c - c_2 + c_2 = c$

$$g^{s_1} = g^{r_1 + c_1 x_1} = g^{r_1} (g^{x_1})^{c_1} = t_1 y_1^{c_1}$$

$$g^{s_2} = t_2 y_2^{c_2}$$

ορθό αφού αν θεωρήσω 2 αποδεκτές συνομιλίες με διαφορετικά c θα πρέπει να έχω γνώση σε κάθε περίπτωση είτε του x_1 είτε του x_2 , ενώ διαθέτει και μηδενική γνώση, καθώς οι κατανομές που ακολουθούν οι πραγματικές και οι προσομοιωμένες εφαρμογές του πρωτοκόλλου με πιθανότητα μάλιστα ίση με $\frac{1}{n^3}$.

Τέλος αναφέρουμε την ύπαρξη και NEQ-proofs οι οποίες διασφαλίζουν ότι τα δύο x_1, x_2 είναι διαφορετικά μεταξύ τους. Η λογική είναι ακριβώς η ίδια αλλά δεν αναφέρεται καθώς δεν χρειάζεται τελικά στα πλαίσια αυτής της διπλωματικής εργασίας.

2.8.3 Αποδείξεις Ανακατέματος (Shuffle-proofs)

Αρχικά, θα μιλήσουμε θεωρητικά για τις αποδείξεις ανακατέματος ή αλλιώς Shuffle-proofs όπως είναι γενικότερα γνωστές. Οι παραπάνω αποδείξεις έχουν ευρεία χρησιμότητα σε πολλά κρυπτοσυστήματα και η μελέτη τους είναι έντονη ειδικά τα τελευταία χρόνια. Σε αυτήν την υποενότητα, λοιπόν, θα κάνουμε μία πρώτη εισαγωγή στην έννοια των παραπάνω αποδείξεων, ενώ στο κεφάλαιο της δικιάς μας υλοποίησης θα γίνει πλήρης ανάλυση μίας τέτοιας απόδειξης.

Στα πλαίσια αυτής της διπλωματικής εργασίας έχουμε αναφερθεί νωρίτερα στα Mixnets. Αυτό που αναφέραμε τότε είναι ότι απαιτείται ένα ανακάτεμα των αρχικών εισόδων σε συνδυασμό με μια επανακρυπτογράφηση τους. Αυτά όπως αναφέραμε θα τα κάνει το δίκτυο, με απαραίτητη προϋπόθεση να μπορεί ο οποιοσδήποτε να ελέγξει την ορθότητα της διαδικασίας. Βεβαίως, η απόδειξη της ορθότητας θα ήταν εύκολη διαδικασία, αν απλά βλέπαμε την σχέση που συνδέει τις εισόδους και τις εξόδους και είχαμε μία 1-1 αντιστοιχία. Κάτι τέτοιο όπως είναι σαφές δεν επιτρέπεται, και υπό αυτήν την ανάγκη δημιουργήθηκαν οι πρώτες Shuffle-proofs.

Θα πρέπει να τονίσουμε ότι υπάρχει μεγάλη βιβλιογραφία όσον αφορά τις συγκεκριμένες αποδείξεις. Εμείς παρόλα αυτά θα επικεντρωθούμε στην απόδειξη που δίνεται από τους FURUKAWA Jun and Kazue Sako στο paper: An Efficient Scheme for Proving a Shuffle[20]. Ουσιαστικά η παραπάνω απόδειξη αποτελεί την πρώτη του είδους της πάνω στην οποία έχουν στηριχθεί πολλές άλλες. Είναι η πρώτη απόδειξη η οποία είναι αποδοτική και σαφώς αποδοτικότερη από όλες τις προηγούμενες τις, ενώ ακόμη και σήμερα δεν έχουν υλοποιηθεί αποδείξεις σημαντικά πιο αποδοτικές από αυτήν. Η επιλογή της στηρίζεται στην απλότητα της, και στην ευκολία κατανόησης της από όλους λόγω της καθόλιξης αναγνώρισης της στους κόλπους της κρυπτογραφίας. Τέλος, τονίζεται ότι η απόδειξη καθώς και η εισαγωγή της στο σύστημα μας, θα γίνει εμφανής σε μεταγενέστερη ενότητα όπου και θα γίνει πλήρης ανάλυση της.

2.8.4 Μη-Διαδραστικές Αποδείξεις

Όταν μιλάγαμε μέχρι τώρα για αποδείξεις μηδενικής γνώσης, αναφερόμασταν σε ένα διαδραστικό μοντέλο στο οποίο ο P και ο V επικοινωνούσαν και ο V πείθονταν για την αλήθεια του ισχυρισμού του P . Όπως είναι εμφανές κάτι τέτοιο δεν είναι πάντα εφαρμόσιμο, με το πιο απλό παράδειγμα να είναι οι ηλεκτρονικές ψηφοφορίες, με τον ψηφοφόρο να πρέπει να αποδείξει την ορθότητα της ψήφο του σε κάθε πιθανό ενδιαφερόμενο.

Έτσι λοιπόν, δημιουργήθηκαν οι μη-διαδραστικές αποδείξεις μηδενικής γνώσης, όπου δεν απαιτείται επικοινωνία μεταξύ του P και του V . Η ιστορία τους είναι μεγάλη, και εμείς θα ασχοληθούμε με τον κυριότερο εκφραστή αυτής, που δεν είναι άλλος από την ευριστική συνάρτηση των Fiat-Shamir[5] η οποία λειτουργεί στο μοντέλο του τυχαίου μαντείου[11].

Επί της ουσίας η συνεισφορά των παραπάνω, ήταν ότι κατά κάποιο τρόπο κατάφεραν να μετατρέψουν ένα διαδραστικό μοντέλο αποδείξεων σε μη-διαδραστικό, υπό την προϋπόθεση ότι υπάρχουν τυχαία μαντεία. Πρακτικά, μπορούμε να θεωρήσουμε τυχαίο, όμως, το αποτέλεσμα μίας συνάρτησης κατακερματισμού όταν η είσοδος της εξαρτάται από μία δήμοσια τιμή, στα πλαίσια πάντα της ασύμμετρης κρυπτογράφησης.

Για την καλύτερη κατανόηση του τρόπου με τον οποίο λειτουργεί το σύστημα θα δείξουμε τον μετασχηματισμό του πρωτοκόλλου Schnorr σε μη διαδραστικό μοντέλο με την βοήθεια μίας συνάρτησης κατακερματισμού H . Αρχικά, ξαναπαρουσιάζουμε το πρωτόκολλο όπως ακριβώς το δείξαμε σε προηγούμενη υποενότητα. Έχουμε μία κυκλική ομάδα G τάξης q , και ένας γεννήτορας της g , και σκοπός είναι η απόδειξη γνώσης ενός x τέτοιου ώστε $x = \log_g y$. Οι αλληλεπιδράσεις μεταξύ του αποδεικτή P και του επαληθευτή V είναι οι εξής τρεις:

- Αρχικά ο P δεσμεύεται σε μία τυχαιότητα, έστω r , και στην συνέχεια στέλνει μήνυμα $t = g^r$ στον V .
- Ο V στην συνέχεια διαλέγει τυχαία πρόκληση, έστω $c \in Z_q$, και την στέλνει στον P

- Ο P αφού λάβει το c στέλνει την απάντηση του, έστω s , με $s = r + cx$. Ο V (και όποιος έχει γνώση του c γενικώς) δέχεται αν $g^s = ty^c$.

Τώρα παρατηρούμε ότι αυτό που χρειάζεται ο V να κάνει μόνο είναι η παραγωγή του c . Άρα, την θέση του μπορεί να αναλάβει ο οποιοσδήποτε μπορεί να παράξει το τυχαίο c . Έτσι, επιλέγουμε το τυχαίο c να παράγεται μέσα από την συνάρτηση κατακερματισμού μας η οποία θα έχει ως είσοδο τα επιλεγμένα από τον P y, t αλλά και την δημόσια τιμή του g . Έτσι, θα έχουμε ένα φαινομενικά τυχαίο c διαφορετικό για κάθε P . Το νέο λοιπόν πρωτόκολλο θα είναι το εξής:

- Αρχικά ο P δεσμεύεται σε μία τυχειότητα, έστω r και υπολογίζει κατά τα γνωστά το $t = g^r$.
- Ο P παράγει την τυχαία πρόκληση c μέσα από το $c = H(g, y, t)$.
- Ο P τέλος παράγει το s , ως $s = r + cx$, και δημοσιοποιεί το στρινγκ αναφοράς της απόδειξης y, t, s . Όποιος θέλει μπορεί λοιπόν να λάβει και το g που είναι δημοσίως γνωστό, αλλά και την H η οποία και αυτή είναι γνωστή, και να παράξει μόνος του το $c = H(g, y, t)$, και να πραγματοποιήσει τον έλεγχο $g^s = ty^c$.

Άρα, το διαδραστικό μοντέλο, έχει γίνει μη-διαδραστικό και μπορεί ο οποιοσδήποτε να ελέγξει την ορθότητα του.

Κεφάλαιο 3

Το σύστημα Helios

3.1 Εισαγωγή στα End-to-end auditable voting systems

Τα End-to-end auditable voting systems(E2E) είναι εκείνα τα συστήματα ψηφοφοριών, τα οποία διασφαλίζουν υψηλή ακεραιότητα και ασφάλεια έναντι πιθανών αλλοιώσεων των αποτελεσμάτων. Τα E2E συστήματα στηρίζονται τις περισσότερες φορές σε κρυπτογραφικές διαδικασίες που επιτρέπουν την ύπαρξη αποδείξεων του τι ψήφισε ο ψηφοφόρος σε συνδυασμό με την δυνατότητα να ελέγχει ο εκάστοτε ψηφοφόρος αν η ψήφος του καταμετρήθηκε όπως αυτός ψήφισε, ενώ ταυτόχρονα δεν αποκαλύπτεται το τι ψήφισε. Γενικότερα τα περισσότερα E2E έχουν να αντιμετωπίσουν δύο κυρίαρχα προβλήματα, το πρόβλημα της ακεραιότητας της ψηφοφορίας και το πρόβλημα της ιδιωτικότητας της ψήφου. Το να επιτύχεις απόλυτη ακεραιότητα και ιδιωτικότητα την ίδια στιγμή είναι ένα ανοιχτό πρόβλημα. Υπάρχουν πάρα πολλές υλοποιήσεις E2E συστημάτων όπως τα Prêt à Voter, Scratch and Vote, ThreeBallot, Scantegrity, Helios και πολλά άλλα. Εμείς θα επικεντρωθούμε στο σύστημα Helios και στις επόμενες ενότητες αυτού του κεφαλαίου θα αναφερθούμε αναλυτικά στην λειτουργία του.

3.2 Εισαγωγή στο Helios

Το Helios[6] είναι ένα web-based σύστημα το οποίο είναι διαθέσιμο για όλους, οποιοσδήποτε μπορεί να μπει στην ιστοσελίδα του και να διοργανώσει μία ψηφοφορία. Φυσικά ο κώδικας του είναι διαθέσιμος προς όλους και μπορεί όποιος θέλει να το κατεβάσει και να το τρέξει στο σύστημα του κάνοντας όποια τροποποίηση επιθυμεί(όπως φυσικά συμβαίνει και στα πλαίσια αυτής της εργασίας). Το συγκεκριμένο σύστημα χρησιμοποιεί κρυπτογραφικές μεθόδους για να διασφαλίσει την ακεραιότητα της ψηφοφορίας και την ιδιωτικότητα των ψηφοφόρων. Στις επόμενες ενότητες αναλύεται η χρησιμότητα, ο τρόπος λειτουργίας αλλά και γενικότερα ό,τι έχει σχέση και είναι ιδιαίτερης σημασίας για το σύστημα.

3.2.1 Πότε και Γιατί Χρησιμοποιούμε το Helios

Το Helios αλλά και κάθε ηλεκτρονικό σύστημα στο οποίο ο ψηφοφόρος ψηφίζει μέσα από mail ή μέσα από κάποιον browser ή γενικότερα χωρίς να υπάρχει φυσική παρουσία είναι επιρρεπές προς προσπάθεια χειραγώγησης της ψηφοφορίας. Αυτό συμβαίνει όπως αντιλαμβάνεται κανείς καθώς είναι αδύνατο αν δεν υπάρχει οπτική επαφή με τον ψηφοφόρο να διασφαλιστούν οι συνθήκες κάτω από τις οποίες θα ψηφίσει ο εκάστοτε ψηφοφόρος.

Ακόμη, ένα ακόμη πρόβλημα που αντιμετωπίζει το υπάρχον σύστημα και προς το παρόν τουλάχιστον δεν συστήνεται για σημαντικές ψηφοφορίες (πχ βουλευτικές εκλογές) είναι η πιθανότητα να υπάρχει πρόβλημα στον εξυπηρετητή (browser) του ψηφοφόρου, καθώς τότε είναι πιθανές διάφορες επιθέσεις όπως αυτές έχουν γίνει γνωστές.

Επιπλέον, όπως αναφέρθηκε προηγουμένως δεν μπορείς να έχεις ταυτόχρονα και απόλυτη ιδιωτικότητα και απόλυτη ακεραιότητα μέσα από ένα σύστημα. Έτσι, το Helios επιλέγει να διασφαλίζει απόλυτη ακεραιότητα μέσα από τις κρυπτογραφικές διαδικασίες, έτσι ώστε η ακεραιότητα να μην στηρίζεται σε κανέναν άλλον. Από την άλλη, για την ιδιωτικότητα της ψήφου είναι απαραίτητο να εμπιστευόμαστε την εκάστοτε αρχή που θα αποκρυπτογραφήσει το αποτέλεσμα, δηλαδή, στην περίπτωση μας το Helios. Βεβαίως, και το ζήτημα της ιδιωτικότητας της ψήφου, μπορεί αν όχι να διασφαλιστεί, τότε τουλάχιστον να ενδυναμωθεί, προσθέτοντας όσο τον δυνατόν περισσότερες αρχές, διαμοιράζοντας μεταξύ τους το κλειδί αποκρυπτογράφησης της ψηφοφορίας. Έτσι, ακόμη και αν κάποιες αρχές είναι διεφθαρμένες, αρκεί μία (ή ένα σχετικό ποσοστό που εμείς θα επιλέξουμε αναλόγως) να μην είναι διεφθαρμένο, ώστε να διασφαλίζεται η ιδιωτικότητα των ψηφοφόρων.

Η ιστοσελίδα του Helios είναι αυτή <http://heliosvoting.org>. Μπορεί ο καθένας να το κατεβάσει στον υπολογιστή του και να δει τον κώδικα που εκτελείται. Στην συνέχεια, θα παρουσιάσουμε την ακριβή διαδικασία με την οποία μπορεί ο καθένας να εισέλθει στο σύστημα για την ψηφοφορία και να ψηφίσει ό,τι θέλει.

3.2.2 Η Διαδικασία της Ψηφοφορίας

Σε αυτήν την υποενότητα θα αναφερθούμε συνοπτικά στην διαδικασία με την οποία κάποιος θα εισέλθει στο σύστημα για να ψηφίσει. Ο λόγος που θα το κάνουμε αυτό είναι για να δείξουμε την απλότητα της διαδικασίας αλλά και κάποια μέτρα για την διασφάλιση της ορθής λειτουργίας του συστήματος. Αρχικά λοιπόν ο ψηφοφόρος εισέρχεται στο σύστημα, και διαλέγει την ψηφοφορία που θέλει να συμμετάσχει. Στην συνέχεια το σύστημα της ψηφοφορίας καθοδηγεί τον εκάστοτε ψηφοφόρο στις ερωτήσεις και καταγράφει τις απαντήσεις του. Με το που ολοκληρωθούν οι ερωτήσεις, και ο ψηφοφόρος έχει επιλέξει τις απαντήσεις του, το κείμενο κρυπτογραφείται και παράγεται και η απόδειξη του εφαρμόζοντας μια συνάρτηση κατακερματισμού στο κρυπτοκείμενο. Σε αυτό το σημείο, δίνεται η δυνατότητα στον ψηφοφόρο είτε να σφραγίσει την ψήφο του είτε να την ελέγξει. Αν επιλέξει τον έλεγχο αυτής τότε το σύστημα δείχνει το κρυπτοκείμενο καθώς και την τυχαιότητα που χρησιμοποιήθηκε για την παραγωγή του, ενώ προφανώς αυτό το κρυπτοκείμενο παύει να είναι πλέον έγκυρο και καταστρέφεται ώστε να δημιουργηθεί

γηθεί καινούριο με νέα άγνωστη τυχαιότητα. Όταν τελικά σφραγιστεί η τελική ψήφος τότε το σύστημα διώχνει κάθε πληροφορία σχετικά με το αρχικό κείμενο και την τυχαιότητα και μένει μόνο το τελικό κρυπτοκείμενο. Με την ολοκλήρωση λοιπόν της διαδικασίας, ζητείται από τον ψηφοφόρο να πιστοποιήσει το ποιος είναι εισάγοντας τους κατάλληλους κωδικούς και όνομα χρήστη.

Παραπάνω, παρατηρούμε ότι η διαδικασία αυθεντικοποίησης και πιστοποίησης του χρήστη είναι το τελευταίο στάδιο και δεν γίνεται στην αρχή. Αυτό αποσκοπεί στην υψηλή ελεγχιμότητα του συστήματος από οποιονδήποτε, αφού όποιος επιθυμεί (ακόμη και άτομα χωρίς την δυνατότητα να ψηφίσουν) μπορεί να εισέλθει στο σύστημα και να παράξει μία ψήφο με σκοπό τον έλεγχο της διαδικασίας. Ακόμη, όπως φαίνεται και από ότι προείπαμε δεν λαμβάνονται ιδιαίτερα μέτρα για την αποφυγή του εξαναγκασμού να ψηφίσει ο εκάστοτε ψηφοφόρος ό,τι του υποδείξει ένας κακόβουλος τρίτος. Επί της ουσίας, κάτι τέτοιο δεν έχει ληφθεί υπόψιν από το υπάρχον σύστημα εσκεμμένα, καθώς όπως έχει αναφερθεί προτείνεται το υπάρχον σύστημα να μην χρησιμοποιείται για σημαντικές ψηφοφορίες παρά μόνο για ψηφοφορίες όπου η πιθανότητα εκβιασμού και εξαναγκασμού να είναι μικρή.

Στις επόμενες ενότητες αναφέρεται η πλήρης κρυπτογραφική διάταξη που χρησιμοποιείται στο E2E σύστημα με σκοπό την κατανόηση του τρόπου με τον οποίο διασφαλίζεται η ακεραιότητα και η γενικότερα ορθή λειτουργία του συστήματος.

3.3 Κρυπτογραφικές Διαδικασίες

Οι κρυπτογραφικές διαδικασίες που χρησιμοποιούνται στο σύστημα Helios αποτελούν σημαντικό κομμάτι της ορθής λειτουργίας του, καθώς μέσω αυτών διασφαλίζεται η απόλυτη ακεραιότητα της ψηφοφορίας, καθώς και επίσης η κατά το δυνατόν μεγαλύτερη δυνατή ιδιωτικότητα της ψήφου. Οι επόμενες υποενότητες, θα παρουσιάσουν τον τρόπο με τον οποίο δημιουργούνται τα ψηφοδέλτια και παράγονται τα κρυπτοκείμενα, τον τρόπο με τον οποίο διασφαλίζεται ότι ολοι οι ψηφοφόροι ψήφισαν ορθώς και γενικά την συνολική ορθότητα του συστήματος.

3.3.1 Παραγωγή της Ψήφου

Αρχικά, θα αναφερθούμε στον τρόπο με τον οποίο παράγονται τα ψηφοδέλτια και στην συνέχεια τι συμβαίνει όταν κάποιος τα συμπληρώνει. Ο κρυπτογραφικός τρόπος με τον οποίο διασφαλίζεται η ακεραιότητα της ψηφοφορίας είναι μέσω της χρήσης ενός πρωτοκόλλου ασύμμετρης κρυπτογράφησης. Ο ακριβής αλγόριθμος που χρησιμοποιείται είναι ο εκθετικός El gamal. Ο εκθετικός El gamal είναι μία μικρή παραλλαγή του κλασικού El gamal με την μόνη διαφορά ότι αντί να κρυπτογραφείται το μήνυμα m κρυπτογραφείται το g^m έτσι ώστε να εκμεταλευτούμε την ομομορφικότητα του απλού El gamal ως προς τον πολλαπλασιασμό και να την μετασχηματίσουμε για να έχουμε το άθροισμα των αρχικών κειμένων (πιο αναλυτικά στην τελευταία υποενότητα αυτής της ενότητας).

Άρα, διαλέγουμε το σύστημα μας να υλοποιεί την κρυπτογράφηση El gamal με $p = 2q + 1$, με p, q να είναι πρώτοι, έναν γεννήτορα g της κυκλικής υποομάδας του Z_p^* . Κατά τα γνωστά όλα τα παραπάνω τα παράγει το σύστημα μας, στην προ-

κείμενη περίπτωση δηλαδή το Helios, και αποτελούν την περιγραφή του δημοσίου κλειδιού σε συνδυασμό με το $y = g^x$ όπου x είναι το ιδιωτικό κλειδί αποκρυπτογράφησης του τελικού κρυπτογραφημένου αποτελέσματος. Όπως αναφέραμε, αν προσθέσουμε πολλαπλές αρχές δεν υπάρχει μονάχα ένα κλειδί αποκρυπτογράφησης, αλλά πολλά, τα οποία αν συνδυαστούν θα δώσουν το κατάλληλο αποτέλεσμα.

Στη συνέχεια, μετά την παραγωγή των παραπάνω παράγονται τα ψηφοδέλτια. Τα ψηφοδέλτια δεν είναι τίποτα άλλο πέρα από μία σειρά ερωτήσεων του τύπου 0 ή 1. Το υπάρχον σύστημα δίνει την δυνατότητα να έχουμε μονάχα ερωτήσεις της παραπάνω μορφής, δηλαδή για κάθε πιθανή ερώτηση υπάρχουν n πιθανές απαντήσεις. Ο κάθε ψηφοφόρος στην συνέχεια του δίνεται η δυνατότητα να επιλέξει από k έως m από τις πιθανές απαντήσεις, με $0 \leq k \leq m \leq n$.

Η ακριβής διαδικασία, λοιπόν, έχει ως εξής, όταν εισέρχεται ο εκάστοτε ψηφοφόρος στο σύστημα και διαλέγει τις εκλογές που θέλει να συμμετάσχει, του δίνονται μία σειρά από ερωτήσεις. Για κάθε ερώτηση ισχύουν αυτά που λέμε στην παραπάνω παράγραφο. Στην συνέχεια ο ψηφοφόρος επιλέγει βάζοντας τικ σε όσες επιλογές θέλει ανάλογα με την ερώτηση και στην συνέχεια προχωράει είτε στην επόμενη ερώτηση είτε στο τέλος της ψηφοφορίας αν ήταν η τελευταία ερώτηση. Κατά την διαδικασία που προχωράει, οι επιλογές που επιλέχθηκαν κρυπτογραφούνται. Ο τρόπος που γίνεται η κρυπτογράφηση είναι είτε κρυπτογραφώντας την τιμή 0 (για την ακρίβεια το g^0) αν η επιλογή της εκάστοτε ερώτησης δεν είναι επιλεγμένη, ή την τιμή 1 (g^1) αν η επιλογή είναι τιχαρισμένη.

Προφανώς το σύστημα ανάλογα με τις συνθήκες που έχει θέσει ο δημιουργός της ψηφοφορίας επιτρέπει την επιλογή τόσων απαντήσεων όσων επιτρέπονται. Βεβαίως, ένας κακόβουλος θα μπορούσε να φτιάξει μόνος του τα κρυπτοκείμενα και να τα βάλει απευθείας στο σύστημα, κρυπτογραφώντας ότι θέλει. Για να αποφευχθεί αυτό, όμως, χρησιμοποιούνται αποδείξεις μηδενικής γνώσης, ώστε να αποδεικνύεται η ορθότητα της ψήφου, δηλαδή ότι οι επιλογές είναι τόσες όσες επιτρέπονται, και ότι η τιμή του κρυπτογραφημένου αρχικού κειμένου είναι είτε 0 είτε 1.

Τέλος, δίνεται η δυνατότητα στον ψηφοφόρο είτε να σφραγίσει την ψήφο του είτε να την ελέγξει όπως έχουμε ήδη πει. Όταν εν τέλει αποφασίσει να την σφραγίσει και κάνει το απαραίτητο login για να πιστοποιήσει την ταυτότητα του, στέλνει το κρυπτοκείμενο στο σύστημα, το οποίο ανακοινώνει τις καταχωρημένες ψήφους σε έναν πίνακα ανακοινώσεων, με σκοπό τον έλεγχο των ψήφων τόσο από τον ίδιο τον ψηφοφόρο όσο και από οποιονδήποτε άλλο ενδιαφέρεται.

3.3.2 Ορθότητα της Ψήφου

Μιλήσαμε προηγουμένως για την διαδικασία με την οποία παράγεται η ψήφος και αναφέραμε την ανάγκη η ψήφος να είναι ορθή. Η απόδειξη της ορθότητας της ψήφου από την μεριά του ψηφοφόρου γίνεται με την δημιουργία μίας απόδειξης μηδενικής γνώσης επί της ψήφου. Για την ακρίβεια ο τύπος αυτός της απόδειξης θα είναι παρόμοιος με εκείνον του διαζευκτικού τύπου αποδείξεων. Πιο συγκεκριμένα στο υπάρχον σύστημα χρησιμοποιείται μία εφαρμογή του paper[24] των Ronald Cramer, Rosario Gennar, Berry Schoenmakers με τίτλο A Secure and Optimally Efficient Multi-Authority Election Scheme

Η ακριβής διαδικασία λοιπόν διαδικασία με την οποία αποδεικνύει ο οποιοδήποτε την ορθότητα της ψήφου είναι μέσω της ύπαρξης δύο ειδών διαζευκτικών αποδείξεων, μίας ατομικής για κάθε επιλογή μίας ερώτησης και μίας συνολικής για μία ολόκληρη ερώτηση. Η ατομική απόδειξη αποτελεί επί της ουσίας μία απόδειξη ότι η τιμή μίας επιλογής θα είναι είτε 0 είτε 1. Άρα, θα έχουμε μία απόδειξη μηδενικής γνώσης για αυτές τις δύο τιμές για όλες τις επιλογές. Ακόμη, η συνολική απόδειξη θα είναι μία συνολική απόδειξη για τον ομομορφικό συνδυασμό των ψήφων. Όπως προαναφέραμε οι πιθανές απαντήσεις που θα είναι τικαρισμένες είναι από k έως m . Άρα, αν συνδυάσουμε ομομορφικά όλες τις επιλογές η ελάχιστη τιμή που θα παρουσιάζεται στον εκθέτη θα είναι το ελάχιστο k και η μέγιστη το πολύ m .

