Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Executing and Proving over Dirty Ledgers

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΧΡΗΣΤΟΣ ΣΤΕΦΟ

Επιβλέπων : Αριστείδης Παγουρτζής
Καθηγητής Ε.Μ.Π.

Αθήνα, Νοέμβριος 2022

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Executing and Proving over Dirty Ledgers

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΧΡΗΣΤΟΣ ΣΤΕΦΟ

Επιβλέπων : Αριστείδης Παγουρτζής
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 1η Νοεμβρίου 2022.

Αριστείδης Παγουρτζής Καθηγητής ΣΗΜΜΥ Ε.Μ.Π.	Δημήτριος Φωτάκης Αναπληρωτής Καθηγητής ΣΗΜΜΥ Ε.Μ.Π.	Ελευθέριος Κόκορης Κόγιας Επίκουρος Καθηγητής Institute of Science and Technology Austria

Αθήνα, Νοέμβριος 2022

Χρήστος Στέφο

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Η κλιμακωσιμότητα των πρωτοκόλλων blockchain ώστε να ανταποκρίνονται στις αναμενόμενες ανάγκες του Web3.0 πρόκειται για ένα δύσκολο πρόβλημα με σχεδόν μια δεκαετία έρευνας. Στο προσκήνιο των τρέχοντων λύσεων βρίσκεται η ιδέα του διαχωρισμού της εκτέλεσης (execution) των ενημερώσεων που κωδικοποιούνται σε ένα μπλοκ από τη διάταξη των μπλοκ σε μια κοινή σειρά (ordering). Υπό αυτό το πρίσμα, έχει εμφανιστεί μια νέα κατηγορία πρωτοκόλλων που ονομάζονται rollups. Τα rollups λαμβάνουν ως είσοδο μια συνολική διάταξη έγκυρων και άκυρων συναλλαγών και ως έξοδο μια νέα έγκυρη κατάσταση-μετάβαση.

Αν μελετήσουμε τα rollups από την οπτική γωνία του Distributed Computing, παρατηρούμε πως παίρνουν ως είσοδο την έξοδο ενός πρωτοκόλλου Byzantine Atomic Broadcast (BAB) και τη μετατρέπουν σε ένα πρωτόκολλο State Machine Replication (SMR). Το BAB και το SMR, ωστόσο, θεωρούνται ισοδύναμα όσον αφορά τον κατανεμημένο υπολογισμό και μια λύση για το ένα μπορεί εύκολα να μετατραπεί σε λύση για το άλλο απλά προσθέτοντας/αφαιρώντας ένα βήμα εκτέλεσης πριν από την επικύρωση της εισόδου.

Αυτό το "εύκολο" βήμα της μετατροπής μιας λύσης Atomic Broadcast για την υλοποίηση ενός SMR έχει, ωστόσο, παραβλεφθεί στην πράξη. Στην παρούσα εργασία, φορμαλοποιούμε το πρόβλημα και δείχνουμε ότι μετά την επίλυση του BAB, τα παραδοσιακά impossibility result για το consensus δεν ισχύουν πλέον προς ένα SMR. Αξιοποιώντας αυτό προτείνουμε ένα κατανεμημένο πρωτόκολλο εκτέλεσης που επιτρέπει μειωμένο κόστος εκτέλεσης και αποθήκευσης ανά εκτελεστή ($O(\frac{log^2 n}{n})$) χωρίς να χαλαρώσουμε τις δικτυακές υποθέσεις του υποκείμενου πρωτοκόλλου BAB και παρέχοντας αντίσταση στη λογοκρισία. Τέλος, προτείνουμε αποτελεσματικές μη-διαδραστικές κατασκευές ελαφρών πελατών που αξιοποιούν τα αποδοτικά πρωτόκολλα εκτέλεσης και δεν απαιτούν υποθέσεις συγχρονισμού ή ακριβές αποδείξεις ZK.

Key words

State Machine Replication, Blockchain, Ordering and Execution Layer, Light Clients

Abstract

Scaling blockchain protocols to perform on par with the expected needs of Web3.0 has been proven to be a challenging task with almost a decade of research. In the forefront of the current solution is the idea of separating the execution of the updates encoded in a block from the ordering of blocks. In order to achieve this, a new class of protocols called Rollups has emerged. Rollups have as input a total ordering of valid and invalid transactions and as output a new valid state-transition.

If we study rollups from a distributed computing perspective, we uncover that rollups take as input the output of a Byzantine Atomic Broadcast (BAB) protocol and convert it to a State Machine Replication (SMR) protocol. BAB and SMR, however, are considered equivalent as far as distributed computing is concerned and a solution to one can easily be retrofitted to solve the other simply by adding/removing an execution step before the validation of the input.