Η μορφή των ατομικών αποδείξεων και των συνολικών θα είναι παρόμοια με την διαφορά ότι οι συνολικές αποδείξεις θα είναι πιο γενικευμένη μορφή των ατομικών. Στο παρακάτω σχήμα φαίνεται ο τρόπος με τον οποίο παράγεται μία ατομική απόδειξη. Το σύστημα παρακάτω είναι διαδραστικό, ενώ προφανώς εμείς το θέλουμε μη διαδραστικό. Αυτό είναι εύκολα μετατρέψιμο μέσω του Fiat-Shamir αν το c προκύψει από μία συνάρτηση κατακερματισμού, απλά για χάριν ευκολίας και κατανόησης το παρουσιάζουμε σαν διαδραστικό.

Επί της ουσίας το παραπάνω μας δείχνει τον τρόπο με τον οποίο δημιουργείται η ατομική απόδειξη. Πιο συγκεκριμένα αν θεωρήσουμε ότι το κρυπτοκείμενο El gamal μίας επιλογής είναι το c_i με $c_i = (\alpha = g^{x_i}, \beta = y^{x_i} g^m) \bmod p$. Ο ψηφοφόρος στην συνέχεια θα παράγει τα

$$A_1, B_1, A_2, B_2$$

με είτε τα A_1, B_1 να είναι προσομοιωμένα και τα A_2, B_2 να είναι κανονικά υπολογισμένα, είτε το αντίστροφο. Στην πρώτη περίπτωση θα τα υπολογίζει ως εξής,

$$A_1 = (g^{\text{response}_1} \alpha^{-c_1}), \quad B_1 = \frac{(y^{\text{response}_1})}{(\frac{\beta}{g^m})^{c_1}},$$

$$\text{και } A_2 = g^w, \quad B_2 = y^w.$$

Στην συνέχεια παράγει την συνολική πρόκληση μέσω μίας συνάρτησης κατακερματισμού με τα δεδομένα κατακερματισμού να είναι ("A": "3bZcd35GAS", "B": "7bXcd352sd", "choice-num": 0, "ciphertext": "alpha": "6BtdxuEwbcS+dfs3", "beta": "nC345Xbadw3235SD", "election-hash": "Nz1fWLvVLH3eY3Ox7u5hxfLZPdW", "question-num": 2) (με τις τιμές προφανώς να είναι τυχαία επιλεγμένες και όπως εμφανίζονται στο επίσημο σύστημα, αλλά σε κάθε περίπτωση να μην είναι αναγκαίο να ανταποκρίνονται πάντα σε αυτό που θα περάσει από την συνάρτηση κατακερματισμού), εφαρμόζοντας επί της ουσία το μοντέλο του Fiat-Shamir για τον μετασχηματισμό μίας απόδειξης σε μη-διαδραστική. Μετά για το τέλος της απόδειξης πρέπει να παράξει την μερική πρόκληση η οποία παράγεται από την σχέση $c_2 = c - c_1 \pmod{q}$, και υπολογίζει ευθέως την απάντηση ως εξής $\text{response}_2 = w + x_i * c_2$.

Συνολικά, λοιπόν παράγει n (όσες δηλαδή και ο αριθμός των επιλογών) τέτοιες ατομικές αποδείξεις. Επιπρόσθετα, σε αυτές πρέπει να παράξει και μία συνολική απόδειξη για να αποδείξει ότι ο αριθμός των επιλογών του είναι μέσα στα επιτρεπτά όρια. Ο τρόπος με τον οποίο συμβαίνει αυτό είναι μέσω της παραγωγής, του ομομορφικού συνδυασμού όλων των επιλογών, δηλαδή μέσω του πολλαπλασιασμού

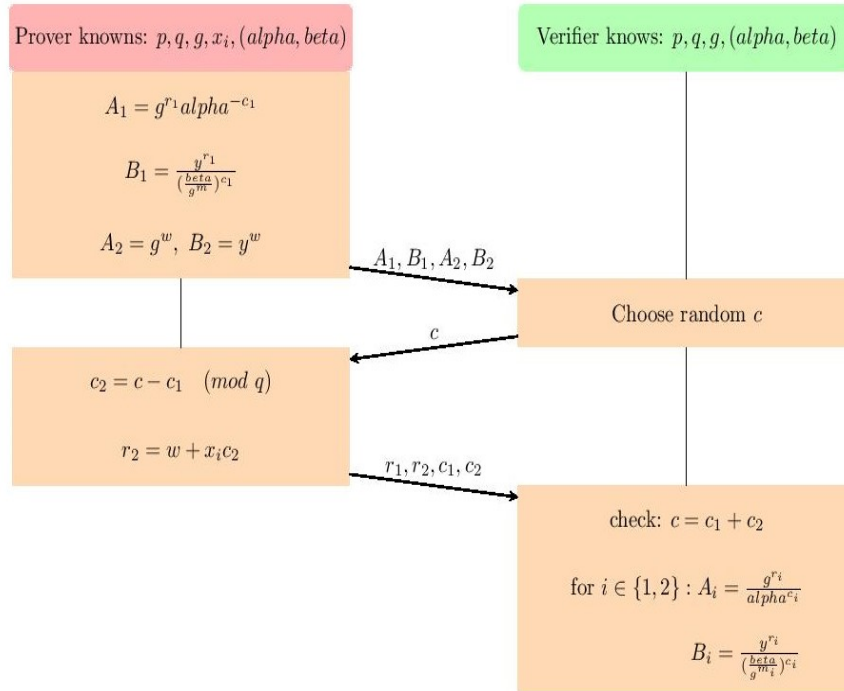


Figure 3.1: Or-proofs[0,1]

των κρυπτοκειμένων. Έτσι προκύπτει το εξής:

$$R = \prod_{i=1}^n ((\alpha_i), (\beta_i)) = (g^{\sum_{i=1}^n x_i}, y^{\sum_{i=1}^n x_i} g^{\sum_{i=1}^n m_i})$$

ενώ αν θεωρήσουμε ότι $\sum_{i=1}^n x_i = X$, $\sum_{i=1}^n m_i = M$, τότε θα είναι σαν να έχω ένα κλασικό κρυπτοκειμένο με τυχαιότητα X που κρυπτογραφεί κείμενο M , ενώ αυτό το κείμενο θα έχει μία τιμή ανάμεσα στην $\min = k$ και στην $\max = m$ που έχει θέσει ο δημιουργός της ψηφοφορίας, όπως έχουμε πει. Αυτήν λοιπόν την σχέση, $\min \leq M \leq \max$ θα πρέπει να την αποδείξει ο ψηφοφόρος.

Ο τρόπος απόδειξης της τελευταίας σχέσης, είναι μέσω της συνολικής απόδειξης. Αυτή η συνολική απόδειξη αποτελεί μία γενίκευση των ατομικών αποδείξεων, μόνο που τώρα θα προσομοιώνονται οι $\max - \min - 1$ αποδείξεις κατά τα γνωστά, και θα έχουμε κανονική παραγωγή της μίας και μοναδικής τιμής που έχει επιλεχθεί

στο φάσμα $[min, max]$. Άρα, αν θεωρήσουμε ότι έχουμε επιλέξει max επιλογές θα ισχύει:

$$A_{min} = (g^{response_{min}} alpha^{-c_{min}}), B_{min} = \frac{(y^{response_{min}})}{(\frac{beta}{g^m})^{c_{min}}},$$

$$A_{min+1} = (g^{response_{min+1}} alpha^{-c_{min+1}}), B_{min+1} = \frac{(y^{response_{min+1}})}{(\frac{beta}{g^m})^{c_{min+1}}},$$

$$\dots$$

$$\dots$$

$$\dots$$

$$A_{max-1} = (g^{response_{max-1}} alpha^{-c_{max-1}}), B_{max-1} = \frac{(y^{response_{max-1}})}{(\frac{beta}{g^m})^{c_{max-1}}},$$

και $A_{max} = g^w, B_{max} = y^w$.

Παρακάτω δίνεται μια απλή μορφή για την εκτενέστερη κατανόηση της μορφής που θα έχει γενικά η ψήφος μας. Υπενθυμίζουμε, ότι η ψήφος μας δεν είναι μόνο τα κρυπτογραφήματα $alpha, beta$ αλλά συνολικά όλα μαζί σε συνδυασμό με τις αποδείξεις και τα στοιχεία της ψηφοφορίας. Πιο συγκεκριμένα θα ισχύουν τα εξής:

<VOTE> =
{"answers": [<ENCRYPTED_ANSWER>, <ENCRYPTED_ANSWER>, ...], "election_hash": "Nz1fWLvVLH3eY3Ox7u5hxfLZPdw", "election_uuid": "1882f79c-65e5-11de-8c90-001b63948875"}
<ENCRYPTED_ANSWER> =
{"choices": [<ELGAMAL_CIPHERTEXT>, <ELGAMAL_CIPHERTEXT>, ...], "individual_proofs": [<ZK_PROOF_0..1>, <ZK_PROOF_0..1>, ...], "overall_proof": <ZK_PROOF_0..max>}

Οι τιμές παραπάνω είναι προφανώς τυχαίες.

3.3.3 Διαδικασία Αποκρυπτογράφησης και Απόδειξη Ορθής Λειτουργίας

Από εκεί και πέρα, μετά την παραγωγή των ψήφων, αυτές τοποθετούνται στον πίνακα ανακοινώσεων με την μορφή που φαίνεται παραπάνω. Στην συνέχεια, ο καθένας μπορεί να ελέγξει μία ψήφο ατομικά ή την συνολική διαδικασία. Πώς όμως γίνεται ο έλεγχος του συνόλου της ψηφοφορίας;

Μετά το πέρας της διαδικασίας που οι ψηφοφόροι επιλέγουν τις επιλογές που επιθυμούν, το σύστημα κλείνει και δεν δέχεται επιπλέον ψήφους. Στην συνέχεια ο διοργανωτής της ψηφοφορίας, επιλέγει να υπολογίσει την κρυπτογραφημένη καταμέτρηση. Με αυτήν την επιλογή, τα κρυπτοκείμενα συνδυάζονται ομομορφικά ώστε να έχουμε το τελικό συνολικό γινόμενο. Αν πχ θεωρήσουμε ότι έχουμε n

ψηφοφόρους το τελικό αποτέλεσμα θα είναι:

$$tally = \prod_{i=1}^n ((alpha_i), (beta_i)) = (g^{\sum_{i=1}^n x_i}, y^{\sum_{i=1}^n x_i} g^{\sum_{i=1}^n m_i})$$

Έπειτα, οι αρχές που διατηρούν τα αρχικά κλειδιά του El gamal αποκρυπτογραφούν το αποτέλεσμα. Προφανώς, με την αποκρυπτογράφηση της τελικής καταμέτρησης πρέπει οι αρχές να αποδείξουν ότι πράγματι κάνανε σωστά την αποκρυπτογράφηση. Αυτό είναι εφικτό με την ύπαρξη μίας απόδειξης μηδενικής γνώσης που παράγεται από την εκάστοτε αρχή. Αν έχουμε περισσότερες από μία αρχές, τότε πρέπει να παραχθούν μερικές αποκρυπτογραφήσεις, και για κάθε μία η εκάστοτε αρχή παρουσιάζει ότι έχει γνώση του κρυφού κλειδιού, και ότι η αποκρυπτογράφηση έγινε ορθώς.

Συνολικά, λοιπόν ο οποιοδήποτε μπορεί να ελέγξει αφενός την ορθότητα της ψήφου όπως αναφέρθηκε στην προηγούμενη ενότητα εκτενέστερα, ενώ μπορεί να ελέγξει και την ορθότητα της καταμέτρησης των ψήφων. Πιο συγκεκριμένα η συνολική διαδικασία απαρτίζεται από τα εξής βήματα, που μπορεί να ακολουθήσει οποιοσδήποτε ενδιαφερόμενος επιθυμεί να ελέγξει την ορθότητα της διαδικασίας:

1. Ο ενδιαφερόμενος ελέγχει το αποτύπωμα της ηλεκτρονικής ψηφοφορίας για την ορθότητα του
2. Ο ενδιαφερόμενος διασφαλίζει ότι η λίστα των ψηφοφόρων που παρουσιάζεται αντιστοιχεί με την λίστα αυτών που ψήφισαν(βασικά τα δεδομένα της λίστας των ψηφισάντων είναι περνάνε από μία συνάρτηση κατακερματισμού για πιο εύκολο έλεγχο)
3. Ελέγχει το αποτύπωμα κάθε ξεχωριστού ψηφοδέλιου
4. Για κάθε ψηφοδέλτιο ελέγχει την απόδειξη μηδενικής γνώσης του
5. Υπολογίζει τον ομομορφικό συνδυασμό όλων των ψηφοδελτίων
6. Ελέγχει αν κάθε αρχή που αποκρυπτογράφησε την ψήφο, έκανε ορθώς την αποκρυπτογράφηση. Αυτό γίνεται καθώς αν θεωρήσουμε ότι ένα κρυπτοκείμενο είναι της μορφής $(alpha, bet)$ κατά τα γνωστά ενώ το ιδιωτικό κλειδί της αρχής είναι x με το αντίστοιχο δημόσιο y τότε η μερική αποκρυπτογράφηση της αρχής θα είναι η $dec_{factor} = alpha^x \text{ mod } p$ και το μόνο που μένει να αποδείξει ο κατέχων του ιδιωτικού κλειδιού είναι ότι το $(g, y, alpha, dec_{factor})$ είναι μία τριάδα DDH το οποίο αποδεικνύεται εύκολο όπως έχουμε δει μέσα της αποδείξεις μηδενικής γνώσης του διακριτού λογάριθμου από την απόδειξη Chaum Pedersen που παρουσιάστηκε προηγούμενα.
7. Συνδυάζει όλες τις μερικές αποκρυπτογραφήσεις(αν υπάρχουν περισσότερες από μία αρχές), και ελέγχει ότι αυτές οι αποκρυπτογραφήσεις και το τελικό αποτέλεσμα συνάδουν. Επί της ουσίας αν θεωρήσουμε ότι το τελικό κείμενο(σε εκθετική μορφή) είναι το A τότε θα πρέπει

$$dec_{factor_1} * dec_{factor_2} * \dots * dec_{factor_k} * A = beta \pmod{p}$$

με k τον αριθμό των πιθανών αρχών που χρειάζονται να συνδυαστούν για να προκύψει το τελικό αποτέλεσμα.

3.4 Η Συνολική Διαδικασία

Σε αυτήν την ενότητα θα παρουσιάσουμε την όλη διαδικασία από την αρχή, δηλαδή την δημιουργία της ψηφοφορίας, μέχρι και το τέλος αυτής. Δεν θα αναφερθούμε αναλυτικά σε κρυπτογραφικές διαδικασίες που αναλύθηκαν σε προηγούμενες ενότητες, παρά μόνο σε κάποια σημεία που ίσως χρειάζεται. Η χρησιμότητα της παρούσας ενότητας είναι για την καλύτερη δυνατή κατανόηση της λειτουργίας του υπάρχοντος συστήματος. Πιο συγκεκριμένα λοιπόν θα έχουμε:

- Αρχικά, εισέρχεται στο σύστημα το άτομο που θέλει να δημιουργήσει την ψηφοφορία, και την δημιουργεί. Επιλέγει τον αριθμό των ερωτήσεων, τον τύπο τους καθώς και τις πιθανές απαντήσεις σε συνδυασμό με την ελάχιστη και την μέγιστη τιμή που μπορεί να επιλέξει από τις επιλογές ο κάθε υποψήφιος.
- Στην συνέχεια ο δημιουργός, αφού φτιάξει τις ερωτήσεις, επιλέγει τα άτομα που θα συμμετάσχουν σε αυτήν την ψηφοφορία. Αυτή η επιλογή γίνεται αφού έχει γράψει σε ένα αρχείο κειμένου τα ονόματα των ψηφοφόρων, μαζί με τα mail τους και τα προσωνύμια τους. Αφού είναι όλα έτοιμα, ο δημιουργός ξεκινάει την ψηφοφορία και στέλνει mail στους ψηφοφόρους της λίστας, το οποίο περιέχει έναν κωδικό που παράγεται τυχαία, σε συνδυασμό με το προσωνύμιο του.
- Μετά την έναρξη της διαδικασίας και αφού έχουν λάβει το mail οι ψηφοφόροι μπορούν να ψηφίσουν πλέον στην εκλογική διαδικασία. Βεβαίως, η είσοδος επιτρέπεται σε όλους, όπως επίσης και η ετοιμασία μίας έγκυρης ψήφου, αλλά δεν θα γίνεται υποβολή αν δεν ολοκληρωθεί το στάδιο της αυθεντικοποίησης όπως θα φανεί και στα παρακάτω.
- Ο κάθε ενδιαφερόμενος, αφού εισέλθει στην ιστοσελίδα και επιλέξει την εκάστοτε ψηφοφορία που θέλει να συμμετάσχει, μπαίνει πλέον σε διαδικασία όπου δεν χρειάζεται σύνδεση στο ίντερνετ καθώς δεν υπάρχει ανάγκη επικοινωνίας, παρά μόνο με το πέρας της διαδικασίας. Έτσι, ο ψηφοφόρος κάνει περιήγηση στις ερωτήσεις και επιλέγει τις απαντήσεις που επιθυμεί.
- Αφού, ολοκληρώσει τις επιλογές του, λαμβάνει χώρα η κρυπτογράφηση της ψήφου του, όπου και του δίνονται δύο επιλογές. Είτε να σφραγίσει την ψήφο του και να την στείλει ως έχει, είτε να την ελέγξει, οπότε και παρουσιάζονται οι συνθήκες κρυπτογράφησης, οπότε και έχουμε επανακρυπτογράφηση της ψήφου του.
- Μετά την τελική κρυπτογράφιση και αφού έχει αποφασίσει να σφραγίσει την ψήφο του, ο ψηφοφόρος πρέπει να κάνει το αναγκαίο log in με στοιχεία που του εστάλησαν στο mail.

- Τέλος, να αναφέρουμε ότι ο ψηφοφόρος μπορεί να κρατήσει αυτό που αποτελεί την απόδειξη της ψήφου του που δεν είναι άλλο από την κρυπτογραφημένη ψήφο του αφού έχει περάσει από μία συνάρτηση κατακεραματισμού. Ακόμη, η ψήφος του αφού επικυρωθεί, ανακοινώνεται στον πίνακα ανακοινώσεων της ψηφοφορίας.
- Μετά το πέρας της εκλογικής διαδικασίας(είτε χρονική λήξη είτε κατέπιλογίν του δημιουργού) η ψηφοφορία παγώνει και δεν γίνεται καμμία άλλη αλλαγή, δηλαδή καμμία άλλη ψήφος δεν γίνεται αποδεκτή. Στην συνέχεια δίνεται η εντολή να γίνει ο ομομορφικός πολλαπλασιασμός των κρυπτοκειμένων, ώστε να παραχθεί το τελικό αποτέλεσμα προς αποκρυπτογράφηση.
- Όταν γίνει διαθέσιμο το αποτέλεσμα του ομομορφικού συνδυασμού, δίνεται η εντολή προς τους κατέχοντες των ιδιωτικών κλειδιών να αποκρυπτογραφήσουν το αποτέλεσμα. Όταν ολοκληρωθεί η διαδικασία, ανακοινώνεται το τελικό αποκρυπτογραφημένο αποτέλεσμα. Προσοχή, επειδή έχουμε εκθετικό El gamal είναι απαραίτητο το αποκρυπτογραφημένο αποτέλεσμα όταν αποκτηθεί να λυθεί ένα μικρός διακριτός λογάριθμος ώστε να αποκτήσουμε το σωστό αποτέλεσμα που βρίσκεται στον εκθέτη.
- Όλη η διαδικασία, δηλαδή ο πίνακας ανακοινώσεων των ψήφων, και όλες οι αποδείξεις είναι διαθέσιμες προς όσους επιθυμούν να ελέγξουν ότι η ψηφοφορία είναι έγκυρη.

3.5 Θετικά και Αρνητικά του Helios

Σε αυτήν την ενότητα, θα μιλήσουμε για τα θετικά και τα αρνητικά που υπάρχουν στο σύστημα Helios. Αρχικά, να πούμε ότι το σύστημα είναι ανοιχτό προς όλους, και μπορεί ο οποιοσδήποτε να αποκτήσει πρόσβαση σε αυτό, να κατεβάσει τον πηγαίο κώδικά του και ακόμη να το ανεβάσει σε δικό του σέρβερ εφαρμόζοντας ό,τι αλλαγή επιθυμεί.

Ακόμη, όπως είπαμε προηγουμένως το σύστημα Helios είναι ένα E2E σύστημα που διασφαλίζει την ακεραιότητα και την ορθή λειτουργία του μέσω κρυπτογραφικών μοντέλων, που προς το παρόν πάντα, θεωρούνται ασφαλή και αδιάβλητα. Αυτή του η ακεραιότητα, και η ορθή λειτουργία στα πλαίσια του E2E είναι επαληθεύσιμα και ελέγξιμα από όλους.

Επιπρόσθετα, το υπάρχον σύστημα σε αντίθεση με άλλα E2E συστήματα(όπως πχ τα μίξνερς) είναι αρκετά πιο γρήγορο και αποδοτικό. Πιο συγκεκριμένα, μπορεί να πραγματοποιήσει πολύ μεγάλες ψηφοφορίες πολύ γρήγορα και με απαραίτητη κρυπτογραφική ασφάλεια. Αυτό οφείλεται στην ομομορφική φύση του συστήματος, το οποίο επιτρέπει την αποκρυπτογράφηση να γίνεται στο τέλος σε ένα και μόνο κρυπτοκειμένο, παραλείποντας τυχόν πράξεις στο ενδιάμεσο, και την ανάγκη για αποκρυπτογράφηση όλων των ψήφων.

Από την άλλη μεριά, πέρα από τα θετικά του το σύστημα έχει και ορισμένα αρνητικά. Σε αυτά πρέπει να τονίσουμε την μειωμένη ιδιωτικότητα της ψήφου, αν

δεν υπάρχει μεγάλος αριθμός(πόσο δε μάλλον αν είναι μόνο ένας) αρχών που κατέχουν τα ιδιωτικά κλειδιά της αποκρυπτογράφησης. Επίσης, εκ κατασκευής, το σύστημα είναι επιρρεπές σε απόπειρες εκβιασμού των ψηφοφόρων λόγω της ηλεκτρονικής φύσης του. Ακόμη, λόγω του ότι η ψηφοφορία γίνεται σε έναν browser αν υπάρχει πρόβλημα σε αυτόν τότε θα υπάρχει θέμα ασφάλειας και ακεραιότητας με την έννοια ότι θα είναι δυνατή κάποιου είδους επίθεση.

Βεβαίως, αυτό το πρόβλημα παραμονεύει σε κάθε έκφανση της ηλεκτρονικής ζωής μας, δηλαδή η πιθανότητα το σύστημα να μην είναι αξιόπιστο και ως εκ τούτου οι πράξεις μας ακόμη και αν θεωρητικά είναι κρυπτογραφικά ασφαλείς να είναι χειραγώγησιμες αν το σύστημα έχει πρόβλημα, κάτι που γίνεται ακόμα πιο σοβαρό αν αναλογιστεί κανείς την ευπάθεια που έχουν οι browsers σε κάποιον ιό ή ακόμη και στοχευμένη επίθεση.

Τέλος, ένα κυρίαρχο πρόβλημα που αντιμετωπίζει το υπάρχον σύστημα είναι ότι υποστηρίζει περιορισμένες τύπου ψηφοφορίες, όπου απλά μας δίνεται η δυνατότητα να επιλέξουμε κάποιες επιλογές, χωρίς να μπορείς να τις κατατάξεις ή και να τις βαθμολογήσεις όπως επιθυμείς. Κάτι τέτοιο είναι πρόβλημα καθώς υπάρχουν πολλαπλά εκλογικά συστήματα τα οποία στηρίζονται σε αυτήν ακριβώς την δυνατότητα κατάταξης των υποψηφίων. Αυτό ακριβώς το πρόβλημα αναλύεται και επιλύεται στο επόμενο κεφάλαιο, όπου και παρουσιάζουμε το νέο προτεινόμενο τροποποιημένο σύστημα.

Κεφάλαιο 4

Το Τροποποιημένο Σύστημα

4.1 Σκοπός του Τροποποιημένου Συστήματος

Σκοπός του νέου τροποποιημένου συστήματος είναι η τροποποίηση του ηλεκτρονικού συστήματος Helios έτσι ώστε να υποστηρίζει και ψηφοφορίες κατάταξης. Με τον όρο ψηφοφορίες κατάταξης, εννοούμε εκείνες τις ψηφοφορίες στις οποίες οι δεδομένες επιλογές μίας εκλογικής ερώτησης, να τοποθετούνται σε σειρά προτεραιότητας/προτίμησης από τον ψηφοφόρο. Κάτι τέτοιο δεν είναι εφικτό από το υπάρχον σύστημα και εμείς σε αυτό το κεφάλαιο θα αποδείξουμε και θα θεμελιώσουμε θεωρητικά πως κάτι τέτοιο είναι εφικτό στο παρόν σύστημα, αλλά όχι μόνο σε αυτό, καθώς η εφαρμογή του μπορεί να ενσωματωθεί σε πολλά παρόμοια ομομορφικά συστήματα.

Αυτό που χρειάζεται να πούμε σε πρώτη φάση για την τροποποίηση του συστήματος είναι ότι το μεγαλύτερο πρόβλημα είναι η αποδοτική εφαρμογή των αποδείξεων μηδενικής γνώσης. Είδαμε ότι στο απλό σύστημα, η απόδειξη μηδενικής γνώσης είναι γραμμική ως προς τους ψηφοφόρους λόγω της διαζευκτικής μορφής 0 ή 1 που είχε. Μία τέτοια απόδειξη προφανώς δεν είναι υλοποιήσιμη σε περίπτωση ψηφοφοριών κατάταξης καθώς η κατάταξη n υποψηφίων προϋποθέτει ότι θα πρέπει οι τιμές να μετατραπούν από 0 ή 1 σε 0 έως n .