This "easy" step of retrofitting an atomic broadcast solution to implement an SMR has, however, been overlooked in practice. In this paper, we formalize the problem and show that after BAB is solved, traditional impossibility results for consensus no longer apply towards an SMR. We propose multiple distributed execution protocols that allow for $n < 3f + 1$ without relaxing the network assumptions of the underlying BAB protocol. Finally, we propose efficient non-interactive light client constructions that leverage our efficient execution protocols and do not require any synchrony assumptions or expensive ZK-proofs.

Key words

State Machine Replication, Blockchain, Ordering and Execution Layer, Light Clients

Ευχαριστίες

Η εργασία προκύπτει έπειτα από συλλογική προσπάθεια με τον Ελευθέριο Κόκορη Κόγια (ISTA) και τον Zhuolun (Daniel) Xiang (Aptos Labs), κατά την διάρκεια της πρακτικής μου στο ISTA. Αρχικά, θα ήθελα να ευχαριστήσω τον Λευτέρη, για την ευκαιρία που μου έδωσε να δουλέψω στο εργαστήριο του, όπου είχα την τύχη να έρθω σε επαφή με αυτόν τον πολύ ενδιαφέρον τομέα, αλλά και να βιώσω αυτήν την ιδιαίτερα διδακτική εμπειρία, ξεκινώντας την ζωή μου από την αρχή στην Βιέννη. Ευχαριστώ λοιπόν για την άριστη αυτή συνεργασία και ανυπομονώ για μελλοντικά κοινά projects. Επιπλέον, θα ήθελα να ευχαριστήσω τον Daniel για την πολύτιμη βοήθεια του, ιδιαίτερα όσον αφορά την φορμαλοποίηση του προβλήματος. Ακόμη, θα ήθελα να ευχαριστήσω τον κ. Παγουρτζή για την εξαίρετη συνεργασία, τον κ. Φωτάκη για την συμμετοχή στην τριμελή επιτροπή, και τέλος και τους δύο μαζί τόσο για την καλλιέργεια της αλγοριθμικής μας παιδείας, αλλά και για την ενεργή παρουσία τους στο πλάι των φοιτητών, υποδεικνύοντας πως πρωταρχικά πρέπει να φροντίζουμε για την ανθρωπιστική μας παιδεία.

Κλείνοντας, ανασκοπώντας αυτά τα όμορφα χρόνια, νιώθω ιδιαίτερα ευγνώμων και πλούσιος για όλους τους ανθρώπους που γνώρισα και αποτελούν πλέον μέρος της ζωής μου. Ο καθένας συντέλεσε με τον δικό του μοναδικό τρόπο στην διαμόρφωση της προσωπικότητας μου και είμαι ιδαίτερα χαρούμενος εάν και εγώ κατάφερα να αφήσω το δικό μου αποτύπωμα στην δική τους ζωή. Θα ήθελα λοιπόν να τους ευχαριστήσω θερμά, αλλά και να μεταβιβάσω τόσο σε εμένα όσο και σε αυτούς τους ανθρώπους, το μερίδιο ευθύνης που έχουμε στο να κρατήσουμε αυτόν τον κύκλο ανοιχτό.

Χρήστος Στέφο,

Αθήνα, 1η Νοεμβρίου 2022

Περιεχόμενα

Κεφάλαιο 1

Εκτεταμένη Ελληνική Περίληψη

1.1 Εισαγωγή

Η άνοδος της τεχνολογίας blockchain έχει οδηγήσει στην ταχεία ανάπτυξη μιας ποικιλίας λύσεων για το πρόβλημα του state machine replication (SMR). Οι κόμβοι που εκτελούν ένα πρωτόκολλο SMR πρέπει τόσο να συμφωνήσουν σε μια κοινή σειρά (ordering) ενός συνόλου συναλλαγών όσο και να τις εκτελέσουν (executing) για να ενημερώσουν το τοπικό τους state, δύο ξεχωριστές αρμοδιότητες που συνήθως συγχωνεύονται σε ένα ενιαίο consensus πρωτόκολλο. Η ιδέα του διαχωρισμού των ρόλων του ordering και του execution των συναλλαγών αποτελεί μια πολλά υποσχόμενη λύση για την αύξηση της κλιμακωσιμότητας των blockchain [11, 16, 31]. Ωστόσο όλη η υπάρχουσα έρευνα επικεντρώνεται στο ordering υποθέτοντας ότι έπειτα κάθε συμμετέχων μπορεί να εκτελέσει τοπικά τις συναλλαγές και να ενημερώσει το state του.