Κατά τα άλλα, οι τροποποιήσεις του υπάρχοντος συστήματος δεν είναι πολλές αφού η γενικότερη μορφή και ιδέα παραμένει η ίδια. Αυτό που αλλάζει συνεπώς σε μεγάλο βαθμό και χρήζει αντιμετώπισης είναι η μορφή που θα έχουν οι τωρινές αποδείξεις μηδενικής γνώσης. Στις επόμενες ενότητες παρουσιάζουμε τις ιδέες για την τροποποίηση του συστήματος ώστε να μπορέσουμε να εφαρμόσουμε ψηφοφορίες κατάταξης.

Βεβαίως όπως προαναφέρθηκε υπάρχει ήδη ένα τροποποιημένο σύστημα, το σύστημα Zeus[1], το οποίο όμως δεν είναι ομομορφικό και στηρίζεται στην λογική

των Mixnets. Αναλυτικές οδηγίες για την χρήση του Zeus μπορεί κάποιος να βρει στις ακόλουθες πηγές, [2], [3].

4.2 Πρώτες Ιδέες Υλοποίησης

Στα αρχικά στάδια της εκπόνησης αυτής της διπλωματικής, υπήρξαν αρκετές ιδέες σχετικά με τον τρόπο με τον οποίο θα μπορούσαμε να εφαρμόσουμε ψηφοφορίες κατάταξης σε ομομορφικά συστήματα και πιο συγκεκριμένα για το Helios.

Αρχικά λοιπόν, η πρώτη σκέψη ήταν η μετατροπή των ψηφοφοριών κατάταξης σε ερωτήσεων τύπου 0-1. Δηλαδή, επί της ουσίας η αρχική ιδέα ήταν η διενέργεια n^2 συγκρίσεων μεταξύ των n επιλογών μέσω της ύπαρξης n^2 ερωτήσεων. Βασικά, λοιπόν δεν θα άλλαζε τίποτα (σε μεγάλο βαθμό τουλάχιστον), απλά θα μετατρέπαμε μία ερώτηση για κατάταξη σε πολλές ερωτήσεις 0-1. Αυτή η ιδέα όμως απορρίφθηκε σύντομα εξαιτίας της αναποδοτικότητας της σε συνδυασμό με διάφορα άλλα προβλήματα και αντιφάσεις που παρουσίαζε.

Η επόμενη ιδέα υλοποίησης, η οποία προχώρησε σε μεγάλο βαθμό μέχρι και την θεωρητική θεμελίωση της, ήταν η διαφοροποίηση των ατομικών αποδείξεων του απλού Helios. σε διαζευκτικές αποδείξεις, όχι πια 0-1 αλλά σε 0 έως $n - 1$. Ακόμη, τώρα πια δεν θα επιλέγονταν απλά όπως στο παλιό σύστημα οι πιθανές επιλογές, αλλά αντιθέτως θα βαθμολογούνταν ανάλογα με την σειρά προτίμησης. Έτσι, το κρυπτοκείμενο θα κρυπτογραφούσε πιθανές τιμές από 0 έως $n - 1$, και θα έπρεπε να αλλάζουμε τις ατομικές αποδείξεις μηδενικής γνώσης.

Βεβαίως, κάτι τέτοιο δεν θα ήταν τόσο περίπλοκο από πλευράς σκέψης, καθώς θα μπορούσαμε απλά να μετασχηματίσουμε τις υπάρχουσες αποδείξεις 0-1. Πράγματι, τέτοιου είδους αποδείξεις ήταν υλοποιημένες και στο απλό σύστημα αλλά έβρισκαν εφαρμογή στις συνολικές αποδείξεις μέχρι εκείνη την στιγμή. Η λογική τους θα ήταν ιδιαίτερα απλή: για κάθε πιθανή επιλογή θα κρυπτογραφούνταν μία τιμή, έστω m . Εμείς με την σειρά μας θα έπρεπε να αποδείξουμε ότι $m \in [0, n - 1]$, με μία απόδειξη μηδενικής γνώσης η οποία θα γινόταν προσομοίωση για τις $n - 1$ τιμές που δεν θα ήταν ίσες με το m και για το m η απόδειξη θα ήταν κλασική. Παρακάτω δίνουμε ένα απλό σχηματικό όπου οι πρώτες $n - 1$ αποδείξεις είναι οι προσομοιωμένες και η n -στη είναι η κανονική.

Βέβαια εύκολα καταλαβαίνει κανείς ότι και το παραπάνω σχήμα έχει ένα μεγάλο ελάττωμα. Το πρόβλημα του είναι ότι, αν και όλες οι ψήφοι θα ανήκαν πράγματι σε ένα εύρος, ποιος θα μου εξασφάλιζε ότι θα υπήρχε μία σχετική κατάταξη. Δηλαδή, με ποιον τρόπο θα ήμουν σίγουρος ότι ένας κακόβουλος δεν θα επέλεγε να βαθμολογήσει 2 ή και παραπάνω υποψηφίους με την ίδια τιμή. Η λύση σε αυτό το πρόβλημα δίνεται αν πέρα από τις ατομικές αποδείξεις μηδενικής γνώσης για το κάθε κρυπτοκείμενο ξεχωριστά, συμπεριλάβουμε και αποδείξεις NEQ στο σύστημα μας.

Για να είμαστε πιο ακριβείς θα έπρεπε να συμπεριλάβουμε ακόμη $\frac{n(n-1)}{2}$ NEQ αποδείξεις, δηλαδή θα έπρεπε να αποδεικνύουμε ανά-2 ότι τα κρυπτοκείμενα μας δεν κρυπτογραφούν την ίδια τιμή. Όπως παρατηρούμε, ένα τέτοιο σύστημα αποδείξεων θα μας εξασφάλιζε ότι οι n επιλογές έχουν λάβει όλες διακριτές και διαφορετικές τιμές μεταξύ τους ενώ ταυτόχρονα ανήκουν και στο κατάλληλο διάστημα. Συ-

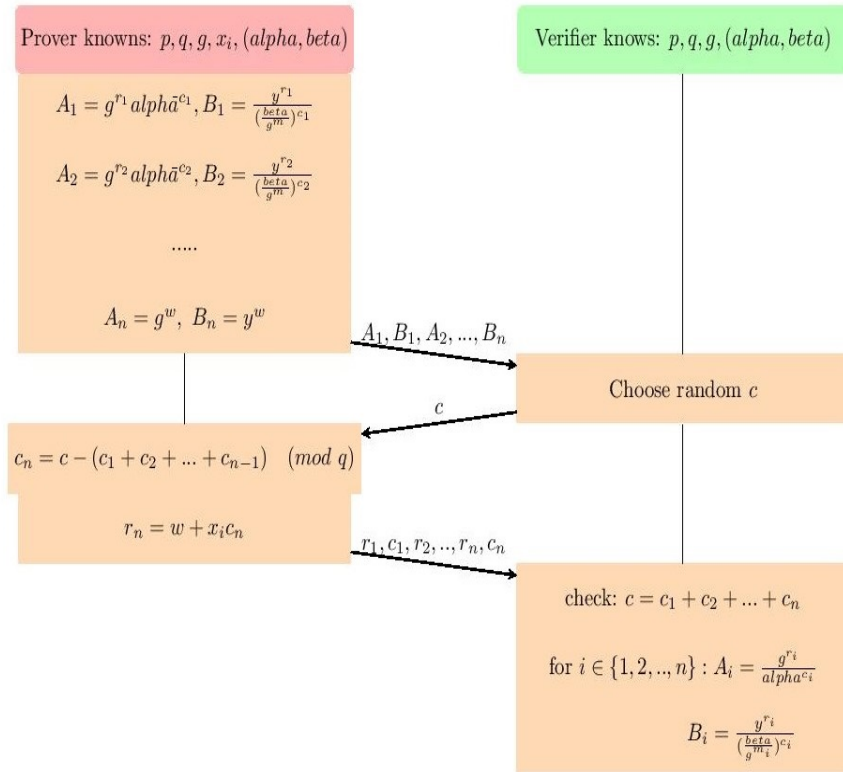


Figure 4.1: Or-proofs(0-max)

νεπώς, με το παραπάνω θα ήταν πλήρως επιτεύξιμο ένα σύστημα κατάταξης των υποψηφίων από τους ψηφοφόρους.

Από την άλλη μεριά, ένα τέτοιο σύστημα παρουσιάζει αυξημένη πολυπλοκότητα (της τάξεως του n^2) ενώ και ο χώρος των αποδείξεων θα ήταν πολύ μεγάλος, καθιστώντας έτσι τον πίνακα ανακοινώσεων ιδιαίτερα δύσχρηστο, και άρα την ελεγχσιμότητα της εκλογικής διαδικασίας εξίσου περίπλοκη. Βεβαίως, θα μπορούσαμε με μετασχηματισμό του κρυπτοκειμένου και της διαδικασίας κρυπτογράφησης, διαλέγοντας ένα διαφορετικό πρωτόκολλο κρυπτογράφησης από το El gamal και έχοντας ένα (πλήρες) ομομορφικό σχήμα, θα μπορούσαμε πιθανότατα να αποφύγουμε τις αποδείξεις NEQ. Σε κάθε περίπτωση πάντως δεν μπορούμε να γλιτώσουμε τις αποδείξεις μηδενικής γνώσεως που χρειάζονται για να αποδείξουμε ότι το κρυπτοκειμένο μας κρυπτογραφεί μία τιμή στο $[0, n - 1]$, και άρα η διαδικασία θα συνέχιζε να είναι εξίσου περίπλοκη.

Συμπερασματικά, αν και η παραπάνω ιδέα αποτελεί μία λύση στο πρόβλημα της κατάταξης των ψηφοφόρων, εντούτοις, δεν είναι αποδοτική και εφαρμόσιμη, για μεγάλες τουλάχιστον ψηφοφορίες. Έτσι, στραφήκαμε σε άλλου τύπου αποδείξεις οι οποίες τελικά είναι και η λύση στο πρόβλημα μας. Ακολουθεί, λοιπόν, η τελική ιδέα υλοποίησης του τροποποιημένου συστήματος.

4.3 Τελική Ιδέα Υλοποίησης

Η τελική ιδέα υλοποίησης του τροποποιημένου συστήματος στηρίζεται στο γεγονός ότι τα τελικά n κρυπτοκείμενα, για να προσφέρουν μία κατάταξη των ψηφοφόρων θα πρέπει να κρυπτογραφούν n διακεκριμένες σταθερές τιμές. Βάση ακριβώς αυτής της ιδιότητας τους, παρατηρούμε ότι κάθε πιθανή κρυπτογράφηση αποτελεί επί της ουσίας μία τυχαία (προς κάποιον τρίτο) επανακρυπτογράφηση κάποιων σταθερών αρχικών κειμένων.

Συνεπώς, η βασική ιδέα στηρίζεται στο γεγονός ότι αρκεί να αποδείξει ο κάθε ψηφοφόρος ότι ανακάτεψε και επανακρυπτογράφησε σωστά κάποια αρχικά κείμενα. Άρα, αρκεί να αποδείξει ο κάθε ψηφοφόρος ότι είναι ένα σωστό Mixnet και ότι εφάρμοσε ένα σωστό ανακάτεμα, μέσω μίας απόδειξης μηδενικής γνώσης. Τέτοιου είδους αποδείξεις υπάρχουν πάρα πολλές υλοποιημένες [20, 17, 19, 10] και εμείς όμως έχουμε ξαναπεί, θα επιλέξουμε την πρώτη αποδοτική απόδειξη του είδους της που δεν είναι άλλη από εκείνη που παρουσιάζεται στο paper των Jun Furukawa and Kazue Sako με τίτλο An Efficient Scheme for Proving a Shuffle

Στην επόμενη ενότητα αναλύεται πλήρως η παραπάνω Shuffle-proof ενώ παρουσιάζομαι και πώς ακριβώς θα εφαρμοστεί στο σύστημα μας. Τέλος, γίνεται πλέον σαφές ότι σχεδόν όλες οι πράξεις και ιδιότητες θα παραμείνουν ίδιες στο τροποποιημένο σύστημα με την μόνη σημαντική διαφορά να είναι η αλλαγή των ατομικών-συνολικών αποδείξεων και αντικατάσταση τους από το νέο σύστημα αποδείξεων.

4.4 Ανάλυση του Νέου Συστήματος Αποδείξεων

Σε αυτήν την ενότητα θα αναφερθούμε στην ανάλυση της Shuffle-proof που θα χρησιμοποιηθεί για την απόδειξη της ορθότητας της ψήφου από τον ψηφοφόρο, αλλά στις δικές μας συνθήκες. Αρχικά ας θεωρήσουμε ότι τα αρχικά κρυπτοκείμενα των επιλογών μου είναι τα (E_1, E_2, \dots, E_n) με $E_i = (g^{x_i}, y^{x_i} g^{m_i})$ και χβγ μπορούμε να θεωρήσουμε ότι $m_i = i$. Η επιδίωξη μας είναι να μπορεί να αποδεικνύει ο ψηφοφόρος ότι τα τελικά κρυπτοκείμενα, έστω $(E'_1, E'_2, \dots, E'_n)$, είναι μία "τυχαία" επανακρυπτογραφημένη εναλλαγή των αρχικών (E_1, E_2, \dots, E_n) , χωρίς βεβαίως να γίνεται γνωστή ακριβώς αυτή η εναλλαγή, ενώ $E'_i = (\alpha_i, \beta_i) = (g^{r_i} \alpha_{\pi^{-1}(i)}, y^{r_i} \beta_{\pi^{-1}(i)})$ με r_i να αποτελεί την τυχαιότητα της κρυπτογράφησης για το κρυπτοσύστημά μας.

Ουσιαστικά, λοιπόν θέλουμε να αποδείξουμε ότι υπάρχει μία συνάρτηση εναλλαγής, έστω π με $\pi, 1-1$ και επί, ενώ $\pi(i) = j, i, j \in [0, n]$ τέτοια ώστε να

θεωρήσουμε ως $Dec()$ την διαδικασία αποκρυπτογράφησης, τότε θα πρέπει να δείξουμε ότι $Dec(E'_i) = Dec(E_{\pi^{-1}(i)})$, χωρίς να αποκαλύπτουμε την συνάρτηση π και χωρίς προφανώς να αποκρυπτογραφούμε τα κρυπτοκείμενα.

Στα πλαίσια, λοιπόν της απόδειξης του παραπάνω η απόδειξη που χρησιμοποιούμε στηρίζεται στον πίνακα ανακατέματος/μεταλλαγής (Permutation-matrix). Ένας τέτοιος πίνακας έστω, A_{ij} , ορίζεται από την συνάρτηση ϕ ως εξής:

$$A_{ij} = \begin{cases} 1 \text{ mod } q, & \text{if } \pi(i) = j \\ 0 \text{ mod } q, & \text{otherwise} \end{cases}$$

Χρησιμοποιώντας λοιπόν, τον παραπάνω πίνακα αρκεί να αποδείξουμε ότι:

$$(\alpha'_i, \beta'_i) = (g^{r_i} \alpha_{\pi^{-1}(i)}, y^{r_i} \beta_{\pi^{-1}(i)}) = (g^{r_i} \prod_{j=1}^n g^{x_j A_{ji}}, y^{r_i} \prod_{j=1}^n (y^{x_j} g^j)^{A_{ji}}) \pmod{p}$$

Πρέπει λοιπόν για να ισχύει το παραπάνω να αποδείξουμε πρώτον, ότι για κάθε (α_i, β_i) έχουν χρησιμοποιηθεί τα ίδια r_i, A_{ij} όπως φαίνονται στον παραπάνω τύπο και δεύτερον ότι ο A_{ij} είναι πράγματι ένας Permutation-matrix. Η πρώτη συνθήκη αποδεικνύεται εύκολα και αποδοτικά, και απλά θα παρουσιαστεί στο τέλος ο τρόπος που γίνεται αυτό. Η δεύτερη συνθήκη, ότι δηλαδή ο A_{ij} είναι πράγματι ένας Permutation-matrix είναι σαφώς πιο περίπλοκη διαδικασία και θα αναλυθεί παρακάτω.

Αρχικά, βάση μαθηματικής απόδειξης που παρέχεται στο paper αρκεί να δείξουμε την ισχύ 2 εξισώσεων οι οποίες είναι:

$$\sum_{h=1}^n A_{hi} A_{hj} = \begin{cases} 1 \text{ mod } q, & \text{if } i = j \\ 0 \text{ mod } q, & \text{otherwise} \end{cases}$$

και

$$\sum_{h=1}^n A_{hi} A_{hj} A_{hk} = \begin{cases} 1 \text{ mod } q, & \text{if } i = j = k \\ 0 \text{ mod } q, & \text{otherwise} \end{cases}$$

Η απόδειξη ότι οι παραπάνω ισχυρισμοί αρκούν για να ισχύει ότι ο A είναι Permutation-matrix είναι καθαρά μαθηματική και δεν παρουσιάζεται στα πλαίσια αυτής της εργασίας. Παρ' όλα αυτά όποιος ενδιαφέρεται μπορεί να την βρει στο paper. Στην συνέχεια παρουσιάζονται 2 αποδείξεις μηδενικής μία για κάθε μαθηματική πρόταση, ενώ στην συνέχεια παρουσιάζεται το συνολικό πρωτόκολλο. Ακόμη, για χάρην ευκολίας μπορούμε να θεωρήσουμε την συνάρτηση δ (παρόμοια με την ντιράκ) για την οποία ισχύει ότι

$$\delta_{ij} = \begin{cases} 1 \text{ mod } q, & \text{if } i = j \\ 0 \text{ mod } q, & \text{otherwise} \end{cases}$$

και

$$\delta_{ijk} = \begin{cases} 1 \text{ mod } q, & \text{if } i = j = k \\ 0 \text{ mod } q, & \text{otherwise} \end{cases}$$

συνεπώς θα μπορέσουμε να απλοποιήσουμε τον τύπο για τον πίνακα A . Τέλος, προκύπτει από τις αποδείξεις για τις 2 συνθήκες για τον πίνακα A , ότι θα έχει ένα μη-μηδενικό στοιχείο ανά γραμμή και ένα μη-μηδενικό στοιχείο ανά στήλη, ενώ και αυτό το στοιχείο θα είναι ίσο με $1 \pmod{q}$. Στις επόμενες υποενότητες παρουσιάζουμε τις ζητούμενες αποδείξεις μηδενικής γνώσης.

4.4.1 Απόδειξη του $\sum_{h=1}^n A_{hi}A_{hj} = \delta_{ij}$

Η βασική ιδέα είναι να δημιουργηθούν τα $s = \sum_{j=1}^n r_j c_j$ και τα $s_i = \sum_{j=1}^n A_{ij} c_j$ σαν απάντηση στα $\{c_j\}$ και έπειτα ο V επαληθεύει ότι:

$$\sum_{i=1}^n s_i^2 = \sum_{j=1}^n c_j^2 \pmod{q}$$

και

$$g^s \prod_{i=1}^n \alpha_i^{s_i} = \prod_{j=1}^n \alpha_j'^{c_j} \pmod{p}$$

Αλλά το παραπάνω δεν είναι χρησιμοποιήσιμο καθώς διαφέρει πληροφορία για τον πίνακα A , συνεπώς για να αποφευχθεί αυτό, προσθέτουμε τυχαιότητες α και α_i στα s και s_i αντίστοιχα. Τότε θα ισχύει το ακόλουθο:

$$\sum_{i=1}^n s_i^2 = \sum_{j=1}^n c_j^2 + \sum_{j=1}^n B_j c_j + D \pmod{q}$$

με τα B_j, D να είναι τετραγωνικά πολυώνυμα του A , και των α_i , άρα θα πρέπει να στέλνονται εκ των προτέρων μαζί με το $g^\alpha \prod_{i=1}^n \alpha_i^{\alpha_i}$ για να μπορεί να πιστοποιηθεί η απόδειξη. Τέλος προσθέτουμε ακόμη μία τυχαιότητα έστω σ και η τελική εξίσωση γίνεται:

$$\sum_{i=1}^n s_i^2 + \sigma s = \sum_{j=1}^n c_j^2 + \sum_{j=1}^n (B_j + \sigma r_j) c_j + (D + \sigma \alpha) \pmod{q}$$

Με βάση τα παραπάνω, και υπό της προϋποθέσεις του τροποποιημένου συστήματός μας θα ισχύουν τα κάτωθι:

- Ο P παράγει τυχαίους αριθμούς $\sigma, \alpha, \{\alpha_i\} \in Z_q$ για $i \in [1, n]$ και υπολογίζει τα εξής:

$$w = g^\sigma \pmod{p}$$

$$g' = g^\alpha \prod_{j=1}^n \alpha_j^{\alpha_j} \pmod{p}$$

$$\dot{w}_i = g^{\sum_{j=1}^n 2a_j A_{ji} + \sigma r_i} (= g^{B_i + \sigma r_i}), \forall i \in [1, n] \pmod{p}$$

$$\dot{w} = g^{\sum_{j=1}^n \alpha_j^2 + \sigma \alpha} (= g^{D + \sigma \alpha}), \pmod{p}$$

- Ο V απαντάει με τυχαία $c_i \in Z_q, \forall i \in [1, n]$. Προφανώς στο τελικό non-interactive πρωτόκολλο αυτές οι προκλήσεις θα παράγονται από μία συνάρτηση κατακερματισμού σε κάποια δεδομένα.
- Ο P αφού λάβει τα c_i παράγει τα $s = \sum_{j=1}^n r_j c_j + \alpha \pmod{q}$ και τα $s_i = \sum_{j=1}^n A_{ij} c_j + \alpha_i \pmod{q}, \forall i \in [1, n]$ και τα στέλνει στον V .
- Τέλος ο V κάνει τους εξής ελέγχους:

$$g^s \prod_{j=1}^n \text{alpha}_j^{s_j} \stackrel{?}{=} g' \prod_{j=1}^n \text{alpha}'_j^{c_j} \pmod{p}$$

και

$$w^s g^{\sum_{j=1}^n (s_j^2 - c_j^2)} \stackrel{?}{=} \dot{w} \prod_{j=1}^n \dot{w}_j^{c_j} \pmod{p},$$

Οι ιδιότητες της παραπάνω απόδειξης, είναι ότι αν ένας V αποδεχθεί την παραπάνω απόδειξη, τότε είτε ο P γνωρίζει και τα r_i και τον πίνακα A , ή μπορεί να βρει $\alpha, \{\alpha_i\}$ τέτοια ώστε $g^\alpha \prod_{i=1}^n g_i^{\alpha_i} = 1$ ή πιο απλά στην περίπτωση μας $g^\alpha = 1$. Ακόμη, το πρωτόκολλο είναι μηδενικής γνώσης και η δυσκολία του βασίζεται στο ότι αν μπορεί να λυθεί τότε μπορεί να λυθεί και το DDH. Οι αποδείξεις των τελευταίων ισχυρισμών, υπάρχουν αναλυτικά στο paper αλλά δεν παρουσιάζονται εδώ χάριν ευκολίας.

4.4.2 Απόδειξη του $\sum_{h=1}^n A_{hi} A_{hj} A_{hk} = \delta_{ijk}$

Αυτή η απόδειξη είναι εντελώς ανάλογη της παραπάνω απόδειξης. Το σύστημα λοιπόν έχει ως εξής:

- Ο P παράγει τυχαίους αριθμούς $\rho, \tau, \alpha, \{\alpha_i\}, \lambda, \{\lambda_i\} \in Z_q$ για $i \in [1, n]$ και υπολογίζει και στέλνει στον V τα εξής:

$$t = g^\tau, v = g^\rho, u = g^\lambda, u_i = g^{\lambda_i} \pmod{p}, \forall i \in [1, n]$$

$$g' = g^\alpha \prod_{j=1}^n \text{alpha}_j^{\alpha_j} \pmod{p}$$

$$\dot{t}_i = g^{\sum_{j=1}^n 3a_j A_{ji} + \tau \lambda_i} \pmod{p}, \forall i \in [1, n]$$

$$\dot{u}_i = g^{\sum_{j=1}^n 3a_j^2 A_{ji} + \rho r_i} \pmod{p}, \forall i \in [1, n]$$

$$\dot{u} = g^{\sum_{j=1}^n a_j^3 + \tau \lambda + \rho \alpha} \pmod{p}$$

- Ο V απαντάει με τυχαία $c_i \in Z_q, \forall i \in [1, n]$. Προφανώς στο τελικό non-interactive πρωτόκολλο αυτές οι προκλήσεις θα παράγονται από μία συνάρτηση κατακερματισμού σε κάποια δεδομένα.

- Ο P αφού λάβει τα c_i παράγει τα $s = \sum_{j=1}^n r_j c_j + \alpha \pmod{q}$ και τα $s_i = \sum_{j=1}^n A_{ij} c_j + \alpha_i \pmod{q}$, $\forall i \in [1, n]$, τα $\lambda' = \sum_{j=1}^n \lambda_j c_j^2 + \lambda \pmod{q}$ και τα στέλνει στον V .
- Τέλος ο V κάνει τους εξής ελέγχους:

$$g^s \prod_{j=1}^n \alpha_j^{s_j} \stackrel{?}{=} g' \prod_{j=1}^n \alpha_j'^{c_j} \pmod{p}$$

$$g^{\lambda'} \stackrel{?}{=} u \prod_{j=1}^n u_j^{c_j^2} \pmod{p}$$

και

$$t^{\lambda'} v^s g^{\sum_{j=1}^n (s_j^3 - c_j^3)} \stackrel{?}{=} v \prod_{j=1}^n u_j^{c_j} t_j^{c_j^2} \pmod{p},$$

Οι ιδιότητες της παραπάνω απόδειξης, είναι ότι αν ένας V αποδεχθεί την παραπάνω απόδειξη, τότε είτε ο P γνωρίζει και τα r_i και τον πίνακα A , ή μπορεί να βρει $\alpha, \{\alpha_i\}$ τέτοια ώστε $g^\alpha \prod_{i=1}^n g_i^{\alpha_i} = 1$ ή πιο απλά στην περίπτωση μας $g^\alpha = 1$. Ακόμη, το πρωτόκολλο είναι μηδενικής γνώσης και η δυσκολία του βασίζεται στο ότι αν μπορεί να λυθεί τότε μπορεί να λυθεί και το DDH. Οι αποδείξεις των τελευταίων ισχυρισμών, υπάρχουν αναλυτικά στο paper αλλά δεν παρουσιάζονται εδώ χάριν ευκολίας.

4.4.3 Η Συνολική Απόδειξη

Το συνολικό πρωτόκολλο είναι ένα συνδυασμός των δύο προαναφερθέντων με κάποιες προσθήκες. Αυτές οι προσθήκες έχουν να κάνουν με το γεγονός ότι οι παραπάνω αποδείξεις δεν περιέχουν τον ακριβή ορισμό της ορθότητας καθώς υπάρχει η πιθανότητα ο P να φτιάξει τα α , έτσι ώστε $g^\alpha \prod_{i=1}^n g_i^{\alpha_i} = 1$, αφού τα α , επιλέγονται από τον ίδιο τον P . Συνεπώς για να αποφευχθεί το παραπάνω αναγκάζουμε τον P να πραγματοποιήσει ό,τι έκανε στα g, g_1, g_2, \dots να τα κάνει και σε ένα τυχαίο σετ $\tilde{g}, \tilde{g}_1, \dots, \tilde{g}_n$, με συνέπεια να μην είναι δυνατό να βρει α, α_i τέτοια ώστε $g^\alpha \prod_{i=1}^n g_i^{\alpha_i} = 1$ και $\tilde{g}^\alpha \prod_{i=1}^n \tilde{g}_i^{\alpha_i} = 1$ (βάση της υπόθεσης της δυσκολίας του διακριτού λογαρίθμου). Τέλος, το συνολικό πρωτόκολλο, εκτελείται παράλληλα για τα g, \tilde{g} και θα έχουμε ένα πρωτόκολλο ως συνδυασμό των παραπάνω δύο πρωτοκόλλων με τις πρόσθετες συνθήκες για τα \tilde{g} ενώ προφανώς εφαρμόζουμε και τις κατάλληλες συνθήκες για τα β ώστε να έχουμε πληρότητα του συστήματος.

Άρα, το συνολικό σύστημα εφαρμοσμένο στο τροποποιημένο σύστημα μας θα είναι:

Θεωρούμε ως κοινή είσοδο όλων τα $p, q, g, y, \tilde{g}, \{\tilde{g}_i\}, \{\alpha_i, \beta_i\}, \{\alpha_i', \beta_i'\}$ και έχουμε

- Ο P παράγει τυχαίους αριθμούς $\sigma, \rho, \tau, \alpha, \{\alpha_i\}, \lambda, \{\lambda_i\} \in \mathbb{Z}_q$ για $i \in [1, n]$ και υπολογίζει και στένει στον V τα εξής (στο σύστημα μας απλώς τα γράφει στην απόδειξη):

$$t = g^\tau, v = g^\rho, w = g^\sigma, u = g^\lambda, u_i = g^{\lambda_i} \pmod{p}, \forall i \in [1, n]$$

$$\tilde{g}'_i = \tilde{g}^{r_i} \prod_{j=1}^n \tilde{g}_j^{A_{ji}} \pmod{p}, \forall i \in [1, n]$$

$$\tilde{g}' = \tilde{g}^\alpha \prod_{j=1}^n \tilde{g}_j^{\alpha_j} \pmod{p}$$

$$g' = g^\alpha \prod_{j=1}^n \text{alpha}_j^{\alpha_j} \pmod{p}$$

$$m' = y^\alpha \prod_{j=1}^n \text{beta}_j^{\alpha_j} \pmod{p}$$

$$t'_i = g^{\sum_{j=1}^n 3a_j A_{ji} + \tau \lambda_i} \pmod{p}, \forall i \in [1, n]$$

$$u_i = g^{\sum_{j=1}^n 3a_j^2 A_{ji} + \rho r_i} \pmod{p}, \forall i \in [1, n]$$

$$\dot{u} = g^{\sum_{j=1}^n a_j^3 + \tau \lambda + \rho \alpha} \pmod{p}$$

$$\dot{w}_i = g^{\sum_{j=1}^n 2a_j A_{ji} + \sigma r_i}, \forall i \in [1, n] \pmod{p}$$

$$\dot{w} = g^{\sum_{j=1}^n a_j^2 + \sigma \alpha}, \pmod{p}$$

- Ο V απαντάει με τυχαία $c_i \in Z_q, \forall i \in [1, n]$. Προφανώς στο τελικό non-interactive πρωτόκολλο αυτές οι προκλήσεις θα παράγονται από μία συνάρτηση κατακερματισμού σε κάποια δεδομένα όπως πχ $c_i = H(\text{alpha}_i, \text{beta}_i, \text{alpha}'_i, \text{beta}'_i, t_i, \dot{w}_i, g)$ ή ακόμη και σε λιγότερα δεδομένα χωρίς βλάβη καθώς έχουμε δώσει και ιδιωτικά και δημόσια δεδομένα μέσα στα c_i .
- Ο P αφού λάβει τα c_i παράγει τα

$$s = \sum_{j=1}^n r_j c_j + \alpha \pmod{q}$$

$$s_i = \sum_{j=1}^n A_{ij} c_j + \alpha_i \pmod{q}, \forall i \in [1, n]$$

$$\lambda' = \sum_{j=1}^n \lambda_j c_j^2 + \lambda \pmod{q}$$

και τα στένει στον V ή απλά στο σύστημα μας τα δημοσιεύει.

- Τέλος ο V , ή στο σύστημα μας όποιος θέλει, κάνει τους εξής ελέγχους:

$$g^s \prod_{j=1}^n \alpha_j^{s_j} \stackrel{?}{=} g' \prod_{j=1}^n \alpha_j'^{c_j} \pmod{p},$$

$$\tilde{g}^s \prod_{j=1}^n \tilde{g}_j^{s_j} \stackrel{?}{=} \tilde{g}' \prod_{j=1}^n \tilde{g}_j'^{c_j} \pmod{p}, \quad g_i = 1$$

$$y^s \prod_{j=1}^n \beta_j^{s_j} \stackrel{?}{=} m' \prod_{j=1}^n \beta_j'^{c_j} \pmod{p}$$

$$g^{\lambda'} \stackrel{?}{=} u \prod_{j=1}^n u_j^{c_j^2} \pmod{p}$$

$$t^{\lambda'} v^s g^{\sum_{j=1}^n (s_j^3 - c_j^3)} \stackrel{?}{=} \dot{v} \prod_{j=1}^n \dot{u}_j^{c_j} \dot{t}_j^{c_j^2} \pmod{p},$$

$$w^s g^{\sum_{j=1}^n (s_j^2 - c_j^2)} \stackrel{?}{=} \dot{w} \prod_{j=1}^n \dot{w}_j^{c_j} \pmod{p},$$

4.4.4 Το Τελικό Σύστημα σε Θεωρητικό Επίπεδο

Παραπάνω αναφέραμε την απόδειξη τελείως θεωρητικά και πιθανώς κάποιος να μην μπορεί να αντιληφθεί πλήρως το πως ακριβώς αποδεικνύεται ότι έχουμε σωστή εναλλαγή των ψήφων. Γι' αυτό θα παρουσιαστεί στο επόμενο κεφάλαιο ένα πρακτικό παράδειγμα όπου θα εξηγηθεί πως ακριβώς γίνεται η απόδειξη της ορθής λειτουργίας του συστήματος αποδείξεων. Τέλος, σε θεωρητικό επίπεδο οι αλλαγές που χρειάζεται το σύστημα περιορίζονται μονάχα στην αλλαγή της τιμής κρυπτογραφημένης ποσότητας, η οποία από 0 και 1 πλέον είναι από 0 έως $n - 1$, και η απόδειξη της ορθότητας της ψήφου. Βεβαίως, στο επόμενο κεφάλαιο θα αναφερθούμε εκτενέστερα και σε τυχόν άλλα προβλήματα που είναι απαραίτητο να επιλυθούν στην πρακτική υλοποίηση του συστήματος.

Κεφάλαιο 5

Πρακτική Υλοποίηση του Νέου Συστήματος

5.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα αναφερθούμε στην πρακτική υλοποίηση του θεωρητικού συστήματος που αναφέρθηκε στο προηγούμενο. Πιο συγκεκριμένα, θα αναφερθεί η πρακτική εφαρμογή της νέας απόδειξης μηδενικής γνώσης μέσω ενός πρακτικού ολοκληρωμένου παραδείγματος. Ακόμη, θα επισημάνουμε ορισμένα κομμάτια κώδικα όπως και φωτογραφίες λειτουργίας του συστήματός μας, ενώ θα παρουσιαστεί και σε μία υποενότητα το αρχείο που περιέχει τις αλλαγές που έγιναν στον πηγαίο κώδικα του αρχικού συστήματος. Τέλος, θα μιλήσουμε για ό,τι χρειάζεται αν κάποιος θέλει να ενσωματώσει το σύστημα που υλοποιείται ώστε να πραγματοποιήσει ψηφοφορίες κατάταξης, όπως και για τυχόν απαραίτητες προσθήκες και ελέγχους οι οποίοι είναι απαραίτητοι προτού κάποιος το χρησιμοποιήσει.

5.2 Ένα Απλό Ολοκληρωμένο Πρακτικό Παράδειγμα

Σε αυτήν την ενότητα θα μελετηθεί λεπτομερώς η διαδικασία της ψηφοφορίας. Για την ακρίβεια θα αναφέρουμε συνοπτικά την διαδικασία που παραμένει ίδια με του κλασικού συστήματος Helios ενώ θα μελετήσουμε σαφώς εκτενέστερα τις πιθανές τροποποιήσεις στην διαδικασία ψηφοφορίας.

Αρχικά, ας υποθέσουμε κατά τα γνωστά ότι ο P είναι ο ψηφοφόρος μας και έστω ότι θέλει να συμμετάσχει σε μία ψηφοφορία. Αρχικά, λοιπόν πρέπει να έχει δημιουργηθεί αυτή η ψηφοφορία. Η διαδικασία της δημιουργίας παραμένει εντελώς ανάλογη με του αρχικού συστήματος. Στην συνέχεια ο ψηφοφόρος αφού δεχτεί το mail με τους κωδικούς κατά τα γνωστά, μπορεί πλέον να παράξει μία έγκυρη ψήφο. Έτσι εισέρχεται στο σύστημα και διαλέγει την ψηφοφορία που θέλει να συμμετάσχει. Κατά τα γνωστά, μπορεί να περιηγηθεί στις ερωτήσεις, με την δια-

φορά ότι τώρα δεν επιλέγει με ένα απλό τικάρισμα, αλλά αντιθέτως δίπλα από κάθε επιλογή δίνεται η δυνατότητα να βάλει μία τιμή προτίμησης από 0 έως $n - 1$, σαν βαθμολογία για την εκάστοτε επιλογή.

Όταν αρχίζει η διαδικασία κρυπτογράφησης είναι που παρουσιάζονται και οι πρώτες σημαντικές διαφορές. Αρχικά, το σύστημα τώρα κρυπτογραφεί μία τιμή 0 έως $n - 1$ και άρα πρέπει να παραχθεί και μία ανάλογη απόδειξη μηδενικής γνώσης για την ορθότητα της ψήφου. Ας υποθέσουμε για λόγους απλότητας και χβγ ότι ο ψηφοφόρος έχει να κατατάξει 4 υποψηφίους αντί για n για να είναι πιο εμφανείς οι πράξεις για την απόδειξη. Ας υποθέσουμε ακόμη ότι αυτοί οι 4 υποψήφιοι παίρνουν διαδοχικά 3,2,1,0 ως βαθμολογία πάλι χβγ. Τότε, θα έχω ότι τα 4 κρυπτοκείμενα ως εξής:

$$\begin{aligned}(\alpha'_1, \beta'_1) &= (g^{r^1}, y^{r^1} g^3) \pmod{p} \\(\alpha'_2, \beta'_2) &= (g^{r^2}, y^{r^2} g^2) \pmod{p} \\(\alpha'_3, \beta'_3) &= (g^{r^3}, y^{r^3} g^1) \pmod{p} \\(\alpha'_4, \beta'_4) &= (g^{r^4}, y^{r^4} g^0) \pmod{p}\end{aligned}$$

Ενώ θα πρέπει να αποδείξω ότι τα παραπάνω κρυπτοκείμενα αποτελούν μία επανακρυπτογράφηση κρυπτοκειμένων που κρυπτογραφούν τις τιμές 0,1,2,3 χωρίς προφανώς να δείχνω την εναλλαγή που έχω εφαρμόσει. Τα αρχικά κρυπτοκείμενα, τα οποία θα επανακρυπτογραφήσω μπορώ να θεωρήσω ότι έχουν ό,τι τυχαιότητα επιθυμώ, και για λόγους απλότητας θεωρώ τυχαιότητα ίση με 0 ή i ή οποιαδήποτε άλλη τιμή αρκεί να την παρουσιάζω ώστε ο καθένας να μπορεί να ελέγξει ότι τα αρχικά κρυπτοκείμενα πράγματι κρυπτογραφούν τις επιτρεπτές τιμές (εμείς θα διαλέξουμε την τυχαία περίπτωση για τυχαιότητα ίση με x_i και στο τέλος απλά θα δημοσιεύσουμε αυτήν την τυχαιότητα). Άρα τα αρχικά κείμενα τα οποία θα επανακρυπτογραφηθούν και θα εναλλαχθούν θα είναι τα:

$$\begin{aligned}(\alpha_1, \beta_1) &= (g^{x_1}, y^{x_1} g^0) \pmod{p} \\(\alpha_2, \beta_2) &= (g^{x_2}, y^{x_2} g^1) \pmod{p} \\(\alpha_3, \beta_3) &= (g^{x_3}, y^{x_3} g^2) \pmod{p} \\(\alpha_4, \beta_4) &= (g^{x_4}, y^{x_4} g^3) \pmod{p}\end{aligned}$$

Μπορούμε φυσικά να θεωρήσουμε τυχαιότητα $x_i = H(\alpha'_i, \beta'_i)$ με H μία συνάρτηση κατακερματισμού και να μην δημοσιεύουμε καν τα x_i αφού μπορεί κάποιος να τα υπολογίσει, ενώ προφανώς η τυχαιότητα επανακρυπτογράφησης θα είναι ίση με $r_\pi = r'_\pi - x_i$.

Επιπλέον, μπορούμε να θεωρήσουμε το x_i πραγματικά τυχαίο και απλά να εισάγουμε n αποδείξεις μηδενικής γνώσης για τα αρχικά $\{\alpha_i, \beta_i\}$, στα οποία θα δείχνουμε με απλά πρωτόκολλα Schnorr (Pedersen) ότι κρυπτογραφούν τις τιμές από 0 έως $n - 1$.

Ακόμη, ο πίνακας A_{ij} θα γίνει στην περίπτωση μας:

$$A_{ij} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Τώρα θα προχωρήσουμε βήμα προς βήμα για να δείξουμε τις 2 απαραίτητες εξισώσεις $\sum_{h=1}^n A_{hi}A_{hj} = \delta_{ij}$, $\sum_{h=1}^n A_{hi}A_{hj}A_{hk} = \delta_{ijk}$ σε συνδυασμό με τις απαραίτητες εφαρμογές στα $beta_i$, μέσω της παρουσίασης του συνολικού συστήματος, και τον τρόπο με τον οποίο διασφαλίζεται αυτό. Αρχικά, θα θεωρήσουμε ότι η βάση \tilde{g} και τα $\{\tilde{g}_i\}$ είναι γνωστά σε όλους όσους θέλουν να ελέγξουν τις αποδείξεις*.

Και θα έχουμε τα εξής:

- Ο P παράγει τυχαίους αριθμούς $\sigma, \rho, \tau, \alpha, \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}, \lambda, \{\lambda_1, \lambda_2, \lambda_3, \lambda_4\} \in Z_q$. Το σ χρησιμοποιείται για την πρώτη απόδειξη, τα $\rho, \tau, \lambda, \{\lambda_1, \lambda_2, \lambda_3, \lambda_4\}$ για την δεύτερη ενώ τα $\alpha, \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$ και για τις 2 όπως θα φανεί και παρακάτω.

Αρχικά, λοιπόν, θα υπολογίσουμε την επανακρυπτογράφηση των $\{\tilde{g}_i\}$ ώστε να εφαρμόσουμε την απόδειξη και επί αυτών, για τους λόγους που έχουμε ήδη αναφέρει:

$$\tilde{g}'_i = \tilde{g}^{r_i} \prod_{j=1}^4 \tilde{g}_j^{A_{ji}} \pmod{p} \Rightarrow$$

$$\tilde{g}'_1 = \tilde{g}^{r_1} \tilde{g}_4, \tilde{g}'_2 = \tilde{g}^{r_2} \tilde{g}_3, \tilde{g}'_3 = \tilde{g}^{r_3} \tilde{g}_2, \tilde{g}'_4 = \tilde{g}^{r_4} \tilde{g}_1$$

Ακόμη, θα υπολογίσουμε για όλους τους τυχαίους αριθμούς που διαλέξαμε παραπάνω το εκθετικό τους (εκτός των α , που θα δούμε τι ακριβώς θα κάνουμε), καθώς δεν θέλουμε να γίνουν γνωστοί, ενώ θέλουμε να μπορούμε να προσφέρουμε και την απόδειξη, άρα όλες οι πράξεις θα γίνονται στον εκθέτη:

$$t = g^\tau, v = g^\rho, w = g^\sigma, u = g^\lambda, \{u_1 = g^{\lambda_1}, u_2 = g^{\lambda_2}, u_3 = g^{\lambda_3}, u_4 = g^{\lambda_4}\} \pmod{p},$$

Επιπρόσθετα, τα α, α_i που χρησιμοποιούνται και στις δύο αποδείξεις δεν είναι τίποτα άλλο από τις τυχαιότητες που προσθέτονται στα s, s_i που θα δώσουμε ως απαντήσεις αργότερα όταν δεχθούμε τις προκλήσεις. Συνεπώς, ούτε αυτά γίνονται ευθέως γνωστά, παρά μόνο το εκθετικό τους. Ακόμη, μιας και θέλουμε να είμαστε σίγουροι ότι δεν θα υπάρξει κακόβουλος P αναγκάζουμε ότι γίνεται στα g να γίνει και στα \tilde{g} , ενώ αφού θέλουμε τα $alpha'_i$ και τα $beta'_i$ να κρυπτογραφούνται με τις ίδιες τυχαιότητες και για τον ίδιο πίνακα A_{ij} θα έχω τα εξής:

$$\tilde{g}' = \tilde{g}^\alpha \prod_{j=1}^4 \tilde{g}_j^{\alpha_j} \pmod{p}$$

$$g' = g^\alpha \prod_{j=1}^4 \text{alpha}_j^{\alpha_j} \pmod{p}$$

$$m' = y^\alpha \prod_{j=1}^4 \text{beta}_j^{\alpha_j} \pmod{p}$$

Στα παραπάνω υπολογίζουμε την δέσμευσή μας στις τιμές των α, α_i που θα χρησιμοποιηθούν σε μετέπειτα στάδιο

Τέλος για την πρώτη απόδειξη τα offset που θα πρέπει προστεθούν είναι τα $(B_i + \sigma r_i)$ το οποίο θα είναι ίσο με $\sum_{j=1}^4 2a_j A_{ji} + \sigma r_i$ και το $D + \sigma \alpha$ το οποίο είναι ίσο με $\sum_{j=1}^4 a_j^2 + \sigma \alpha$. Προφανώς αυτά θα δοθούν κατά τα γνωστά σε εκθετική μορφή:

$$\begin{aligned} \dot{w}_i &= g^{\sum_{j=1}^4 2a_j A_{ji} + \sigma r_i}, \quad \forall i \in [1, 4] \pmod{p} \Rightarrow \\ \dot{w}_1 &= g^{2a_4 + \sigma r_1}, \quad \dot{w}_2 = g^{2a_3 + \sigma r_2}, \\ \dot{w}_3 &= g^{2a_2 + \sigma r_3}, \quad \dot{w}_4 = g^{2a_1 + \sigma r_1} \\ \dot{w} &= g^{\sum_{j=1}^4 a_j^2 + \sigma \alpha}, \quad \pmod{p} \end{aligned}$$

δηλαδή βάση των παραπάνω, $B_1 = 2a_4, B_2 = 2a_3, B_3 = 2a_2, B_4 = 2a_1$ και $D = a_1^2 + a_2^2 + a_3^2 + a_4^2$. Τα παραπάνω δίνονται για τον πολύ απλό λόγο ότι, θα πρέπει με κάποιο τρόπο να εξασφαλίσω τις ισότητες ελέγχου:

$$\sum_{i=1}^4 s_i^2 + \sigma s = \sum_{j=1}^4 c_j^2 + \sum_{j=1}^n (B_j + \sigma r_j) c_j + (D + \sigma \alpha) \pmod{q}$$

που σε συνδυασμό με τις σχέσεις για τα α, α_i πιο πάνω θα ισχύει ότι:

$$s = \sum_{j=1}^4 r_j c_j + a, \quad s_i = \sum_{j=1}^4 A_{ij} c_j + a_i$$

οι οποίες αποτελούν εξέλιξη (προσθέτοντας τις απαραίτητες τυχαιότητες α, α_i) των πιο απλών, που όμως διαρρέουν πληροφορία:

$$\begin{aligned} \sum_{i=1}^4 s_i^2 &= \sum_{j=1}^4 c_j^2 \\ s &= \sum_{j=1}^4 r_j c_j, \quad s_i = \sum_{j=1}^4 A_{ij} c_j \end{aligned}$$

Έτσι, με την εξακρίβωση των παραπάνω ο οποιοσδήποτε μπορεί να θεωρήσει ότι η πρώτη απόδειξη ήταν ορθή. Αυτή ακριβώς η εξακρίβωση θα γίνει στο τέλος αφού έχει λάβει ο P τα c_i , και παράξει τις κατάλληλες απαντήσεις.

Από την άλλη μεριά, για την δεύτερη απόδειξη θα έχω εντελώς ανάλογα την παραγωγή των offset τέτοιων ώστε $E_i = \sum_{j=1}^n 3a_j A_{ji}$, $F_i = \sum_{j=1}^n 3a_j^2 A_{ji}$, $G = \sum_{j=1}^n a_j^3$, στα οποία προφανώς προσθέτουμε και τις κατάλληλες τυχαίες όψεις όπως ακριβώς και στην πρώτη απόδειξη.

$$\begin{aligned} \dot{t}_i &= g^{\sum_{j=1}^4 3a_j A_{ji} + \tau \lambda_i (=E_i + \tau \lambda_i)} \pmod{p}, \forall i \in [1, 4] \Rightarrow \\ \dot{t}_1 &= g^{3a_4 + \tau \lambda_1}, \dot{t}_2 = g^{3a_3 + \tau \lambda_2}, \dot{t}_3 = g^{3a_2 + \tau \lambda_3}, \dot{t}_4 = g^{3a_1 + \tau \lambda_4}, \end{aligned}$$

Από τα παραπάνω, παρατηρούμε ότι $E_1 = 3a_4$, $E_2 = 3a_3$, $E_3 = 3a_2$, $E_4 = 3a_1$

Ακόμη έχουμε:

$$\begin{aligned} \dot{u}_i &= g^{\sum_{j=1}^4 3a_j^2 A_{ji} + \rho \lambda_i (=F_i + \rho \lambda_i)} \pmod{p}, \forall i \in [1, n] \Rightarrow \\ \dot{u}_1 &= g^{3a_4^2 + \rho \lambda_1}, \dot{u}_2 = g^{3a_3^2 + \rho \lambda_2}, \dot{u}_3 = g^{3a_2^2 + \rho \lambda_3}, \dot{u}_4 = g^{3a_1^2 + \rho \lambda_4}, \end{aligned}$$

Από τα παραπάνω, παρατηρούμε ότι $F_1 = 3a_4^2$, $F_2 = 3a_3^2$, $F_3 = 3a_2^2$, $F_4 = 3a_1^2$

$$\dot{u} = g^{\sum_{j=1}^4 a_j^3 + \tau \lambda + \rho \alpha (=G + \tau \lambda + \rho \alpha)} \pmod{p}$$

Από τα παραπάνω, παρατηρούμε ότι $G = a_1^3 + a_2^3 + a_3^3 + a_4^3$

Όλα τα παραπάνω χρειάζονται γιατί τώρα θα γράψουμε στην απάντησή μας το λ το οποίο θα ορίζεται ως $\lambda = \sum_{j=1}^n \lambda_j c_j^2 + \lambda \pmod{q}$, ενώ τώρα θα έχουμε το $\sum_{j=1}^4 s_j^3$ ανάλογα όπως στην πρώτη απόδειξη είχαμε το $\sum_{j=1}^4 s_j^2$ γι' αυτό και χρειαζόταν να στείλουμε τα παραπάνω *offset* ώστε στο τέλος να είναι δυνατοί οι υπολογισμοί. Προφανώς δουλεύουμε πάλι εκθετικά γιατί δεν θέλουμε να διαφρευσει πληροφορία για τον πίνακα A, ούτε για την τυχαιότητα της επανακρυπτογράφησης, ενώ δεν ξεχνάμε να προσθέσουμε και διάφορες τυχαίες μεταβλητές όπως φαίνεται παραπάνω. Τα παραπάνω θα φανούν και στο στάδιο του verification. Τέλος, προφανώς δεν ξεχνάμε ότι για την δεύτερη απόδειξη χρειαζόμαστε και τις αρχικές εξισώσεις με τα α, α_i όπως έχουμε πει και στην αρχή.