Σε αυτή την εργασία, διερευνούμε το ερώτημα του "πώς να κλιμακώσουμε το execution μετά το ordering". Με άλλα λόγια, δεδομένου ότι οι συναλλαγές έχουν διαταχθεί σε μια κοινή σειρά, πόσο κλιμακώσιμο μπορεί να είναι ένα πρωτόκολλο εκτέλεσης (execution). Επί του παρόντος υπάρχουν δύο προτεινόμενες λύσεις. Σύμφωνα με την πρώτη και πιο διαδεδομένη, κάθε κόμβος εκτελεί τοπικά τις συναλλαγές και προσθέτει ένα state commitment σε ένα επόμενο μπλοκ [5, 11, 14]. Η δεύτερη βασίζεται σε έναν semi-trusted κόμβο που εκτελεί ένα πρωτόκολλο "rollup" [7]. Ο εκτελεστής που προτείνει το καινούριο state μετά την εκτέλεση των διατεταγμένων συναλλαγών, είτε παρέχει ένα επαρκώς μεγάλο χρονικό παράθυρο ώστε σε περίπτωση που το state είναι λάθος κάποιος έντιμος εκτελεστής μπορεί να το αμφισβητήσει με μια απόδειξη απάτης [9, 4, 2], ή μια zk-απόδειξη [13, 1] που αποδεικνύει την ορθότητα του state. Παρόλα αυτά, η πρώτη λύση είναι απλώς μια υπόθεση συγχρονισμού που αντιτίθεται στις υποθέσεις χρονισμού του ordering layer [34, 15, 14, 24] ενώ η δεύτερη προτείνεται ως λύση για την υπόθεση αυτή, η οποία επιβάλλει ως επί το πλείστον υπολογιστικά απαιτητικά πρωτόκολλα για την χρήση αποδείξεων μηδενικής γνώσης, καθώς και επιτρέπει στον εκτελεστή να λογοκρίνει συναλλαγές.

Στην παρούσα εργασία, εξετάζουμε και θεμελιώνουμε αυτό το πρόβλημα. Ως πρώτη συνεισφορά απλώς επισημαίνουμε ότι η αποσύνδεση της διάταξης σε μια κοινή σειρά (ordering) και της εκτέλεσης (executing) των συναλλαγών, δεν πρόκειται για κάτι περισσότερο από τον συνδυασμό ενός Byzantine Atomic Broadcast (BAB) [18] πρωτοκόλλου, το οποίο εξασφαλίζει τη συνολική διάταξη των συναλλαγών, και μιας ντετερμινιστικής μηχανής εκτέλεσης [19, 21, 3] για την επίλυση του προβλήματος SMR. Στα συστήματα blockchain, το στρώμα BAB ονομάζεται *dirty* ledger επειδή οι συναλλαγές δεν ελέγχονται ως προς την εγκυρότητά τους. Οι κόμβοι που

συμμετέχουν στο δίκτυο, τους οποίους αποκαλούμε *consensus nodes*, απφασίζουν σε μια κοινή σειρά των συναλλαγών χωρίς να τις επικυρώσουν ή να τις εκτελέσουν.

Αφού ορίσουμε το πρόβλημά μας, προτείνουμε ένα πρωτόκολλο για το execution layer ενός dirty ledger τόσο για τα permissioned settings όσο και για τα Proof-of-Stake settings. Το πρωτόκολλό μας λειτουργεί σε ασύγχρονο περιβάλλον, δεν κάνει επιπλέον υποθέσεις και δεν χρησιμοποιεί μηχανισμούς zk-proving. Υποθέτουμε απλώς την ύπαρξη ενός dirty ledger, το οποίο εξασφαλίζει τόσο την συνολική διάταξη όσο και τη διαθεσιμότητα των συναλλαγών που δεσμεύονται σε αυτό. Στη συνέχεια, για το επίπεδο εκτέλεσης, χρησιμοποιούμε ένα σύνολο κόμβων που ονομάζουμε $executors$. Οι $executors$, οι οποίοι μπορεί να είναι υποσύνολο των κόμβων συναίνεσης ή εξωτερικοί, επικυρώνουν συναλλαγές και ενημερώνουν την κατάσταση του συστήματος. Παραδόξως μια έντιμη πλειοψηφία των εκτελεστών είναι επαρκής, παρόλο που δεν έχουμε χρονικές υποθέσεις και μόνο ένας λογαριθμικός αριθμός από αυτούς χρειάζεται να εκτελέσει κάθε μπλοκ. Ως αποτέλεσμα, η λύση μας παρέχει τόσο καλύτερη ανοχή σε σφάλματα ($f \leq (1-\epsilon)\frac{n}{2}$ αντί για $f < \frac{n}{3}$) όσο και σημαντικά καλύτερη κλιμακωσιμότητα σε δύο διαστάσεις, εκτέλεση και αποθήκευση, με αναμενόμενο κόστος $O(\frac{log^2 n}{n})$ (αντί για $O(1)$) για κάθε εκτελεστή ανά μπλοκ που σημαίνει ότι το σύστημα μπορεί να είναι πραγματικά κλασικώσιμο και αποκεντρωμένο.