- Παράγονται οι προκλήσεις $c_i = H(\alpha_i, \beta_i, \alpha'_i, \beta'_i, t_i, w_i, g)$ (ή ακόμη και με λιγότερα δεδομένα στην συνάρτηση κατακερματισμού).
- Ο P μετά την παραγωγή των c_i , παράγει τις κάτωθι απαντήσεις (και τις δημοσιεύει) οι οποίες πρέπει να ελεγχθούν από οποιονδήποτε.

$$s = \sum_{j=1}^4 r_j c_j + \alpha \pmod{q} \Rightarrow$$

$$s = r_1 c_1 + r_2 c_2 + r_3 c_3 + r_4 c_4 + \alpha$$

$$s_i = \sum_{j=1}^4 A_{ij} c_j + \alpha_i \pmod{q}, \forall i \in [1, 4] \Rightarrow$$

$$s_1 = c_4 + a_1, s_2 = c_3 + a_2, s_3 = c_2 + a_3, s_4 = c_1 + a_4,$$

$$\dot{\lambda} = \sum_{j=1}^4 \lambda_j c_j^2 + \lambda \pmod{q} \Rightarrow$$

$$\dot{\lambda} = \lambda_1 c_1^2 + \lambda_2 c_2^2 + \lambda_3 c_3^2 + \lambda_4 c_4^2 + \lambda$$

Αν ισχύουν τότε πράγματι είχαμε ένα σωστό re-encryption permutation. Ο έλεγχος ακολουθεί στην συνέχεια.

- Το στάδιο της επαλήθευσης λοιπόν γίνεται:

$$\left\{ \begin{array}{l} g^s \prod_{j=1}^4 \alpha_j^{s_j} \stackrel{?}{=} g' \prod_{j=1}^4 g^{r_j c_j} \pmod{p}, \\ \tilde{g}^s \prod_{j=1}^4 \tilde{g}_j^{s_j} \stackrel{?}{=} \tilde{g}' \prod_{j=1}^4 \tilde{g}_j^{c_j} \pmod{p}, \\ y^s \prod_{j=1}^4 \beta_j^{s_j} \stackrel{?}{=} m' \prod_{j=1}^n \beta_j^{c_j} \pmod{p} \end{array} \right\}$$

Οι παραπάνω σχέσεις διασφαλίζουν, αν κάνουμε τις πράξεις στον εκθέτη, ότι (δείχνουμε μόνο για την g αλλά ισχύει και για τις άλλες 2): $g^s = g^{(r_1 c_1 + r_2 c_2 + r_3 c_3 + r_4 c_4)} g^\alpha$ και $\prod_{j=1}^4 \alpha_j^{s_j} = \prod_{j=1}^4 \alpha_j^{a_j} \prod_{j=1}^4 \alpha_j^{\sum A_{ij} c_j}$. Άρα, $g^s \prod_{j=1}^4 \alpha_j^{s_j} = g' \prod_{j=1}^4 g^{r_j c_j}$ αν έχουμε επιλέξει τις τιμές όπως πρέπει. Οι παραπάνω 3 σχέσεις αποτελούν συμπλήρωμα ως προς τον έλεγχο των 2 αρχικών αποδείξεων, ενώ η παρουσία των 2 πέρα από την κλασική διασφαλίζουν την απόδειξη και της επιπλέον απαραίτητης συνθήκης.

Για την πρώτη απόδειξη όπως είδαμε είναι απαραίτητη η παρακάτω συνθήκη:

$$w^s g^{\sum_{j=1}^n (s_j^2 - c_j^2)} \stackrel{?}{=} w \prod_{j=1}^n w_j^{c_j} \pmod{p},$$

σε συνδυασμό πάντα με τις αρχικές. Παραπάνω βλέπουμε ότι στον εκθέτη στο αριστερό μέλος θα έχει σχηματιστεί το $ss + \sum_{j=1}^4 s_j^2 - c_j^2$ το οποίο θα πρέπει να είναι ίσο με το αρχικό offset (επί της ουσίας είναι τα τετραγωνικά υπόλοιπα που έχουμε στείλει, δηλαδή το $D + \sigma a + \sum_{j=1}^4 (B_j + \sigma r_j) c_j$

Τέλος, τα εντελώς ανάλογα ισχύουν και για την δεύτερη απόδειξη μόνο που τώρα θα έχουμε τα κυβικά υπόλοιπα και τα ανάλογα offset που έχουμε στείλει, ενώ εδώ απλά έχουμε και τους ελέγχους για τα λ .

$$g^{\lambda'} \stackrel{?}{=} u \prod_{j=1}^n u_j^{c_j^2} \pmod{p}$$

$$t^{\lambda'} v^s g^{\sum_{j=1}^n (s_j^3 - c_j^3)} \stackrel{?}{=} v \prod_{j=1}^n u_j^{c_j} t_j^{c_j^2} \pmod{p},$$

Στην παραπάνω περίπτωση το reference string της απόδειξης θα είναι το:

$\{\alpha_i', \beta_i'\}, t, v, w, u, \{u_i\}, g', \{\tilde{g}'_i\}, \tilde{g}', m', \{t_i\}, \{t_i\}, \dot{u}, \{\dot{u}_i\}, \{\dot{w}_i\}, \dot{w}, s, \{s_i\}, \dot{\lambda} \dots$

5.3 Κώδικες του Νέου Συστήματος

Σε αυτήν την ενότητα θα παρουσιάσουμε τις σημαντικότερες αλλαγές που εφαρμόσαμε στο σύστημα Helios ώστε αυτό να υποστηρίζει την υλοποίηση ψηφοφοριών κατάταξης. Σε πρώτη φάση, αντικαταστήσαμε πλήρως τα απαραίτητα αρχεία που επιτελούν την κρυπτογράφηση των ψήφων ώστε να κρυπτογραφούν τιμές πέρα από 0 και 1. Ακόμη, αντικαταστήσαμε τα αρχεία που παράγουν και ελέγχουν τις αποδείξεις. Προφανώς, αυτή η αντικατάσταση επέφερε προβλήματα στο κλασικό σύστημα, κάτι που όμως διορθώνετε πανεύκολα απλά έχοντας 2 αρχεία για κάθε περίπτωση(προφανώς μπορούμε να τα βάλουμε και στο ίδιο αρχείο απλά για λόγους ευκολίας αυτό δεν έχει υλοποιηθεί ακόμη). Συνεπώς, παρακάτω θα παρατεθούν οι σημαντικότερες αλλαγές που έγιναν. Οι κυριότερες αλλαγές έγιναν στα αρχεία `elgamal.js`, `helios.js` τα οποία είναι υπεύθυνα για την κρυπτογράφηση της ψήφου και της παραγωγής της απόδειξης. Τα παραπάνω αρχεία παρατίθενται παρακάτω.

```
1
2 //
3 // Helios Protocols
4 //
5 // ben@adida.net
6 //
7 // FIXME: needs a healthy refactor/cleanup based on Class.extend()
8 //
9
10 // extend jquery to do object keys
11 // from http://snipplr.com/view.php?codeview&id=10430
12 /*
13 $.extend({
14     keys: function(obj){
15         var a = [];
16         $.each(obj, function(k){ a.push(k) });
17         return a.sort();
18     }
19 });
20 */
21
22 var UTILS = {};
23
24 UTILS.array_remove_value = function(arr, val) {
25     var new_arr = [];
26     _(arr).each(function(v, i) {
27         if (v !== val) {
28             new_arr.push(v);
29         }
30     });
31
32     return new_arr;
33 };
34
35 UTILS.select_element_content = function(element) {
36     var range;
37     if (window.getSelection) { // FF, Safari, Opera
38         var sel = window.getSelection();
39         range = document.createRange();
40         range.selectNodeContents(element);
```



```

41     sel.removeAllRanges();
42     sel.addRange(range);
43 } else {
44     document.selection.empty();
45     range = document.body.createTextRange();
46     range.moveToElementText(el);
47     range.select();
48 }
49 };
50
51 // a progress tracker
52 UTILS.PROGRESS = Class.extend({
53     init: function() {
54         this.n_ticks = 0.0;
55         this.current_tick = 0.0;
56     },
57
58     addTicks: function(n_ticks) {
59         this.n_ticks += n_ticks;
60     },
61
62     tick: function() {
63         this.current_tick += 1.0;
64     },
65
66     progress: function() {
67         return Math.round((this.current_tick / this.n_ticks) * 100);
68     }
69 });
70
71 // produce the same object but with keys sorted
72 UTILS.object_sort_keys = function(obj) {
73     var new_obj = {};
74     _(_.keys(obj)).each(function(k) {
75         new_obj[k] = obj[k];
76     });
77     return new_obj;
78 };
79
80 //
81 // Helios Stuff
82 //
83
84 HELIOS = {};
85
86 // a bogus default public key to allow for ballot previewing,
87 // nothing more
88 // this public key should not be used ever, that's why the secret
89 // key is
90 // not given.
91 HELIOS.get_bogus_public_key = function() {
92     return ElGamal.PublicKey.fromJSONObject(JSON.parse('{
93 "g": "14887492224963187634282421537186040...",
94 "p": "16328632084933010002384055033805...",
95 "q": "61329566248342901292543872769978...",
96 "y": "8049609819434159960341080485505...."}'));
97 };

```

```

96
97 // election
98 HELIOS.Election = Class.extend({
99   init: function() {
100     },
101
102   toJSONObject: function() {
103     var json_obj = {uuid : this.uuid,
104     description : this.description, short_name : this.short_name,
105     name : this.name,
106     public_key: this.public_key.toJSONObject(), questions : this.
107     questions,
108     cast_url: this.cast_url, frozen_at: this.frozen_at,
109     openreg: this.openreg, voters_hash: this.voters_hash,
110     use_voter_aliases: this.use_voter_aliases,
111     voting_starts_at: this.voting_starts_at,
112     voting_ends_at: this.voting_ends_at};
113
114     return UTILS.object_sort_keys(json_obj);
115   },
116
117   get_hash: function() {
118     if (this.election_hash)
119       return this.election_hash;
120
121     // otherwise
122     return b64_sha256(this.toJSON());
123   },
124
125   toJSON: function() {
126     // FIXME: only way around the backslash thing for now... how
127     // ugly
128     //return jQuery.toJSON(this.toJSONObject()).replace(/\\/g
129     //,"\\");
130     return JSON.stringify(this.toJSONObject());
131   }
132 });
133
134 HELIOS.Election.fromJSONString = function(raw_json) {
135   var json_object = JSON.parse(raw_json);
136
137   // let's hash the raw_json
138   var election = HELIOS.Election.fromJSONObject(json_object);
139   election.election_hash = b64_sha256(raw_json);
140
141   return election;
142 };
143
144 HELIOS.Election.fromJSONObject = function(d) {
145   var e1 = new HELIOS.Election();
146   e1.extend(e1, d);
147
148   // empty questions
149   if (!e1.questions)
150     e1.questions = [];
151
152   if (e1.public_key) {

```

```

149     el.public_key = ElGamal.PublicKey.fromJSONObject(el.public_key)
150     ;
151   } else {
152     // a placeholder that will allow hashing;
153     el.public_key = HELIOS.get_bogus_public_key();
154     el.BOGUS_P = true;
155   }
156   return el;
157 };
158
159 HELIOS.Election.setup = function(election) {
160   return ELECTION.fromJSONObject(election);
161 };
162
163
164 // ballot handling
165 BALLOT = {};
166
167 BALLOT.pretty_choices = function(election, ballot) {
168   var questions = election.questions;
169   var answers = ballot.answers;
170
171   // process the answers
172   var choices = _(questions).map(function(q, q_num) {
173     return _(answers[q_num]).map(function(ans) {
174       return questions[q_num].answers[ans];
175     });
176   });
177
178   return choices;
179 };
180
181
182 // open up a new window and do something with it.
183 UTILS.open_window_with_content = function(content, mime_type) {
184   if (!mime_type)
185     mime_type = "text/plain";
186   if (BigInt.is_ie) {
187     w = window.open("");
188     w.document.open(mime_type);
189     w.document.write(content);
190     w.document.close();
191   } else {
192     w = window.open("data:" + mime_type + "," +
193       encodeURIComponent(content));
194   }
195 };
196
197 // generate an array of the first few plaintexts
198 UTILS.generate_plaintexts = function(pk, min, max) {
199   var last_plaintext = BigInt.ONE;
200
201   // an array of plaintexts
202   var plaintexts = [];
203
204   if (min == null)

```

```

204     min = 0;
205
206     // questions with more than one possible answer, add to the array
207     for (var i=0; i<=max; i++) {
208         if (i >= min)
209             plaintexts.push(new ElGamal.Plaintext(last_plaintext, pk,
210                 false));
211         last_plaintext = last_plaintext.multiply(pk.g).mod(pk.p);
212     }
213     return plaintexts;
214 }
215
216 //
217 // crypto
218 //
219 //
220
221
222 HELIOS.EncryptedAnswer = Class.extend({
223     init: function(question, answer, pk, progress) {
224         // if nothing in the constructor
225         if (question == null)
226             return;
227
228         // store answer
229         // CHANGE 2008-08-06: answer is now an *array* of answers, not
230             just a single integer
231         this.answer = answer;
232
233         // do the encryption
234         var enc_result = this.doEncryption(question, answer, pk, null,
235             progress);
236
237         this.choices = enc_result.choices;
238         this.randomness = enc_result.randomness;
239         this.individual_proofs = enc_result.individual_proofs;
240         this.overall_proof = enc_result.overall_proof;
241     },
242     doEncryption: function(question, answer, pk, randomness, progress
243     ) {
244         var choices = [];
245         var individual_proofs = [];
246         var overall_proof = null;
247
248         // possible plaintexts [question.min .. , question.max]
249         var plaintexts = null;
250         if (question.max != null) {
251             plaintexts = UTILS.generate_plaintexts(pk, question.min,
252                 question.max);
253         }
254         var zero_one_plaintexts = UTILS.generate_plaintexts(pk, 0, 1);
255
256         // keep track of whether we need to generate new randomness

```

```

255     var generate_new_randomness = false;
256     if (!randomness) {
257         randomness = [];
258         generate_new_randomness = true;
259     }
260
261     // keep track of number of options selected.
262     var num_selected_answers = 0;
263
264     // go through each possible answer and encrypt either a g^0 or
265     // a g^1.
266     for (var i=0; i<question.answers.length; i++) {
267         var index, plaintext_index;
268         // if this is the answer, swap them so m is encryption 1 (g)
269         if (_(answer).include(i)) {
270             plaintext_index = 1;
271             num_selected_answers += 1;
272         } else {
273             plaintext_index = 0;
274         }
275
276         // generate randomness?
277         if (generate_new_randomness) {
278             randomness[i] = Random.getRandomInteger(pk.q);
279         }
280
281         choices[i] = ElGamal.encrypt(pk, zero_one_plaintexts[
282             plaintext_index], randomness[i]);
283
284         // generate proof
285         if (generate_new_randomness) {
286             // generate proof that this ciphertext is a 0 or a 1
287             individual_proofs[i] = choices[i].generateDisjunctiveProof(
288                 zero_one_plaintexts, plaintext_index, randomness[i],
289                 ElGamal.disjunctive_challenge_generator);
290         }
291
292         if (progress)
293             progress.tick();
294     }
295
296     if (generate_new_randomness && question.max != null) {
297         // we also need proof that the whole thing sums up to the
298         // right number
299         // only if max is non-null, otherwise it's full approval
300         // voting
301
302         // compute the homomorphic sum of all the options
303         var hom_sum = choices[0];
304         var rand_sum = randomness[0];
305         for (var i=1; i<question.answers.length; i++) {
306             hom_sum = hom_sum.multiply(choices[i]);
307             rand_sum = rand_sum.add(randomness[i]).mod(pk.q);
308         }
309
310         // prove that the sum is 0 or 1 (can be "blank vote" for this
311         // answer)

```

```

305     // num_selected_answers is 0 or 1, which is the index into
        the plaintext that is actually encoded
306     //
307     // now that "plaintexts" only contains the array of
        plaintexts that are possible starting with min
308     // and going to max, the num_selected_answers needs to be
        reduced by min to be the proper index
309     var overall_plaintext_index = num_selected_answers;
310     if (question.min)
311         overall_plaintext_index -= question.min;
312
313     overall_proof = hom_sum.generateDisjunctiveProof(plaintexts,
        overall_plaintext_index, rand_sum, ElGamal.
        disjunctive_challenge_generator);
314
315     if (progress) {
316         for (var i=0; i<question.max; i++)
317             progress.tick();
318     }
319 }
320
321 return {
322     'choices' : choices,
323     'randomness' : randomness,
324     'individual_proofs' : individual_proofs,
325     'overall_proof' : overall_proof
326 };
327 },
328
329 clearPlaintexts: function() {
330     this.answer = null;
331     this.randomness = null;
332 },
333
334 // FIXME: should verifyEncryption really generate proofs?
        Overkill.
335 verifyEncryption: function(question, pk) {
336     //return true;
337     var result = this.doEncryption(question, this.answer, pk, this.
        randomness);
338
339     // check that we have the same number of ciphertexts
340     if (result.choices.length != this.choices.length) {
341         return false;
342     }
343
344     // check the ciphertexts
345     for (var i=0; i<result.choices.length; i++) {
346         if (!result.choices[i].equals(this.choices[i])) {
347             // alert ("oy: " + result.choices[i] + "/" + this.choices[i]
                );
348             return false;
349         }
350     }
351
352     // we made it, we're good
353     return true;

```

```

354 },
355
356 toString: function() {
357     // get each ciphertext as a JSON string
358     var choices_strings = _(this.choices).map(function(c) {return c
        .toString();});
359     return choices_strings.join("|");
360 },
361
362 toJSONObject: function(include_plaintext) {
363     var return_obj = {
364         'choices' : _(this.choices).map(function(choice) {
365             return choice.toJSONObject();
366         }),
367         'individual_proofs' : _(this.individual_proofs).map(function(
368             disj_proof) {
369             return disj_proof.toJSONObject();
370         });
371     };
372
373     if (this.overall_proof != null) {
374         return_obj.overall_proof = this.overall_proof.toJSONObject();
375     } else {
376         return_obj.overall_proof = null;
377     }
378
379     if (include_plaintext) {
380         return_obj.answer = this.answer;
381         return_obj.randomness = _(this.randomness).map(function(r) {
382             return r.toJSONObject();
383         });
384     }
385
386     return return_obj;
387 });
388
389
390
391 //Edit for preferential
392 HELIOS.EncryptedAnswer2 = Class.extend({
393     init: function(question, answer, answer_val, pk, progress) {
394         // if nothing in the constructor
395         if (question == null)
396             return;
397
398         // store answer
399         // CHANGE 2008-08-06: answer is now an *array* of answers, not
400             just a single integer
401         this.answer = answer;
402         this.answer_val = answer_val;
403         // do the encryption
404         var enc_result = this.doEncryption(question, answer, answer_val,
405             pk, null, progress);
406         this.choices = enc_result.choices;
407         this.randomness = enc_result.randomness;
408         this.individual_proofs = enc_result.individual_proofs;

```

```

407     this.overall_proof = enc_result.overall_proof;
408 },
409
410 doEncryption: function(question, answer, answer_val, pk,
411     randomness, progress) {
412     var choices = [];
413     var individual_proofs = [];
414     var overall_proof = null;
415
416     // possible plaintexts [question.min .. , question.max]
417     var plaintexts = null;
418     var jj=0;
419     //jj= Math.round(1.5*question.answers.length +3);
420     jj=question.answers.length+1;
421     if (question.max != null) {
422         plaintexts = UTILS.generate_plaintexts(pk, 0, jj);
423     }
424     var zero_one_plaintexts = UTILS.generate_plaintexts(pk, 0,
425         question.max -1 );//1-> question.max (because of the
426         biggest value)
427
428     // keep track of whether we need to generate new randomness
429     var generate_new_randomness = false;
430     if (!randomness) {
431         randomness = [];
432         random_randomness = [];
433         random_alpha = [];
434         random_lambda = [];
435         permutation= [];
436         generate_new_randomness = true;
437     }
438
439     // keep track of number of options selected.
440     var num_selected_answers = question.max;
441
442     // go through each possible answer and encrypt either a g^0 or
443     // a g^1.
444     for (var i=0; i<question.answers.length; i++) {
445         var index, plaintext_index;
446
447         // generate randomness?
448         //-----//
449         if (generate_new_randomness) {
450             randomness[i] = Random.getRandomInteger(pk.q);
451         }
452         if (generate_new_randomness) {
453             random_alpha[i]=Random.getRandomInteger(pk.q);
454         }
455         if (generate_new_randomness) {
456             random_lambda[i]=Random.getRandomInteger(pk.q);
457         }
458
459         plaintext_index = answer_val[i];// we put the right value
460         permutation[i] = BigInt.fromInt(plaintext_index);//either this
461             or we put the right permutation index
462         random_randomness[i]=randomness[i];//finally we will add the
463             initial x_i

```



```

458     choices[i] = ElGamal.encrypt(pk, zero_one_plaintexts[
        plaintext_index], randomness[i]); //change it!!
459 }
460 if (generate_new_randomness) {
461     var random_sigma=Random.getRandomInteger(pk.q);
462 }
463 if (generate_new_randomness) {
464     var random_rho=Random.getRandomInteger(pk.q);
465 }
466 if (generate_new_randomness) {
467     var random_tau=Random.getRandomInteger(pk.q);
468 }
469 if (generate_new_randomness) {
470     var random_alpha_sum=Random.getRandomInteger(pk.q);
471 }
472 if (generate_new_randomness) {
473     var random_lambda_sum=Random.getRandomInteger(pk.q);
474 }
475 }
476
477
478 overall_proof=choices[0].generateshuffleproof(plaintexts, 0,
    randomness[0], ElGamal.disjunctive_challenge_generator,
    random_tau,random_rho,random_sigma,random_lambda_sum,
    random_lambda,random_alpha_sum,random_alpha,permutation,
    random_randomness,choices);
479 return {
480     'choices' : choices,
481     'randomness' : randomness,
482     'individual_proofs' : individual_proofs,
483     'overall_proof' : overall_proof
484 };
485 },
486
487 clearPlaintexts: function() {
488     this.answer = null;
489     this.randomness = null;
490 },
491
492 // FIXME: should verifyEncryption really generate proofs?
    Overkill.
493 verifyEncryption: function(question, pk) {
494     var zero_one_plaintexts = UTILS.generate_plaintexts(pk, 0,
        question.max -1 );
495     var choices=[];
496     for (var i=0; i<question.max; i++) {
497         choices[i] = ElGamal.encrypt(pk, zero_one_plaintexts[this.
            answer[i]], this.randomness[i]);
498     }
499     //var result = this.doEncryption(question, this.answer, this.
        answer, pk, this.randomness);
500     var flag1=false;
501     var flag2=true;
502     for (var i=0; i<question.max; i++) {
503         flag1=true;
504         for (var j=0; j<question.max; j++) {

```

```

505     choices[j] = ElGamal.encrypt(pk, zero_one_plaintexts[this.
        answer[j]], this.randomness[i]);
506     if (choices[j].equals(this.choices[i])) {
507         // alert ("oy: " + result.choices[i] + "/" + this.choices[i])
        ;
508         flag1=false;
509     }
510 }
511 if (flag1){
512     flag2=false;
513 }
514 }
515 return flag2;
516 },
517
518 toString: function() {
519     // get each ciphertext as a JSON string
520     var choices_strings = _(this.choices).map(function(c) {return c
        .toString();});
521     return choices_strings.join("|");
522 },
523
524 toJSONObject: function(include_plaintext) {
525     var return_obj = {
526         'choices' : _(this.choices).map(function(choice) {
527             return choice.toJSONObject();
528         }),
529         'individual_proofs' : _(this.individual_proofs).map(function(
        disj_proof) {
530             return disj_proof.toJSONObject();
531         })
532     };
533
534     if (this.overall_proof != null) {
535         return_obj.overall_proof = this.overall_proof.toJSONObject();
536     } else {
537         return_obj.overall_proof = null;
538     }
539
540     if (include_plaintext) {
541         return_obj.answer = this.answer;
542         return_obj.randomness = _(this.randomness).map(function(r) {
543             return r.toJSONObject();
544         });
545     }
546
547     return return_obj;
548 }
549 });
550
551
552 HELIOS.EncryptedAnswer.fromJSONObject = function(d, election) {
553     var ea = new HELIOS.EncryptedAnswer();
554     ea.choices = _(d.choices).map(function(choice) {
555         return ElGamal.Ciphertext.fromJSONObject(choice, election.
        public_key);
556     });

```

```

557
558   ea.individual_proofs = _(d.individual_proofs).map(function (p) {
559     return ElGamal.DisjunctiveProof.fromJSONObject(p);
560   });
561
562   ea.overall_proof = ElGamal.DisjunctiveProof.fromJSONObject(d.
    overall_proof);
563
564   // possibly load randomness and plaintext
565   if (d.randomness) {
566     ea.randomness = _(d.randomness).map(function(r) {
567       return BigInt.fromJSONObject(r);
568     });
569     ea.answer = d.answer;
570   }
571
572   return ea;
573 };
574
575 HELIOS.EncryptedAnswer2.fromJSONObject = function(d, election) {
576   var ea = new HELIOS.EncryptedAnswer2();
577   ea.choices = _(d.choices).map(function(choice) {
578     return ElGamal.Ciphertext.fromJSONObject(choice, election.
    public_key);
579   });
580
581   ea.individual_proofs = _(d.individual_proofs).map(function (p) {
582     return ElGamal.DisjunctiveProof.fromJSONObject(p);
583   });
584
585   ea.overall_proof = ElGamal.DisjunctiveProof.fromJSONObject(d.
    overall_proof);
586
587   // possibly load randomness and plaintext
588   if (d.randomness) {
589     ea.randomness = _(d.randomness).map(function(r) {
590       return BigInt.fromJSONObject(r);
591     });
592     ea.answer = d.answer;
593     ea.answer_val=d.answer_val;
594   }
595
596   return ea;
597 };
598
599 HELIOS.EncryptedVote = Class.extend({
600   init: function(election, answers, progress) {
601     // empty constructor
602     if (election == null)
603       return;
604
605     // keep information about the election around
606     this.election_uuid = election.uuid;
607     this.election_hash = election.get_hash();
608     this.election = election;
609
610     if (answers == null)

```

```

611     return;
612
613     var n_questions = election.questions.length;
614     this.encrypted_answers = [];
615
616     if (progress) {
617         // set up the number of ticks
618         _(election.questions).each(function(q, q_num) {
619             // + 1 for the overall proof
620             progress.addTicks(q.answers.length);
621             if (q.max != null)
622                 progress.addTicks(q.max);
623         });
624
625         progress.addTicks(0, n_questions);
626     }
627
628     // loop through questions
629     for (var i=0; i<n_questions; i++) {
630         this.encrypted_answers[i] = new HELIOS.EncryptedAnswer(
631             election.questions[i], answers[i], election.public_key,
632             progress);
633     }
634
635     toString: function() {
636         // for each question, get the encrypted answer as a string
637         var answer_strings = _(this.encrypted_answers).map(function(a)
638             {return a.toString();});
639
640         return answer_strings.join("//");
641     },
642
643     clearPlaintexts: function() {
644         _(this.encrypted_answers).each(function(ea) {
645             ea.clearPlaintexts();
646         });
647     },
648
649     verifyEncryption: function(questions, pk) {
650         var overall_result = true;
651         _(this.encrypted_answers).each(function(ea, i) {
652             overall_result = overall_result && ea.verifyEncryption(
653                 questions[i], pk);
654         });
655         return overall_result;
656     },
657
658     toJSONObject: function(include_plaintext) {
659         var answers = _(this.encrypted_answers).map(function(ea,i) {
660             return ea.toJSONObject(include_plaintext);
661         });
662
663         return {
664             answers : answers,
665             election_hash : this.election_hash,
666             election_uuid : this.election_uuid

```

```

664     }
665   },
666
667   get_hash: function() {
668     return b64_sha256(JSON.stringify(this.toJSONObject()));
669   },
670
671   get_audit_trail: function() {
672     return this.toJSONObject(true);
673   },
674
675   verifyProofs: function(pk, outcome_callback) {
676     var zero_or_one = UTILS.generate_plaintexts(pk, 0, 1);
677
678     var VALID_P = true;
679
680     var self = this;
681
682     // for each question and associate encrypted answer
683     _(this.encrypted_answers).each(function(enc_answer, ea_num) {
684       var overall_result = 1;
685
686       // the max number of answers (decides whether this is
687       // approval or not and requires an overall proof)
688       var max = self.election.questions[ea_num].max;
689
690       // go through each individual proof
691       _(enc_answer.choices).each(function(choice, choice_num) {
692         var result = choice.verifyDisjunctiveProof(zero_or_one,
693           enc_answer.individual_proofs[choice_num], ElGamal.
694             disjunctive_challenge_generator);
695         outcome_callback(ea_num, choice_num, result, choice);
696
697         VALID_P = VALID_P && result;
698
699         // keep track of homomorphic product, if needed
700         if (max != null)
701           overall_result = choice.multiply(overall_result);
702       });
703
704       if (max != null) {
705         // possible plaintexts [0, 1, .. , question.max]
706         var plaintexts = UTILS.generate_plaintexts(pk, self.
707           election.questions[ea_num].min, self.election.
708             questions[ea_num].max);
709
710         // check the proof on the overall product
711         var overall_check = overall_result.verifyDisjunctiveProof
712           (plaintexts, enc_answer.overall_proof, ElGamal.
713             disjunctive_challenge_generator);
714         outcome_callback(ea_num, null, overall_check, null);
715         VALID_P = VALID_P && overall_check;
716       } else {
717         // check to make sure the overall_proof is null, since it
718         // 's approval voting
719         VALID_P = VALID_P && (enc_answer.overall_proof == null)
720       }
721     });

```

```

713     });
714
715     return VALID_P;
716   }
717 });
718
719 //edw new Helios.EncryptedVote for preferential
720 HELIOS.EncryptedVote2 = Class.extend({
721   init: function(election, answers, answersval, progress) {
722     // empty constructor
723     if (election == null)
724       return;
725
726     // keep information about the election around
727     this.election_uuid = election.uuid;
728     this.election_hash = election.get_hash();
729     this.election = election;
730
731     if (answers == null)
732       return;
733
734     var n_questions = election.questions.length;
735     this.encrypted_answers = [];
736
737     if (progress) {
738       // set up the number of ticks
739       _(election.questions).each(function(q, q_num) {
740         // + 1 for the overall proof
741         progress.addTicks(q.answers.length);
742         if (q.max != null)
743           progress.addTicks(q.max);
744       });
745
746       progress.addTicks(0, n_questions);
747     }
748
749     // loop through questions
750     for (var i=0; i<n_questions; i++) {
751       this.encrypted_answers[i] = new HELIOS.EncryptedAnswer2(
752         election.questions[i], answers[i], answersval[i],
753         election.public_key, progress);
754     }
755   },
756
757   toString: function() {
758     // for each question, get the encrypted answer as a string
759     var answer_strings = _(this.encrypted_answers).map(function(a)
760       {return a.toString();});
761
762     return answer_strings.join("//");
763   },
764
765   clearPlaintexts: function() {
766     _(this.encrypted_answers).each(function(ea) {
767       ea.clearPlaintexts();
768     });
769   },

```

```

767
768 verifyEncryption: function(questions, pk) {
769     var overall_result = true;
770     _(this.encrypted_answers).each(function(ea, i) {
771         overall_result = overall_result && ea.verifyEncryption(
            questions[i], pk);
772     });
773     return overall_result;//edw
774 },
775
776 toJSONObject: function(include_plaintext) {
777     var answers = _(this.encrypted_answers).map(function(ea,i) {
778         return ea.toJSONObject(include_plaintext);
779     });
780
781     return {
782         answers : answers,
783         election_hash : this.election_hash,
784         election_uuid : this.election_uuid
785     }
786 },
787
788 get_hash: function() {
789     return b64_sha256(JSON.stringify(this.toJSONObject()));
790 },
791
792 get_audit_trail: function() {
793     return this.toJSONObject(true);
794 },
795
796 verifyProofs: function(pk, outcome_callback) {
797     var zero_or_one = UTILS.generate_plaintexts(pk, 0, 1);
798
799     var VALID_P = true;
800     var self = this;
801
802     // for each question and associate encrypted answer
803     _(this.encrypted_answers).each(function(enc_answer, ea_num) {
804         var overall_result = 1;
805         var all_choices=[];
806         //var ii=1;
807         var max = self.election.questions[ea_num].max +1;
808         _(enc_answer.choices).each(function(choice, choice_num) {
809             //var result = choice.verifyDisjunctiveProof(zero_or_one,
            enc_answer.individual_proofs[choice_num], ElGamal.
            disjunctive_challenge_generator);
810             outcome_callback(ea_num, choice_num, true, choice);
811             VALID_P = VALID_P// && result;
812             all_choices[choice_num]=choice;
813             if (max != null)
814
815                 overall_result = choice.multiply(overall_result);
816         });
817
818         if (max != null) {
819             // possible plaintexts [0, 1, .. , question.max]
820             var plaintexts = UTILS.generate_plaintexts(pk, 0, max);

```

```

821
822     // check the proof on the overall product
823     var overall_check = overall_result.verifyShuffleProof(
        plaintexts, enc_answer.overall_proof, all_choices,
        ElGamal.disjunctive_challenge_generator);
824     outcome_callback(ea_num, null, overall_check, null);
825     VALID_P = VALID_P && overall_check;
826     }
827     });
828
829     return VALID_P;
830     }
831     });
832
833
834
835 HELIOS.EncryptedVote.fromJSONObject = function(d, election) {
836     if (d == null)
837         return null;
838
839     var ev = new HELIOS.EncryptedVote2(election); //EDW: change it
        back
840
841     ev.encrypted_answers = _(d.answers).map(function(ea) {
842         return HELIOS.EncryptedAnswer2.fromJSONObject(ea, election); //
            EDW!!! change it back
843     });
844
845     ev.election_hash = d.election_hash;
846     ev.election_uuid = d.election_uuid;
847
848     return ev;
849     });
850
851     // create an encrypted vote from a set of answers
852     HELIOS.EncryptedVote.fromEncryptedAnswers = function(election,
        enc_answers) {
853         var enc_vote = new HELIOS.EncryptedVote2(election, null); //EDW
            !!! change it back
854         enc_vote.encrypted_answers = [];
855         _(enc_answers).each(function(enc_answer, answer_num) {
856             enc_vote.encrypted_answers[answer_num] = enc_answer;
857         });
858         return enc_vote;
859     });
860
861     //
862     // Tally abstraction
863     //
864
865     HELIOS.Tally = Class.extend({
866         init: function(raw_tally, num_tallied) {
867             this.tally = raw_tally;
868             this.num_tallied = num_tallied;
869         },
870
871         toJSONObject: function() {

```



```

872     var tally_json_obj = _(this.tally).map(function(one_q) {
873         return _(one_q).map(function(one_a) {
874             return one_a.toJSONObject();
875         });
876     });
877
878     return {
879         num_tallied : this.num_tallied,
880         tally: tally_json_obj
881     };
882 }
883
884 });
885
886 HELIOS.Tally.fromJSONObject = function(d, public_key) {
887     var num_tallied = d['num_tallied'];
888
889     var raw_tally = _(d['tally']).map(function(one_q) {
890         return _(one_q).map(function(one_a) {
891             var new_val= ElGamal.Ciphertext.fromJSONObject(one_a,
892                 public_key);
893             return new_val;
894         });
895     });
896
897     return new HELIOS.Tally(raw_tally, num_tallied);
898 }
899 //
900 // distributed decryption : Trustees
901 //
902
903 // a utility function for jsonifying a list of lists of items
904 HELIOS.jsonify_list_of_lists = function lol) {
905     if (!lol)
906         return null;
907
908     return _(lol).map(function(sublist) {return _(sublist).map(
909         function(item) {return item.toJSONObject();}})});
910 }
911 // a utility function for doing the opposite with an item-level de-
912 // jsonifier
913 HELIOS.dejsonify_list_of_lists = function lol, item_dejsonifier) {
914     if (!lol)
915         return null;
916
917     return _(lol).map(function(sublist) {return _(sublist).map(
918         function(item) {return item_dejsonifier(item);}})});
919 }
920
921 HELIOS.Trustee = Class.extend({
922     init: function(uuid, public_key, public_key_hash, pok,
923         decryption_factors, decryption_proofs) {
924         this.uuid = uuid;
925         this.public_key = public_key;
926         this.public_key_hash = public_key_hash;

```

```

924     this.pok = pok;
925     this.decryption_factors = decryption_factors;
926     this.decryption_proofs = decryption_proofs;
927 },
928
929 toJSONObject: function() {
930     return {
931         'decryption_factors' : HELIOS.jsonify_list_of_lists(this.
            decryption_factors),
932         'decryption_proofs' : HELIOS.jsonify_list_of_list(this.
            decryption_proofs),
933         'email' : this.email, 'name' : this.name, 'pok' : this.pok.
            toJSONObject(), 'public_key' : this.public_key.
            toJSONObject()
934     };
935 }
936 });
937
938 HELIOS.Trustee.fromJSONObject = function(d) {
939     return new HELIOS.Trustee(d.uuid,
940         ElGamal.PublicKey.fromJSONObject(d.public_key), d.
            public_key_hash, ElGamal.DLogProof.fromJSONObject(d.pok),
941         HELIOS.dejsonify_list_of_lists(d.decryption_factors, BigInt.
            fromJSONObject),
942         HELIOS.dejsonify_list_of_lists(d.decryption_proofs, ElGamal.
            Proof.fromJSONObject));
943 };

```

Listing 5.1: helios.js

```

1
2 //
3 // inspired by George Danezis, rewritten by Ben Adida.
4 //
5
6 ElGamal = {};
7
8 ElGamal.Params = Class.extend({
9     init: function(p, q, g) {
10         this.p = p;
11         this.q = q;
12         this.g = g;
13     },
14
15     generate: function() {
16         // get the value x
17         var x = Random.getRandomInteger(this.q);
18         var y = this.g.modPow(x, this.p);
19         var pk = new ElGamal.PublicKey(this.p, this.q, this.g, y);
20         var sk = new ElGamal.SecretKey(x, pk);
21         return sk;
22     },
23
24     toJSONObject: function() {
25         return {g: this.g.toJSONObject(), p: this.p.toJSONObject(), q:
            this.q.toJSONObject()};
26     }
27 });

```

```

28
29 ElGamal.Params.fromJSONObject = function(d) {
30     var params = new ElGamal.Params();
31     params.p = BigInt.fromJSONObject(d.p);
32     params.q = BigInt.fromJSONObject(d.q);
33     params.g = BigInt.fromJSONObject(d.g);
34     return params;
35 };
36
37 ElGamal.PublicKey = Class.extend({
38     init : function(p,q,g,y) {
39         this.p = p;
40         this.q = q;
41         this.g = g;
42         this.y = y;
43     },
44
45     toJSONObject: function() {
46         return {g : this.g.toJSONObject(), p : this.p.toJSONObject(), q
47             : this.q.toJSONObject(), y : this.y.toJSONObject()};
48     },
49     verifyKnowledgeOfSecretKey: function(proof, challenge_generator)
50     {
51         // if challenge_generator is present, we have to check that the
52         // challenge was properly generated.
53         if (challenge_generator != null) {
54             if (!proof.challenge.equals(challenge_generator(proof.
55                 commitment))) {
56                 return false;
57             }
58         }
59
60         // verify that  $g^{\text{response}} = s * y^{\text{challenge}}$ 
61         var check = this.g.modPow(proof.response, this.p).equals(this.y
62             .modPow(proof.challenge, this.p).multiply(proof.commitment)
63             .mod(this.p));
64
65         return check;
66     },
67
68     // check if the decryption factor is correct for this public key,
69     // given the proof
70     verifyDecryptionFactor: function(ciphertext, decryption_factor,
71         decryption_proof, challenge_generator) {
72         return decryption_proof.verify(this.g, ciphertext.alpha, this.y
73             , decryption_factor, this.p, this.q, challenge_generator);
74     },
75
76     multiply: function(other) {
77         // base condition
78         if (other == 0 || other == 1) {
79             return this;
80         }
81
82         // check params
83         if (!this.p.equals(other.p))

```

```

76     throw "mismatched params";
77     if (!this.g.equals(other.g))
78         throw "mismatched params";
79
80     var new_pk = new ElGamal.PublicKey(this.p, this.q, this.g, this
81         .y.multiply(other.y).mod(this.p));
82     return new_pk;
83 },
84 equals: function(other) {
85     return (this.p.equals(other.p) && this.q.equals(other.q) &&
86         this.g.equals(other.g) && this.y.equals(other.y));
87 }
88 });
89
90 ElGamal.PublicKey.fromJSONObject = function(d) {
91     var pk = new ElGamal.PublicKey();
92     pk.p = BigInt.fromJSONObject(d.p);
93     pk.q = BigInt.fromJSONObject(d.q);
94     pk.g = BigInt.fromJSONObject(d.g);
95     pk.y = BigInt.fromJSONObject(d.y);
96     return pk;
97 };
98
99 ElGamal.SecretKey = Class.extend({
100     init: function(x, pk) {
101         this.x = x;
102         this.pk = pk;
103     },
104     toJSONObject: function() {
105         return {public_key: this.pk.toJSONObject(), x: this.x.
106             toJSONObject()};
107     },
108     // a decryption factor is *not yet* mod-inverted, because it
109     // needs to be part of the proof.
110     decryptionFactor: function(ciphertext) {
111         var decryption_factor = ciphertext.alpha.modPow(this.x, this.pk
112             .p);
113         return decryption_factor;
114     },
115     decrypt: function(ciphertext, decryption_factor) {
116         if (!decryption_factor)
117             decryption_factor = this.decryptionFactor(ciphertext);
118
119         // use the ciphertext's built-in decryption given a list of
120         // decryption factors.
121         return ciphertext.decrypt([decryption_factor]);
122     },
123     decryptAndProve: function(ciphertext, challenge_generator) {
124         var dec_factor_and_proof = this.decryptionFactorAndProof(
125             ciphertext, challenge_generator);

```

```

126 // decrypt, but using the already computed decryption factor
127 var plaintext = this.decrypt(ciphertext, dec_factor_and_proof.
    decryption_factor);
128
129 return {
130   'plaintext': plaintext,
131   'proof': dec_factor_and_proof.decryption_proof
132 };
133 },
134
135 decryptionFactorAndProof: function(ciphertext,
    challenge_generator) {
136   var decryption_factor = this.decryptionFactor(ciphertext);
137
138   // the DH tuple we need to prove, given the secret key x, is:
139   // g, alpha, y, beta/m
140   var proof = ElGamal.Proof.generate(this.pk.g, ciphertext.alpha,
    this.x, this.pk.p, this.pk.q, challenge_generator);
141
142   return {
143     'decryption_factor' : decryption_factor,
144     'decryption_proof' : proof
145   }
146 },
147
148 // generate a proof of knowledge of the secret exponent x
149 proveKnowledge: function(challenge_generator) {
150   // generate random w
151   var w = Random.getRandomInteger(this.pk.q);
152
153   // compute s = g^w for random w.
154   var s = this.pk.g.modPow(w, this.pk.p);
155
156   // get challenge
157   var challenge = challenge_generator(s);
158
159   // compute response = w + x * challenge
160   var response = w.add(this.x.multiply(challenge)).mod(this.pk.q)
    ;
161
162   return new ElGamal.DLogProof(s, challenge, response);
163 }
164 });
165
166 ElGamal.SecretKey.fromJSONObject = function(d) {
167   var sk = new ElGamal.SecretKey();
168   sk.pk = ElGamal.PublicKey.fromJSONObject(d.public_key);
169   sk.x = BigInt.fromJSONObject(d.x);
170   return sk;
171 }
172
173 ElGamal.Ciphertext = Class.extend({
174   init: function(alpha, beta, pk) {
175     this.alpha = alpha;
176     this.beta = beta;
177     this.pk = pk;
178   },

```

```

179
180 toString: function() {
181     return this.alpha.toString() + ',' + this.beta.toString();
182 },
183
184 toJSONObject: function() {
185     return {alpha: this.alpha.toJSONObject(), beta: this.beta.
186             toJSONObject()}
187 },
188
189 multiply: function(other) {
190     // special case if other is 1 to enable easy aggregate ops
191     if (other == 1)
192         return this;
193
194     // homomorphic multiply
195     return new ElGamal.Ciphertext(this.alpha.multiply(other.alpha).
196                                   mod(this.pk.p),
197                                   this.beta.multiply(other.beta).
198                                   mod(this.pk.p),
199                                   this.pk);
200 },
201
202 // a decryption method by decryption factors
203 decrypt: function(list_of_dec_factors) {
204     var running_decryption = this.beta;
205     var self = this;
206     _(list_of_dec_factors).each(function(dec_factor) {
207         running_decryption = dec_factor.modInverse(self.pk.p).
208             multiply(running_decryption).mod(self.pk.p);
209     });
210     return new ElGamal.Plaintext(running_decryption, this.pk, false
211                                 );
212 },
213
214 generateProof: function(plaintext, randomness,
215                        challenge_generator) {
216     // DH tuple to prove is
217     // g, y, alpha, beta/m
218     // with dlog randomness
219     var proof = ElGamal.Proof.generate(this.pk.g, this.pk.y,
220                                       randomness, this.pk.p, this.pk.q, challenge_generator);
221
222     return proof;
223 },
224
225 simulateProof: function(plaintext, challenge) {
226     if (challenge == null) {
227         challenge = Random.getRandomInteger(this.pk.q);
228     }
229
230     challenge = BigInt.ONE;
231     var response = BigInt.ONE;
232     var A = BigInt.TWO.mod(this.pk.p);
233     var B = BigInt.ONE.mod(this.pk.p);
234     var C = BigInt.ZERO.mod(this.pk.p);

```

```

229 //EDW
230 return new ElGamal.Proof(A, B, C, challenge, response);
231 },
232
233 verifyProof: function(plaintext, proof, challenge_generator) {
234     // DH tuple to verify is
235     // g, y, alpha, beta/m
236     var beta_over_m = this.beta.multiply(plaintext.m.modInverse(
237         this.pk.p)).mod(this.pk.p);
238
239     return proof.verify(this.pk.g, this.pk.y, this.alpha,
240         beta_over_m, this.pk.p, this.pk.q, challenge_generator);
241 },
242
243 verifyDecryptionProof: function(plaintext, proof,
244     challenge_generator) {
245     // DH tuple to verify is
246     // g, alpha, y, beta/m
247     // since the proven dlog is the secret key x, y=g^x.
248     var beta_over_m = this.beta.multiply(plaintext.m.modInverse(
249         this.pk.p)).mod(this.pk.p);
250
251     return proof.verify(this.pk.g, this.alpha, this.pk.y,
252         beta_over_m, this.pk.p, this.pk.q, challenge_generator);
253 },
254
255 generateshuffleproof: function(list_of_plaintexts, real_index,
256     randomness, challenge_generator, random_tau, random_rho,
257     random_sigma, random_lambda_sum, random_lambda,
258     random_alpha_sum, random_alpha, permutation, random_randomness,
259     choices){
260     var self = this;
261     var ii=BigInt.ZERO;
262     var p_1= self.pk.p; //p
263     var p_2= p_1.multiply(p_1);//p^2
264     var p_3= p_2.multiply(p_1);//p^3
265     var t=BigInt.ONE;
266     var v=BigInt.ONE;
267     var w=BigInt.ONE;
268     var u=BigInt.ONE;
269     var dot_random_g= BigInt.ONE;
270     var dot_m=BigInt.ONE;
271     var dot_g=BigInt.ONE;
272     var sa=BigInt.ONE;
273     var dot_w=BigInt.ONE;
274     var dot_u=BigInt.ONE;
275     var uu =BigInt.ONE;
276     var t1=BigInt.ONE;
277     var ra=BigInt.ONE;
278     var t1_plus_ra=BigInt.ONE;
279     var counter1=BigInt.ONE;
280     var counter2=BigInt.ONE;
281     var counter3=BigInt.ONE;
282     var counter4=BigInt.ONE;
283     var a_j_square=BigInt.ONE;
284     var a_j_square_A=BigInt.ONE;
285     var a_j_cube=BigInt.ONE;

```

```

277 var exp_dot_w_i=BigInt.ONE;
278 var exp_dot_t_i=BigInt.ONE;
279 var exp_dot_u_i=BigInt.ONE;
280 var a_j_cube=BigInt.ONE;
281 var s=BigInt.ONE;
282 var s2=BigInt.ONE;//s2
283 var check= BigInt.ONE;
284 var u_i = [];
285 var dot_w_i = [];
286 var dot_u_i = [];
287 var dot_t_i = [];
288 var chall=[];
289 var s_i=[];
290     var dot_w_i_new=[];
291     var u_i_new=[];
292     var dot_t_i_new=[];
293     var dot_u_i_new=[];
294     var counter1_new=BigInt.ONE;
295     var a_j_square_new_A_new=BigInt.ONE;
296     var counter3_new=BigInt.ONE;
297     var temp_w_i_new=BigInt.ONE;
298     var exp_dot_w_i_new=BigInt.ONE;
299     var exp_dot_t_i_new=BigInt.ONE;
300     var exp_dot_u_i_new=BigInt.ONE;
301     var tempa_new=BigInt.ONE;
302     var A_new=BigInt.ONE;
303     var temp1_new=BigInt.ONE;
304     var temp2_new=BigInt.ONE;
305     var temp3_new=BigInt.ONE;
306     var B_new=BigInt.ONE;
307     var ii2=BigInt.ONE;
308     var uu2=BigInt.ONE;
309     var cA_ij=BigInt.ONE;
310
311
312
313 var proofs = _(list_of_plaintexts).map(function(plaintext,
    p_num) {
314     if (p_num == real_index) {
315         // no real proof yet
316
317         return {};
318     } else {
319         var A = BigInt.ONE;
320         var B = BigInt.ONE;
321         var C = BigInt.ONE;
322         var challenge = BigInt.ONE;
323         var response = BigInt.ONE;
324         var jjj=list_of_plaintexts.length-1;
325         var jj=BigInt.fromInt(jjj);
326
327         if(ii.equals(BigInt.ZERO)){
328             t=self.pk.g.modPow(random_tau, self.pk.p);
329             v=self.pk.g.modPow(random_rho, self.pk.p);
330             w=self.pk.g.modPow(random_sigma, self.pk.p);
331             u=self.pk.g.modPow(random_lambda_sum, self.pk.p);
332             dot_random_g= BigInt.ONE;

```



```

333 dot_m=self.pk.y.modPow(random_alpha_sum, self.pk.p);//m'=y^
    a
334 dot_g=self.pk.g.modPow(random_alpha_sum, self.pk.p);//g'=g^
    a
335 sa=random_sigma.multiply(random_alpha_sum);
336 dot_w=self.pk.g.modPow(sa, self.pk.p);//w'=g^{sa}
337 tl=random_tau.multiply(random_lambda_sum);
338 ra=random_alpha_sum.multiply(random_rho);
339 tl_plus_ra=tl.add(ra);
340 dot_u=self.pk.g.modPow(tl_plus_ra, self.pk.p);//u'=g^{tau*1
    +rho*a}
341 s=random_alpha_sum;
342 s2=random_lambda_sum;//s2
343 check=s2;
344
345 var ciphertext_i=choices[ii];
346 b_aj=ciphertext_i.beta.modPow(random_alpha[ii], self.pk.p);
347 dot_m=dot_m.multiply(b_aj).mod(self.pk.p); //m'=m'*beta_j
    ^{a_j}
348 a_aj=ciphertext_i.alpha.modPow(random_alpha[ii], self.pk.p)
    ;
349 dot_g=dot_g.multiply(a_aj).mod(self.pk.p); //g'=g'*alpha_j
    ^{a_j}
350 uu = permutation[ii]; //the right A_ji for next equations
351 counter1=self.pk.g.modPow(random_alpha[uu], self.pk.p); //g
    ^{a_j*A_ji}
352 a_j_square=random_alpha[ii].multiply(random_alpha[ii]); //
    a_j^2
353 counter2=self.pk.g.modPow(a_j_square, self.pk.p); //g^{a_j
    ^2}
354 a_j_square_A=random_alpha[uu].multiply(random_alpha[uu]);
    //(a_j^2)*(A_ji)
355 counter3=self.pk.g.modPow(a_j_square_A, self.pk.p); //g^{(
    a_j^2)*(A_ji)}
356 a_j_cube=a_j_square.multiply(random_alpha[ii]); //a_j^3
357 counter4=self.pk.g.modPow(a_j_cube, self.pk.p);//g^{a_j^3}
358 dot_u=dot_u.multiply(counter4).mod(self.pk.p);//u'=u'*g^{
    a_j^3}
359 var temp_w_i=random_sigma.multiply(random_randomness[ii]);
360 exp_dot_w_i=self.pk.g.modPow(temp_w_i, self.pk.p);//g^{s*
    r_i}
361 dot_w_i[ii]= exp_dot_w_i.multiply(counter1).mod(self.pk.p);
    //g^{s*r_i}*g^{a_j*A_ji}
362 dot_w_i[ii]= dot_w_i[ii].multiply(counter1).mod(self.pk.p);
    //g^{s*r_i}*g^{a_j*A_ji}*g^{a_j*A_ji}
363 //check3=check3.divide(check2);
364 dot_w= dot_w.multiply(counter2).mod(self.pk.p);//w'=w'*g^{
    a_j^2}
365 exp_dot_t_i=self.pk.g.modPow(random_tau.multiply(
    random_lambda[ii]), self.pk.p);//g^{tau*1_i}
366 dot_t_i[ii]=exp_dot_t_i.multiply(counter1).multiply(
    counter1).multiply(counter1).mod(self.pk.p);//g^{tau*
    1_i}*g^{a_j*A_ji}
367 //dot_t_i[ii]= dot_t_i[ii].multiply(counter1).mod(self.pk.p
    );//g^{tau*1_i}*g^{a_j*A_ji}*g^{a_j*A_ji}
368 // dot_t_i[ii]= dot_t_i[ii].multiply(counter1).mod(self.pk.p
    );//g^{s*r_i}*g^{a_j*A_ji}*g^{a_j*A_ji}*g^{a_j*A_ji}

```

```

369     exp_dot_u_i=self.pk.g.modPow(random_rho.multiply(
370         random_randomness[ii]), self.pk.p);//g^{rho*r_i}
371     dot_u_i[ii]=exp_dot_u_i.multiply(counter3).multiply(
372         counter3).multiply(counter3).mod(self.pk.p);//g^{rho*
373         r_i}*g^{(a_j^2)*(A_ji)}*g^{(a_j^2)*(A_ji)}*g^{(a_j^2)*(
374         A_ji)}
375     u_i[ii]=self.pk.g.modPow(random_lambda[ii],self.pk.p);
376     s_i[ii]=random_alpha[ii];
377     //A=u_i+p_1*dot_w_i (+ p_2*random_g_i(when produced))
378     //B=dot_t_i+p_1*dot_u_i
379     //challenge=hash(A,B)
380     //response=s_i
381     var tempa=p_1.multiply(dot_w_i[ii]);
382     A=u_i[ii].add(tempa);//plus p_2*random_g_i(when produced)
383
384     var temp1=dot_u_i[ii].mod(self.pk.p);
385     var temp2=dot_t_i[ii].mod(self.pk.p);
386     var temp3=p_1.multiply(temp1);
387     B=temp2.add(temp3);
388     var strings_to_hash = [];
389     strings_to_hash[strings_to_hash.length] = A.toString();
390     strings_to_hash[strings_to_hash.length] = B.toString();
391     challenge= new BigInt(hex_shal(strings_to_hash.join(","),
392         16));
393     chall[ii]=challenge;
394     s=s.add(random_randomness[ii].multiply(challenge));//s=s+
395         r_j*c_j
396     s2=s2.add(random_lambda[ii].multiply(challenge).multiply(
397         challenge));//s2=s2+ r_j*c_j
398
399     //we have to produce the Sum(challenge*A_ij) = challenge(u[
400         ii])
401     ii2=uu;
402     uu2=permutation[ii2];
403     counter1_new=self.pk.g.modPow(random_alpha[uu2], self.pk
404         .p); //g^{a_j*A_ji}
405     a_j_square_new_A_new=random_alpha[uu2].multiply(
406         random_alpha[uu2]); //(a_j^2)*(A_ji)
407     counter3_new=self.pk.g.modPow(a_j_square_new_A_new, self
408         .pk.p); //g^{(a_j^2)*(A_ji)}
409     temp_w_i_new=random_sigma.multiply(random_randomness[ii2
410         ]);
411     exp_dot_w_i_new=self.pk.g.modPow(temp_w_i_new, self.pk.p
412         );//g^{s*r_i}
413     dot_w_i_new[ii2]= exp_dot_w_i_new.multiply(counter1_new)
414         .mod(self.pk.p);//g^{s*r_i}*g^{a_j*A_ji}
415     dot_w_i_new[ii2]= dot_w_i_new[ii2].multiply(counter1_new
416         ).mod(self.pk.p);//g^{s*r_i}*g^{a_j*A_ji}*g^{a_j*
417         A_ji}
418     exp_dot_t_i_new=self.pk.g.modPow(random_tau.multiply(
419         random_lambda[ii2]), self.pk.p);//g^{tau*l_i}
420     dot_t_i_new[ii2]=exp_dot_t_i_new.multiply(counter1_new).
421         multiply(counter1_new).multiply(counter1_new).mod(
422         self.pk.p);//g^{tau*l_i}*g^{a_j*A_ji}
423     exp_dot_u_i_new=self.pk.g.modPow(random_rho.multiply(
424         random_randomness[ii2]), self.pk.p);//g^{rho*r_i}
425     dot_u_i_new[ii2]=exp_dot_u_i_new.multiply(counter3_new).

```

```

        multiply(counter3_new).multiply(counter3_new).mod(
        self.pk.p); //g^{rho*r_i}*g^{(a_j^2)*(A_ji)}*g^{(a_j
        ^2)*(A_ji)}*g^{(a_j^2)*(A_ji)}
406     u_i_new[ii2]=self.pk.g.modPow(random_lambda[ii2],self.pk
        .p);
407     tempa_new=p_1.multiply(dot_w_i_new[ii2]);
408     A_new=u_i_new[ii2].add(tempa_new); //plus p_2*random_g_i(
        when produced)
409     temp1_new=dot_u_i_new[ii2].mod(self.pk.p);
410     temp2_new=dot_t_i_new[ii2].mod(self.pk.p);
411     temp3_new=p_1.multiply(temp1_new);
412     B_new=temp2_new.add(temp3_new);
413     var strings_to_hash_new = [];
414     strings_to_hash_new[strings_to_hash_new.length] = A_new.
        toString();
415     strings_to_hash_new[strings_to_hash_new.length] = B_new.
        toString();
416     cA_ij= new BigInt(hex_sha1(strings_to_hash_new.join(",")
        ), 16);
417     s_i[ii]=s_i[ii].add(cA_ij);
418     response=s_i[ii].mod(self.pk.q);
419
420 }else{
421     if (ii.equals(jj.add(BigInt.ONE.negate()))){
422         C=BigInt.ZERO;
423     }
424     else{
425         var ciphertext_i=choices[ii];
426         b_aj=ciphertext_i.beta.modPow(random_alpha[ii], self.pk.p
        );
427         dot_m=dot_m.multiply(b_aj).mod(self.pk.p); //m'=m'*
        beta_j^{a_j}
428         a_aj=ciphertext_i.alpha.modPow(random_alpha[ii], self.pk.
        p);
429         dot_g=dot_g.multiply(a_aj).mod(self.pk.p); //g'=g'*
        alpha_j^{a_j}
430         uu = permutation[ii]; //the right A_ji for next equations
431         counter1=self.pk.g.modPow(random_alpha[uu], self.pk.p);
        //g^{a_j*A_ji}
432         a_j_square=random_alpha[ii].multiply(random_alpha[ii]);
        //a_j^2
433         counter2=self.pk.g.modPow(a_j_square, self.pk.p); //g^{
        a_j^2}
434         a_j_square_A=random_alpha[uu].multiply(random_alpha[uu]);
        //(a_j^2)*(A_ji)
435         counter3=self.pk.g.modPow(a_j_square_A, self.pk.p); //g
        ^{(a_j^2)*(A_ji)}
436         a_j_cube=a_j_square.multiply(random_alpha[ii]); //a_j^3
437         counter4=self.pk.g.modPow(a_j_cube, self.pk.p); //g^{a_j
        ^3}
438         dot_u=dot_u.multiply(counter4).mod(self.pk.p); //u'=u'*g^{
        a_j^3}
439         var temp_w_i=random_sigma.multiply(random_randomness[ii])
        ;
440         exp_dot_w_i=self.pk.g.modPow(temp_w_i, self.pk.p); //g^{s*
        r_i}
441         dot_w_i[ii]= exp_dot_w_i.multiply(counter1).mod(self.pk.p

```

```

    );//g^{s*r_i}*g^{a_j*A_ji}
442 dot_w_i[ii]= dot_w_i[ii].multiply(counter1).mod(self.pk.p)
    );//g^{s*r_i}*g^{a_j*A_ji}*g^{a_j*A_ji}
443 dot_w= dot_w.multiply(counter2).mod(self.pk.p);//w'=w'*g
    ^{a_j^2}
444 exp_dot_t_i=self.pk.g.modPow(random_tau.multiply(
    random_lambda[ii]), self.pk.p);//g^{tau*1_i}
445 dot_t_i[ii]=exp_dot_t_i.multiply(counter1).multiply(
    counter1).multiply(counter1).mod(self.pk.p);//g^{tau*
    1_i}*g^{a_j*A_ji}
446 //dot_t_i[ii]= dot_t_i[ii].multiply(counter1).mod(self.pk
    .p);//g^{tau*1_i}*g^{a_j*A_ji}*g^{a_j*A_ji}
447 // dot_t_i[ii]= dot_t_i[ii].multiply(counter1).mod(self.pk
    .p);//g^{s*r_i}*g^{a_j*A_ji}*g^{a_j*A_ji}*g^{a_j*A_ji}
448 exp_dot_u_i=self.pk.g.modPow(random_rho.multiply(
    random_randomness[ii]), self.pk.p);//g^{rho*r_i}
449 dot_u_i[ii]=exp_dot_u_i.multiply(counter3).multiply(
    counter3).multiply(counter3).mod(self.pk.p);//g^{rho*
    r_i}*g^{(a_j^2)*(A_ji)}*g^{(a_j^2)*(A_ji)}*g^{(a_j^2)
    *(A_ji)}
450 u_i[ii]=self.pk.g.modPow(random_lambda[ii],self.pk.p);
451 s_i[ii]=random_alpha[ii];
452 //A=u_i+p_1*dot_w_i (+ p_2*random_g_i(when produced))
453 //B=dot_t_i+p_1*dot_u_i
454 //challenge=hash(A,B) where A,B is just
455 //response=s_i
456 var tempa=p_1.multiply(dot_w_i[ii]);
457 A=u_i[ii].add(tempa);//plus p_2*random_g_i(when produced)
458
459 var temp1=dot_u_i[ii].mod(self.pk.p);
460 var temp2=dot_t_i[ii].mod(self.pk.p);
461 var temp3=p_1.multiply(temp1);
462 B=temp2.add(temp3);
463 var strings_to_hash = [];
464 strings_to_hash[strings_to_hash.length] = A.toString();
465 strings_to_hash[strings_to_hash.length] = B.toString();
466 challenge= new BigInt(hex_sha1(strings_to_hash.join(",")
    , 16);
467 chall[ii]=challenge;
468
469 s=s.add(random_randomness[ii].multiply(challenge));//s=s+
    r_j*c_j
470 s2=s2.add(random_lambda[ii].multiply(challenge).multiply(
    challenge));//s2=s2+ r_j*c_j
471
472 //we have to produce the Sum(challenge*A_ij) = challenge(
    u[ii])
473 ii2=uu;
474 uu2=permutation[ii2];
475 counter1_new=self.pk.g.modPow(random_alpha[uu2], self.
    pk.p); //g^{a_j*A_ji}
476 a_j_square_new_A_new=random_alpha[uu2].multiply(
    random_alpha[uu2]); //(a_j^2)*(A_ji)
477 counter3_new=self.pk.g.modPow(a_j_square_new_A_new,
    self.pk.p); //g^{(a_j^2)*(A_ji)}
478 temp_w_i_new=random_sigma.multiply(random_randomness[
    ii2]);

```

```

479     exp_dot_w_i_new=self.pk.g.modPow(temp_w_i_new, self.pk
480     .p);//g^{s*r_i}
481     dot_w_i_new[ii2]= exp_dot_w_i_new.multiply(
482     counter1_new).mod(self.pk.p);//g^{s*r_i}*g^{a_j*
483     A_ji}
484     dot_w_i_new[ii2]= dot_w_i_new[ii2].multiply(
485     counter1_new).mod(self.pk.p);//g^{s*r_i}*g^{a_j*
486     A_ji}*g^{a_j*A_ji}
487     exp_dot_t_i_new=self.pk.g.modPow(random_tau.multiply(
488     random_lambda[ii2]), self.pk.p);//g^{tau*l_i}
489     dot_t_i_new[ii2]=exp_dot_t_i_new.multiply(counter1_new)
490     .multiply(counter1_new).multiply(counter1_new).mod(
491     self.pk.p);//g^{tau*l_i}*g^{a_j*A_ji}
492     exp_dot_u_i_new=self.pk.g.modPow(random_rho.multiply(
493     random_randomness[ii2]), self.pk.p);//g^{rho*r_i}
494     dot_u_i_new[ii2]=exp_dot_u_i_new.multiply(counter3_new)
495     .multiply(counter3_new).multiply(counter3_new).
496     mod(self.pk.p);//g^{rho*r_i}*g^{(a_j^2)*(A_ji)}*g
497     ^{(a_j^2)*(A_ji)}*g^{(a_j^2)*(A_ji)}
498     u_i_new[ii2]=self.pk.g.modPow(random_lambda[ii2],self.
499     pk.p);
500     tempa_new=p_1.multiply(dot_w_i_new[ii2]);
501     A_new=u_i_new[ii2].add(tempa_new);//plus p_2*
502     random_g_i(when produced)
503     temp1_new=dot_u_i_new[ii2].mod(self.pk.p);
504     temp2_new=dot_t_i_new[ii2].mod(self.pk.p);
505     temp3_new=p_1.multiply(temp1_new);
506     B_new=temp2_new.add(temp3_new);
507     var strings_to_hash_new = [];
508     strings_to_hash_new[strings_to_hash_new.length] = A_new
509     .toString();
510     strings_to_hash_new[strings_to_hash_new.length] = B_new
511     .toString();
512     cA_ij= new BigInt(hex_shal(strings_to_hash_new.join(",
513     ")), 16);
514
515     s_i[ii]=s_i[ii].add(cA_ij);
516     response=s_i[ii].mod(self.pk.q);
517
518 }
519 }
520
521 ii=ii.add(BigInt.ONE);
522
523 if (ii.equals(jj)){//we produce the final result of the
524     proofs.
525     //A=t+p_1*v+p_2*w+p_3*u
526     //B=g'+p_1*m'+p_2*dot_w+p_3*dot_u
527     //challenge=random_g' when produced
528     //response=s+s2*p_1
529     s2=s2.mod(self.pk.q);
530     s=s.mod(self.pk.q);
531     var A1=p_1.multiply(v);
532     var A2=p_2.multiply(w);
533     var A3=p_3.multiply(u);
534     A=t.add(A1);

```

```

518     A=A.add(A2);
519     A=A.add(A3);
520     var B1=p_1.multiply(dot_m);
521     var B2=p_2.multiply(dot_w);
522     var B3=p_3.multiply(dot_u);
523     B=dot_g.add(B1)
524     B=B.add(B2);
525     B=B.add(B3);
526     challenge=ii;//temporarily, will be changed later
527     response=s.add(s2.multiply(p_1));
528     return new ElGamal.Proof(A, B, C, challenge, response);
529
530 }
531 else{
532     //A=u_i+p_1*dot_w_i+p_2*w    (+ p_3*random_g_i(when
533     //B=dot_t_i+p_1*dot_u_i+p_2*dot_w+p_3*dot_u
534     //challenge=hash(A,B)
535     //response=s_i
536     //A=BigInt.ONE;
537     //B=cA_ij;
538     //response=BigInt.ZERO;
539     return new ElGamal.Proof(A, B, C, challenge, response);
540 }
541 }
542 });
543
544
545
546 // do the real proof
547 var real_proof = this.generateProof(list_of_plaintexts[
548     real_index], randomness, function(commitment) {
549     proofs[real_index] = {'commitment' : commitment};
550     var commitments = _(proofs).map(function(proof) {
551         return proof.commitment;
552     });
553     var disjunctive_challenge = challenge_generator(commitments);
554     var real_challenge = disjunctive_challenge;
555     _(proofs).each(function(proof, proof_num) {
556         if (proof_num != real_index)
557             real_challenge = real_challenge.add(proof.challenge.
558                 negate());
559     });
560     return real_challenge.mod(self.pk.q);
561 });
562 proofs[real_index] = real_proof;
563 return new ElGamal.DisjunctiveProof(proofs);
564 },
565 generateDisjunctiveProof: function(list_of_plaintexts, real_index
566     , randomness, challenge_generator) {
567     // go through all plaintexts and simulate the ones that must be
568     // simulated.
569     // note how the interface is as such so that the result does
570     // not reveal which is the real proof.
571     var self = this;

```

```

569     var proofs = _(list_of_plaintexts).map(function(plaintext,
570         p_num) {
571         if (p_num == real_index) {
572             // no real proof yet
573             return {};
574         } else {
575             // simulate!
576             return self.simulateProof(plaintext);
577         }
578     });
579
580
581     // do the real proof
582     var real_proof = this.generateProof(list_of_plaintexts[
583         real_index], randomness, function(commitment) {
584         // now we generate the challenge for the real proof by first
585         // determining
586         // the challenge for the whole disjunctive proof.
587
588         // set up the partial real proof so we're ready to get the
589         // hash;
590         proofs[real_index] = {'commitment' : commitment};
591
592         // get the commitments in a list and generate the whole
593         // disjunctive challenge
594         var commitments = _(proofs).map(function(proof) {
595             return proof.commitment;
596         });
597
598         var disjunctive_challenge = challenge_generator(commitments);
599
600         // now we must subtract all of the other challenges from this
601         // challenge.
602         var real_challenge = disjunctive_challenge;
603         _(proofs).each(function(proof, proof_num) {
604             if (proof_num != real_index)
605                 real_challenge = real_challenge.add(proof.challenge.
606                 negate());
607         });
608
609         // make sure we mod q, the exponent modulus
610         return real_challenge.mod(self.pk.q);
611     });
612     //var real_proof = self.simulateProof(plaintext);
613     // set the real proof
614     proofs[real_index] = real_proof;
615     return new ElGamal.DisjunctiveProof(proofs);
616 },
617 //edw: change it back to normal
618 verifyDisjunctiveProof: function(list_of_plaintexts, disj_proof,
619     challenge_generator) {
620     var result = true;
621     var proofs = disj_proof.proofs;
622
623     // for loop because we want to bail out of the inner loop
624     // if we fail one of the verifications.

```

```

618
619     for (var i=0; i < list_of_plaintexts.length; i++) {
620         if (!this.verifyProof(list_of_plaintexts[i], proofs[i]))
621             return false; //edw
622     }
623
624     // check the overall challenge
625
626     // first the one expected from the proofs
627     var commitments = _(proofs).map(function(proof) {return proof.
        commitment;});
628     var expected_challenge = challenge_generator(commitments);
629
630     // then the one that is the sum of the previous one.
631     var sum = new BigInt("0", 10); var self = this;
632     _(proofs).each(function(proof) {sum = sum.add(proof.challenge).
        mod(self.pk.q);});
633
634     return expected_challenge.equals(sum);
635 },
636
637 //Edw: new shuffle proof
638 verifyShuffleProof: function(list_of_plaintexts, disj_proof,
        all_choices, challenge_generator) {
639     var result = true;
640     var proofs = disj_proof.proofs;
641
642     // for loop because we want to bail out of the inner loop
643     // if we fail one of the verifications.
644     var self=this;
645     var requested_challenge=BigInt.ONE;
646     var A=BigInt.ONE;
647     var B=BigInt.ONE;
648     var res=BigInt.ONE;
649
650     var dot_w_i=BigInt.ONE;
651     var u_i=BigInt.ONE;
652     var dot_t_i=BigInt.ONE;
653     var dot_u_i=BigInt.ONE;
654     var alpha=BigInt.ONE;
655     var beta=BigInt.ONE;
656
657     var prod_1a=BigInt.ONE;
658     var prod_1b=BigInt.ONE;
659     var prod_3a=BigInt.ONE;
660     var prod_3b=BigInt.ONE;
661     var sum_4a=BigInt.ZERO;
662     var prod_4a=BigInt.ONE;
663     var prod_4b=BigInt.ONE;
664     var prod_5a=BigInt.ONE;
665     var prod_5b=BigInt.ONE;
666     var sum_6a=BigInt.ZERO;
667     var prod_6a=BigInt.ONE;
668     var prod_6b=BigInt.ONE;
669     var p_1= self.pk.p; //p
670     var p_2= p_1.multiply(p_1);//p^2
671     var p_3= p_2.multiply(p_1);//p^3

```



```

672 var i=BigInt.ONE;
673 for (var ii=1; ii < list_of_plaintexts.length-1; ii++) {
674     A=proofs[i].commitment.A;
675     B=proofs[i].commitment.B;
676     res=proofs[i].response;
677     var strings_to_hash = [];
678     strings_to_hash[strings_to_hash.length] = A.toString();
679     strings_to_hash[strings_to_hash.length] = B.toString();
680     requested_challenge= new BigInt(hex_sha1(strings_to_hash.join("
681         ,")), 16);
682     if (requested_challenge.equals(proofs[i].challenge)){
683         result=true;
684     }else{
685         return false; //edw
686     }
687     //A=u_i+p_1*dot_w_i (+ p_2*random_g_i(when produced))
688     //B=dot_t_i+p_1*dot_u_i
689     //challenge=hash(A,B) where A,B is just
690     //response=s_i
691     dot_w_i=A.divide(p_1);
692     u_i=A.mod(p_1);
693     dot_u_i=B.divide(p_1);
694     dot_t_i=B.mod(p_1);
695     alpha=all_choices[i-1].alpha;
696     beta=all_choices[i-1].beta;
697
698     //var temp1=alpha.modPow(res, p_1); //later
699     //prod_1a=prod_1a.multiply(temp1).pow(p_1);
700     var temp2=alpha.modPow(requested_challenge, p_1);
701     prod_1b=prod_1b.multiply(temp2).pow(p_1);
702
703     //var temp3=beta.modPow(res, p_1); //later
704     //prod_3a=prod_3a.multiply(temp3).pow(p_1);
705     var temp4=beta.modPow(requested_challenge, p_1);
706     prod_3b=prod_3b.multiply(temp4).pow(p_1);
707
708     var temp4a=res.multiply(res);
709     var temp4b=requested_challenge.multiply(requested_challenge);
710     var temp4c=temp4b.negate();
711     sum_4a=temp4a;
712     sum_4a=sum_4a.add(temp4c);
713     var temp7=self.pk.g.modPow(sum_4a, p_1);
714     prod_4a=prod_4a.multiply(temp7).mod(p_1);
715     var temp4d=dot_w_i.modPow(requested_challenge, p_1);
716     prod_4b=prod_4b.multiply(temp4d).mod(p_1);
717
718     var temp5=requested_challenge.multiply(requested_challenge);
719     var temp6=u_i.modPow(temp5, p_1);
720     prod_5b=prod_5b.multiply(temp6).mod(p_1);
721
722     var temp8=temp4c.multiply(requested_challenge);
723     var temp9=res.multiply(temp4a);
724     sum_6a=temp9;
725     sum_6a=sum_6a.add(temp8);
726     var temp10=self.pk.g.modPow(sum_6a, p_1);
727     prod_6a=prod_6a.multiply(temp10).mod(p_1);
728     var temp11=dot_u_i.modPow(requested_challenge, p_1);

```

```

728     var temp12=dot_t_i.modPow(temp5, p_1);
729     var prod_6b=prod_6b.multiply(temp11).mod(p_1);
730     var prod_6b=prod_6b.multiply(temp12).mod(p_1);
731
732
733     i=i.add(BigInt.ONE);
734 }
735 //A=t+p_1*v+p_2*w+p_3*u
736 //B=g'+p_1*m'+p_2*dot_w+p_3*dot_u
737 //challenge=random_g' when produced
738 //response=s+s2*p_1
739 A=proofs[i].commitment.A;
740 B=proofs[i].commitment.B;
741
742 res=proofs[i].response;
743 var u=A.divide(p_3);
744 A=A.mod(p_3);
745 var w=A.divide(p_2);
746     A=A.mod(p_2);
747 var v=A.divide(p_1);
748 A=A.mod(p_1);
749 var t=A;
750
751 var dot_u=B.divide(p_3);
752 B=B.mod(p_3);
753 var dot_w=B.divide(p_2);
754     B=B.mod(p_2);
755 var dot_m=B.divide(p_1);
756 B=B.mod(p_1);
757 var dot_g=B;
758
759 var dot_l=res.divide(p_1).mod(self.pk.q);
760 var s=res.mod(p_1);
761
762 prod_1a=prod_1a.multiply(self.pk.g.modPow(s,p_1)).pow(p_1);
763 prod_1b=prod_1b.multiply(dot_g).pow(p_1);
764 if (prod_1a.equals(prod_1b)){
765     result=true;
766 }else{
767     return false;
768 }
769
770 prod_3a=prod_3a.multiply(self.pk.y.modPow(s,p_1)).pow(p_1);
771 prod_3b=prod_3b.multiply(dot_m).pow(p_1);
772 if (prod_3a.equals(prod_3b)){
773     result=true;
774 }else{
775     return false;
776 }
777
778 var temp4e=w.modPow(s, p_1);
779 prod_4a=prod_4a.multiply(temp4e).mod(p_1);
780 prod_4b=prod_4b.multiply(dot_w).mod(p_1);
781
782 if (prod_4a.equals(prod_4b)){
783     result=true;
784 }else{

```

```

785     return false;
786 }
787
788 var temp_a=self.pk.g;
789 prod_5a=temp_a.modPow(dot_l, p_1);
790 prod_5b=prod_5b.multiply(u).mod(p_1);
791 if (prod_5a.equals(prod_5b)){
792     result=true;
793 }else{
794     return false;
795 }
796
797 var temp13=v.modPow(s, p_1);
798 var temp14=t.modPow(dot_l, p_1);
799 prod_6a=prod_6a.multiply(temp13).mod(p_1);
800 prod_6a=prod_6a.multiply(temp14).mod(p_1);
801 prod_6b=prod_6b.multiply(dot_u).mod(p_1);
802 if (prod_6a.equals(prod_6b)){
803     result=true;
804 }else{
805     return false; //edw
806 }
807
808 return true;
809 },
810
811 equals: function(other) {
812     return (this.alpha.equals(other.alpha) && this.beta.equals(
813         other.beta));
814 });
815
816 ElGamal.Ciphertext.fromJSONObject = function(d, pk) {
817     return new ElGamal.Ciphertext(BigInt.fromJSONObject(d.alpha),
818         BigInt.fromJSONObject(d.beta), pk);
819 };
820 // we need the public key to figure out how to encode m
821 ElGamal.Plaintext = Class.extend({
822     init: function(m, pk, encode_m) {
823         if (m == null) {
824             alert('oy null m');
825             return;
826         }
827
828         this.pk = pk;
829
830         if (encode_m) {
831             // need to encode the message given that p = 2q+1
832             var y = m.add(BigInt.ONE);
833             var test = y.modPow(pk.q, pk.p);
834             if (test.equals(BigInt.ONE)) {
835                 this.m = y;
836             } else {
837                 this.m = y.negate().mod(pk.p);
838             }
839         } else {

```

```

840     this.m = m;
841   }
842 },
843
844 getPlaintext: function() {
845   var y;
846
847   // if m < q
848   if (this.m.compareTo(this.pk.q) < 0) {
849     y = this.m;
850   } else {
851     y = this.m.negate().mod(this.pk.p);
852   }
853
854   return y.subtract(BigInt.ONE);
855 },
856
857 getM: function() {
858   return this.m;
859 }
860
861 });
862
863 //
864 // Proof abstraction
865 //
866
867 ElGamal.Proof = Class.extend({
868   init: function(A, B, C, challenge, response) { //edw: change it
869     back, if you need the old system
870     this.commitment = {};
871     this.commitment.A = A;
872     this.commitment.B = B;
873     this.commitment.C = C;
874     this.challenge = challenge;
875     this.response = response;
876   },
877   toJSONObject: function() {
878     return {
879       challenge : this.challenge.toJSONObject(),
880       commitment : {A: this.commitment.A.toJSONObject(), B: this.
881         commitment.B.toJSONObject() }, //Edw
882       response : this.response.toJSONObject()
883     }
884   },
885   // verify a DH tuple proof
886   verify: function(little_g, little_h, big_g, big_h, p, q,
887     challenge_generator) {
888     // check that little_g^response = A * big_g^challenge
889     var first_check = little_g.modPow(this.response, p).equals(
890       big_g.modPow(this.challenge, p).multiply(this.commitment.A)
891       .mod(p));
892
893     // check that little_h^response = B * big_h^challenge
894     var second_check = little_h.modPow(this.response, p).equals(

```

```

        big_h.modPow(this.challenge, p).multiply(this.commitment.B)
        .mod(p));
892
893     var third_check = true;
894
895     if (challenge_generator) {
896         third_check = this.challenge.equals(challenge_generator(this.
            commitment));
897     }
898
899     return (first_check && second_check && third_check);
900 //return (true);
901 },
902
903 verify2: function(little_g, little_h, big_g, big_h, p, q,
    challenge_generator) {
904     // check that little_g^response = A * big_g^challenge
905     var first_check = little_g.modPow(this.response, p).equals(
        big_g.modPow(this.challenge, p).multiply(this.commitment.A)
        .mod(p));
906
907     // check that little_h^response = B * big_h^challenge
908     var second_check = little_h.modPow(this.response, p).equals(
        big_h.modPow(this.challenge, p).multiply(this.commitment.B)
        .mod(p));
909
910     var third_check = true;
911
912     if (challenge_generator) {
913         third_check = this.challenge.equals(challenge_generator(this.
            commitment));
914     }
915
916     return (first_check && second_check && third_check);
917 //return (true);
918 }
919 });
920
921
922 //edw: insert the new proof text
923 ElGamal.Proof2 = Class.extend({
924     init: function(t, v, w, u, ui, dot_m, dot_ti, dot_vi, dot_v,
        dot_wi, dot_w, challenge, response, si, dot_l) {
925         this.commitment = {};
926         this.commitment.t = t;
927         this.commitment.v = v;
928         this.commitment.w = w;
929         this.commitment.u = u;
930         this.commitment.ui = ui;
931         this.commitment.dot_m = dot_m;
932         this.commitment.dot_ti = dot_ti;
933         this.commitment.dot_vi = dot_vi;
934         this.commitment.dot_v = dot_v;
935         this.commitment.dot_wi = dot_wi;
936         this.commitment.dot_w = dot_w;
937         this.challenge = challenge;
938         //this.s = s;

```

```

939     this.response = response;
940     this.si = si;
941     this.dot_l = dot_l;
942 },
943
944 toJSONObject: function() {
945     return {
946         challenge : this.challenge.toJSONObject(),
947         commitment : {t: this.commitment.t.toJSONObject(), v: this.
            commitment.v.toJSONObject(), w: this.commitment.w.
            toJSONObject(), u: this.commitment.u.toJSONObject(), ui:
            this.commitment.ui.toJSONObject(), dot_m: this.commitment
            .dot_m.toJSONObject(), dot_ti: this.commitment.dot_ti.
            toJSONObject(), dot_vi: this.commitment.dot_vi.
            toJSONObject(), dot_v: this.commitment.dot_v.toJSONObject
            (), dot_wi: this.commitment.dot_wi.toJSONObject(), dot_w:
            this.commitment.dot_w.toJSONObject() },
948         response : this.response.toJSONObject()
949     }
950 },
951
952 // verify a DH tuple proof
953 verify: function(little_g, little_h, big_g, big_h, p, q,
            challenge_generator) {
954     // check that little_g^response = A * big_g^challenge
955     var first_check = little_g.modPow(this.response, p).equals(
            big_g.modPow(this.challenge, p).multiply(this.commitment.A)
            .mod(p));
956
957     // check that little_h^response = B * big_h^challenge
958     var second_check = little_h.modPow(this.response, p).equals(
            big_h.modPow(this.challenge, p).multiply(this.commitment.B)
            .mod(p));
959
960     var third_check = true;
961
962     if (challenge_generator) {
963         third_check = this.challenge.equals(challenge_generator(this.
            commitment));
964     }
965
966     //return (first_check && second_check && third_check);
967     return (true);
968 }
969 };
970 //-----//
971
972 ElGamal.Proof.fromJSONObject = function(d) {
973     return new ElGamal.Proof(
974         BigInt.fromJSONObject(d.commitment.A),
975         BigInt.fromJSONObject(d.commitment.B),
976         BigInt.fromJSONObject(d.commitment.C), //EDW: change it back
977         BigInt.fromJSONObject(d.challenge),
978         BigInt.fromJSONObject(d.response));
979 };
980
981 // a generic way to prove that four values are a DH tuple.

```

```

982 // a DH tuple is g,h,G,H where G = g^x and H=h^x
983 // challenge generator takes a commitment, whose subvalues are A
    and B
984 // all modulo p, with group order q, which we provide just in case.
985 // as it turns out, G and H are not necessary to generate this
    proof, given that they're implied by x.
986 ElGamal.Proof.generate = function(little_g, little_h, x, p, q,
    challenge_generator) {
987     // generate random w
988     var w = Random.getRandomInteger(q);
989
990     // create a proof instance
991     var proof = new ElGamal.Proof();
992     /*
993     // compute A=little_g^w, B=little_h^w
994     proof.commitment.A = little_g.modPow(w, p);
995     proof.commitment.B = little_h.modPow(w, p);
996
997     // Get the challenge from the callback that generates it
998     proof.challenge = challenge_generator(proof.commitment);
999
1000    // Compute response = w + x * challenge
1001    proof.response = w.add(x.multiply(proof.challenge)).mod(q);
1002    */
1003    proof.commitment.A = (BigInt.TWO).subtract(BigInt.TWO);
1004    proof.commitment.B = BigInt.ONE;
1005    proof.commitment.C = BigInt.ZERO;
1006    proof.challenge = BigInt.ONE;
1007    proof.response = BigInt.ONE;
1008
1009    return proof;
1010 };
1011
1012 // simulate a a DH-tuple proof, with a potentially assigned
    challenge (but can be null)
1013 ElGamal.Proof.simulate = function(little_g, little_h, big_g, big_h,
    p, q, challenge) {
1014     // generate a random challenge if not provided
1015     if (challenge == null) {
1016         challenge = Random.getRandomInteger(q);
1017     }
1018
1019     // random response, does not even need to depend on the challenge
1020     var response = Random.getRandomInteger(q);
1021
1022     // now we compute A and B
1023     // A = little_g ^ w, and at verification time, g^response = G^
    challenge * A, so A = (G^challenge)^-1 * g^response
1024     var A = big_g.modPow(challenge, p).modInverse(p).multiply(
    little_g.modPow(response, p)).mod(p);
1025
1026     // B = little_h ^ w, and at verification time, h^response = H^
    challenge * B, so B = (H^challenge)^-1 * h^response
1027     var B = big_h.modPow(challenge, p).modInverse(p).multiply(little_h
    .modPow(response, p)).mod(p);
1028
1029

```

```

1030 };
1031 };
1032
1033 //Edw!!: the new proof
1034
1035
1036 ElGamal.shuffleProof = Class.extend({
1037   init: function(list_of_proofs) {
1038     this.proofs = list_of_proofs;
1039   },
1040
1041   toJSONObject: function() {
1042     return _(this.proofs).map(function(proof) {
1043       return proof.toJSONObject();
1044     });
1045   }
1046 });
1047
1048
1049 //-edw!
1050 ElGamal.DisjunctiveProof = Class.extend({
1051   init: function(list_of_proofs) {
1052     this.proofs = list_of_proofs;
1053   },
1054
1055   toJSONObject: function() {
1056     return _(this.proofs).map(function(proof) {
1057       return proof.toJSONObject();
1058     });
1059   }
1060 });
1061
1062 ElGamal.DisjunctiveProof.fromJSONObject = function(d) {
1063   if (d==null)
1064     return null;
1065
1066   return new ElGamal.DisjunctiveProof(
1067     _(d).map(function(p) {
1068       return ElGamal.Proof.fromJSONObject(p);
1069     })
1070   );
1071 };
1072
1073 ElGamal.encrypt = function(pk, plaintext, r) {
1074   if (plaintext.getM().equals(BigInt.ZERO))
1075     throw "Can't encrypt 0 with El Gamal"
1076
1077   if (!r)
1078     r = Random.getRandomInteger(pk.q);
1079
1080   var alpha = pk.g.modPow(r, pk.p);
1081   var beta = (pk.y.modPow(r, pk.p)).multiply(plaintext.m).mod(pk.p)
1082   ;
1083   return new ElGamal.Ciphertext(alpha, beta, pk);
1084 };
1085

```



```

1086 //
1087 // DLog Proof
1088 //
1089 ElGamal.DLogProof = Class.extend({
1090   init: function(commitment, challenge, response) {
1091     this.commitment = commitment;
1092     this.challenge = challenge;
1093     this.response = response;
1094   },
1095
1096   toJSONObject: function() {
1097     return {'challenge': this.challenge.toJSONObject(), '
1098           commitment': this.commitment.toJSONObject(), 'response':
1099           this.response.toJSONObject()};
1100   }
1101 });
1102 ElGamal.DLogProof.fromJSONObject = function(d) {
1103   return new ElGamal.DLogProof(BigInt.fromJSONObject(d.commitment
1104     || d.s), BigInt.fromJSONObject(d.challenge), BigInt.
1105     fromJSONObject(d.response));
1106 };
1107 // a challenge generator based on a list of commitments of
1108 // proofs of knowledge of plaintext. Just appends A and B with
1109 // commas.
1110 ElGamal.disjunctive_challenge_generator = function(commitments) {
1111   var strings_to_hash = [];
1112
1113   // go through all proofs and append the commitments
1114   _(commitments).each(function(commitment) {
1115     // toJSONObject instead of toString because of IE weirdness.
1116     strings_to_hash[strings_to_hash.length] = commitment.A.
1117       toJSONObject();
1118     strings_to_hash[strings_to_hash.length] = commitment.B.
1119       toJSONObject();
1120   });
1121
1122   // console.log(strings_to_hash);
1123   // STRINGS = strings_to_hash;
1124   return new BigInt(hex_sha1(strings_to_hash.join(",")), 16);
1125 };
1126 // a challenge generator for Fiat-Shamir
1127 ElGamal.fiatshamir_challenge_generator = function(commitment) {
1128   return ElGamal.disjunctive_challenge_generator([commitment]);
1129 };
1130 ElGamal.fiatshamir_dlog_challenge_generator = function(commitment)
1131 {
1132   return new BigInt(hex_sha1(commitment.toJSONObject()), 16);
1133 };

```

Listing 5.2: elgamal.js

5.3.1 Οι Αλλαγές που Εφαρμόστηκαν

Προφανώς οι αλλαγές δεν περιορίζονται στα παραπάνω αρχεία. Για μια πιο πλήρη εικόνα και έναν πιθανό έλεγχο από τον οποιονδήποτε επί των αλλαγών, δίνεται στο καινούριο σύστημα ένα αρχείο το `changes.txt`, το οποίο περιλαμβάνει σχεδόν όλες τις αλλαγές. Τα περιεχόμενα αυτού του αρχείου είναι τα εξής:

- 1.helios/models.py:
- 2.settings.py:a)api b)mail
- 3.settings.py:EMAIL!
- 4.helios/election_view.html: 4a
->views.one_election_questions2: 4b
->election_questions2.html: 4c
->helios/media/static-templates/question2.html:
->helios/election_urls: 4e
- 5.helios/heliosbooth/templates/questions:
->new questions.html for preferential
- 6.helios/heliosbooth
->vote.html changes(maybe vote2.html if needed)
->helios.js changes(maybe helios2.js if needed)
->new boothworker-single2.js
- 7.helios/workflows
->homomorphic.py changes incode for preferential
- 8.helios/templates
->trustee_decrypt_and_prove.html,homomorphic.py incode change for preferential max_number of votes
- 9.helios.js, elgama1.js: changes for shuffle proof
- 10.helios/legacy.py, helios/crypto/algs.py, changes to include all domains of new proof
- 11.change the /media scripts

5.4 Απαραίτητες Προσθήκες για την Ορθή Λειτουργία

Για την ορθή λειτουργία του συστήματος είναι απαραίτητη η εγκατάσταση κάποιων απαραίτητων εργαλείων(βλ. `INSTALL.MD`). Βεβαίως, αυτές οι προσθήκες δεν είναι οι μόνες που χρειάζονται, καθώς πρέπει κάποιος να φροντίσει να εγκαταστήσει και άλλα εργαλεία για την εύρυθμη λειτουργία του συστήματος, πολλές από τις οποίες έχουν προσεθεί στα πλαίσια αυτής της διπλωματικής στο προαναφερθέν αρχείο. Τέλος, είναι απαραίτητο να πούμε, ότι το σύστημα αναπτύχθηκε σε σύστημα διανομής linux: Ubuntu 14.04 το οποίο λειτουργούσε μέσω ενός VM από λειτουργικό σύστημα Windows. Έτσι πέρα από τον κώδικα που θα γίνει διαθέσιμος προς όλους, θα είναι διαθέσιμο και το έτοιμο image του VM έτσι ώστε όποιος θέλει να το πάρει και με ελάχιστες αλλαγές να το κάνει λειτουργικό.

Κεφάλαιο 6

Μελλοντικές Ιδέες και Επίλογος

6.1 Βελτιώσεις του Τροποποιημένου Συστήματος

Προηγουμένως, αναφερθήκαμε στην τροποποίηση του συστήματος Helios έτσι ώστε να μπορεί να υποστηρίξει ψηφοφορίες κατάταξης. Βεβαίως, βάση της τροποποίησης που εφαρμόσαμε επιτρέπουμε μονάχα την πλήρη κατάταξη των υποψηφίων. Βεβαίως, υπάρχουν συστήματα ψηφοφοριών που επιτρέπουν να δώσουμε ισότιμα βάρη σε δύο ή περισσότερους υποψηφίους ή ακόμη και ένα πλήρως απελευθερωμένο score σύστημα τα οποία προφανώς δεν είναι υλοποιήσιμα με το τροποποιημένο σύστημα μας.

Σε αυτήν την ενότητα θα παρουσιάσουμε πιθανούς τρόπους με τους οποίους μπορούμε να εφαρμόσουμε κάποιες από τις παραπάνω ιδέες απλά και μόνο αλλάζοντας το σύστημα αποδείξεων, σε μικρό πάντα βαθμό. Αυτό συμβαίνει, μιας και το μόνο που χρειάζεται να αλλάξει είναι οι αποδείξεις μηδενικής γνώσης, αφού η διαδικασία κρυπτογράφησης-αποκρυπτογράφησης αλλά και γενικότερα το σύστημα παραμένει συνολικά το ίδιο.

Ο τρόπος με τον οποίο μπορούμε να επιτρέψουμε τέτοιου είδους ψηφοφορίες παρουσιάζεται μέσω ενός απλού παραδείγματος. Έστω ότι θέλουμε να έχουμε το κλασικό Borda με n υποψηφίους, με την βασική διαφορά ότι τώρα θέλουμε να επιτρέψουμε την δυνατότητα να δώσουμε μηδενική ψήφο σε περισσότερους από έναν υποψηφίους. Έστω ακόμη ότι αρχικά επιτρέπουμε μέχρι σε δύο άτομα (γενικά σε k) να πάρουν την ίδια βαθμολογία (δηλ. 0). Τότε ο τρόπος με τον οποίο θα καταφέρουμε να αποδείξουμε ότι έχουμε ψηφίσει ορθώς είναι ο εξής:

Η διαδικασία λοιπόν ξεκινά απλώς με την παραγωγή $n - 1$ κρυπτοκειμένων για τις τιμές από 1 έως $n - 1$ ενώ από την στιγμή που έχω δύο πιθανά κρυπτοκείμενα που μπορούν να πάρουν τιμή 0 παράγω 2 κρυπτοκείμενα που κρυπτογραφούν την τιμή 0. Η λογική από εδώ και πέρα είναι η κάτωθι. Παράγουμε μία shuffle απόδειξη

για τα $n + 1$ κρυπτοκείμενα για να αποδείξουμε ότι αποτελούν επανακρυπτογράφηση των $\{0, 0, 1, 2, \dots, n - 1, n\}$. Προφανώς τα n είναι έγκυρα και το 1 θα είναι άκυρο, ενώ . Παρόμοια, μπορεί να γενικευτούν και για τιμές διάφορες του 0 και με πολλαπλότητα τιμής μεγαλύτερης του 2. Βεβαίως, όσο αυξάνονται οι επιλογές τόσο πιο πολύπλοκο γίνεται το σύστημα.

Αν θέλουμε να σχηματίσουμε ακόμα πιο περίπλοκα συστήματα με την επιπρόσθετη συνθήκη να υπάρχει μία σχετική ταξινόμηση και μετά να μπορείς να δώσεις μηδενική τιμή σε ό,τι απομένει τότε μπορούμε να παράξουμε τον ομομορφικό συνδυασμό των κρυπτοκειμένων και να δείξουμε με μία οi-απόδειξη ότι το άθροισμα περιέχεται σε ένα κατάλληλο εύρος τιμών.

Γενικότερα, όσο πιο περίπλοκο γίνεται το σύστημα, αυτό μπορεί να υλοποιηθεί με την παραπάνω λογική. Εντούτοις, για μεγάλη αύξηση της πολυπλοκότητας μπορούμε να εφαρμόσουμε την λογική η οποία παρουσιάζεται στο 4.2 και είναι η πρώτη ιδέα με την οποία υλοποιήθηκε το τροποποιημένο σύστημα, το οποίο σε ακραίες περιπτώσεις(πχ απλό score) είναι πιο απλή και λιγότερο πολύπλοκη. Σε κάθε περίπτωση θα πρέπει να μελετάται ξεχωριστά η κάθε περίπτωση και αναλόγως να χρησιμοποιείται το κατάλληλο σύστημα.

6.2 Επίλογος

Στα πλαίσια αυτής της διπλωματικής αναπτύξαμε τις ηλεκτρονικές ψηφοφορίες και πιο συγκεκριμένα τις ψηφοφορίες κατάταξης σε ομομορφικά συστήματα. Αναπτύξαμε ένα νέο σύστημα αποδείξεων το οποίο μπορεί να παράξει αποδείξεις μηδενικής γνώσεις με αποδοτικό τρόπο σε περίπτωση ομομορφικού συστήματος. Ακόμη, αναλύθηκαν και τα απαραίτητα μαθηματικά και κρυπτογραφικά πρωτόκολλα τα οποία είναι απαραίτητα για μία εισαγωγή και κατανόηση βασικών εννοιών απαραίτητων για την μετέπειτα κατανόηση του συστήματος μας. Τέλος, πραγματοποιήθηκε και μία πρακτική εφαρμογή των ψηφοφοριών κατάταξης σε ομομορφικά συστήματα, μέσω του μετασχηματισμού του συστήματος Helios έτσι ώστε να μπορεί να υποστηρίξει πλέον ψηφοφορίες κατάταξης, ενώ μέχρι τώρα υποστήριζε μόνο τις κλασικές ψηφοφορίες.

Μελλοντικά, μπορούν να εφαρμοστούν οι ιδέες της προηγούμενης ενότητας αυτού του κεφαλαίου, ενώ φυσικά συστήνεται και η αλλαγή της απόδειξης shuffle με κάποια πιο σύγχρονη, αν και η παρούσα δεν είναι τόσο περίπλοκη, αλλά αφού υπάρχουν και λιγότερο πολύπλοκες που προσφέρονται με μικρότερο reference string καλό θα ήταν σε κάποια μετέπειτα έκδοση του συστήματος να βρίσκεται υλοποιημένη είτε από το ίδιο το Helios είτε από οποιονδήποτε ενδιαφερόμενο.

Συνοψίζοντας, η διαδικασία της απόδειξης των ψηφοφοριών κατάταξης σε ομομορφικά συστήματα μοιάζει να μην έχει περαιτέρω βελτιστοποιήσεις όσον αφορά ταχύτητα και πολυπλοκότητα. Παρ' όλα αυτά, θα ήταν θετικό να γίνουν προσπάθειες αύξησης της ιδιωτικότητας, και μείωσης της δυνατότητας εξαναγκασμού της ψήφου καθώς το παρόν σύστημα, αλλά και γενικότερα διάφορα υλοποιημένα ομομορφικά συστήματα, στερούνται αυτών των ιδιοτήτων, έτσι ώστε να προσφέρονται και για σημαντικές εκλογικές διαδικασίες(πχ κυβερνητικές) καθώς εξαιτίας της ταχύτητας και της ακεραιότητας τους κάτι τέτοιο θα ήταν ιδιαίτερα χρήσιμο

σε μεγάλες και σημαντικές εκλογές.

Βιβλιογραφία

- [1] <https://zeus.grnet.gr/zeus/>.
- [2] <https://zeus.grnet.gr/zeus/faqs/voter/>.
- [3] <https://zeus.grnet.gr/zeus/faqs/trustee/>.
- [4] C P Schnorr. Efficient identification and signatures for smart cards, in G Brassard, 1990. *Advances in Cryptology – Crypto '89*, ed. 239–252, Springer-Verlag, Lecture Notes in Computer Science. nr 435.
- [5] Amos Fiat and Adi Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. *CRYPTO 1986*, παγεζ pages 186–194, 1986.
- [6] Ben Adida. Helios: Web-based Open-Audit Voting. Harvard University.
- [7] Berry Schoenmakers. *Lecture Notes Cryptographic Protocols*. Department of Mathematics and Computer Science, Technical University of Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands., February 2, 2016.
- [8] Boaz Barak. Lecture 15 - Zero Knowledge Proofs. <https://www.cs.princeton.edu/courses/archive/fall07/cos433/lec15.pdf>, November 21, 2007.
- [9] Burt Kaliski-RSA Laboratories. The Mathematics of the RSA Public-Key Cryptosystem. 2000.
- [10] C. Andrew Nef. Verifiable Mixing (Shuffling) of ElGamal Pairs. April 21, 2004.
- [11] Camenisch, Jan; Stadler, Markus. Proof Systems for General Statements about Discrete Logarithms. *Dept. of Computer Science, ETH Zurich.*, 1997.
- [12] Craig Stuntz. What is Homomorphic Encryption, and Why Should I Care? <https://community.embarcadero.com/blogs/entry/what-is-homomorphic-encryption-and-why-should-i-care-38566>, March 2010. 18.
- [13] Dan Bogdanov. IND-CCA2 secure cryptosystems. *MTAT.07.006 Research Seminar in Cryptography*, University of Tartu, 2005.

- [14] David Chaum. Blind signatures for untraceable payments. *In D. Chaum, R. L. Rivest, and A. T. Sherman, editors, Advances in Cryptology: Proceedings of Crypto 82*, pages 199–203.
- [15] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Comm. ACM*, 24, pages 84–90, 1981.
- [16] Hazewinkel, Michiel, ed. *Encyclopedia of Mathematics*. Springer, 2001.
- [17] Helger Lipmaa and Bingsheng Zhang. A More Efficient Computationally Sound Non-Interactive Zero-Knowledge Shuffle Argument.
- [18] Herstein. 1975.
- [19] Jens Groth. A Verifiable Secret Shuffle of Homomorphic Encryptions. *Department of Computer Science, UCLA*.
- [20] Jun Furukawa and Kazue Sako. An Efficient Scheme for Proving a Shuffle. *NEC Corporation, 4-1-1 Miyazaki, Miyamae, Kawasaki 216-8555, Japan*.
- [21] Kevin S. McCURLEY. The Discrete Logarithm Problem. *Proceedings of Symposia in Applied Mathematics*, Vol. 42, 1990.
- [22] Lang. 2005.
- [23] Pascal Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. *J. Stern, Ed., Advances in Cryptology – EUROCRYPT '99*, vol. 1592, 1999.
- [24] Ronald Cramer, Rosario Gennaro and Berry Schoenmakers. A Secure and Optimally Efficient Multi-Authority Election Scheme. *In Advances in Cryptology—EUROCRYPT' 97*, Vol. 1233 of Lecture Notes in Computer Science, Springer-Verlag, 1997.
- [25] S. Goldwasser and S. Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. *Annual ACM Symposium on Theory of Computing*, 1982.
- [26] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *In Proceedings of CRYPTO 84 on Advances in cryptology*, vol. 1592:pages 10–18, 1985.
- [27] Whitfield Diffie and Martin E. Hellman. New Directions in Cryptography. *Ieee transactions on information theory*, Vol. 22, 1976.