Η εργασία μας έχει τις ακόλουθες συνεισφορές.

- Φορμαλοποιούμε το πρόβλημα SMR διαχωρίζοντάς το σε ordering και execution layer.
- Προτείνουμε μια λύση για το SMR execution layer με κόστος $O(\frac{log^2(n)}{n})$ ανά εκτελεστή και σχεδόν βέλτιστη ανοχή σφαλμάτων $f < (1-\epsilon)n/2$, υποθέτοντας την ύπαρξη του ordering layer.
- Επεκτείνουμε το πρωτόκολλό μας για τις ρυθμίσεις proof-of-stake.
- Παρουσιάζουμε τον πρώτο μη-διαδραστικό light client για dirty ledgers.

1.2 Προκαταρτικά

1.2.1 Σχετική έρευνα

Ο φυσικός τρόπος για να διαχωρίσουμε το ordering από το execution είναι να αφήσουμε κάθε κόμβο να εκτελεί κάθε γύρο και να προσθέτει το state commitment σε ένα επόμενο μπλοκ, οδηγώντας σε ένα μέσο κόστος ανά εκτέλεση μπλοκ $O(1)$ [5, 11, 14]. Η άλλη πολλά υποσχόμενη προσέγγιση για τη μετακίνηση του υπολογισμού της κατάστασης εκτός αλυσίδας είναι η χρήση ενός πρωτοκόλλου *rollup* όπου ένας συντονιστής, ενημερώνει το state του συστήματος τοπικά και δημοσιεύει μόνο το state commitment στην κύρια αλυσίδα. Υπάρχουν δύο κατευθύνσεις για την επαλήθευση των state commitments, *optimistic rollups* [6] και *ZK-rollups* [8].

Στα *optimistic rollups*, υπάρχει μια περίοδος κατά την οποία οι εκτελεστές μπορούν να αποδείξουν ότι μια δέσμευση κατάστασης που έχει αναρτηθεί στην κύρια αλυσίδα είναι invalid. Ωστόσο, αυτή η τεχνική απαιτεί υποθέσεις συγχρονισμού και μέσο κόστος ανά εκτέλεση μπλοκ $O(1)$ για να εγγυηθεί ότι μόνο έγκυρα state commitments αναρτώνται στην κύρια αλυσίδα. Από την άλλη πλευρά, στα *ZK-rollups*, ο συντονιστής δεσμεύει το state commitment μαζί με μια

απόδειξη μηδενικής γνώσης (ZK-STARK) που υποδεικνύει ότι ένα συγκεκριμένο σύνολο συναλλαγών έχει εφαρμοστεί στο state. Παρ' όλα αυτά, τα *ZK-rollups* δεν είναι *censorship-resistant* αφού ο συντονιστής μπορεί απλώς να μην συμπεριλάβει κάποιες έγκυρες συναλλαγές σε αυτό το σύνολο. Επιπλέον, ο υπολογισμός των ZK-STARKs είναι υπολογιστικά βαρύς για τους χρήστες και η κλιμάκωση των εφαρμογών γενικού σκοπού είναι δύσκολη λόγω της δυσκολίας έκφρασης του γενικού υπολογισμού.

Τέλος, στον τομέα του light-client για dirty ledgers, οι Tas et al. [32] πρότειναν την πρώτη τέτοια λύση στο σύγχρονο μοντέλο. Στην εργασία αυτή, ένας αριθμός κόμβων, οι οποίοι ονομάζονται $full\ nodes$, είναι υπεύθυνοι για την ενημέρωση του state του συστήματος και την παροχή state commitments στους light clients, αποδεικνύοντας την εγκυρότητά τους μέσω ενός διαδραστικού παιχνιδιού (bisection game). Σε αντίθεση με αυτή τη λύση προτείνουμε μια κατασκευή light-client που είναι μη διαδραστική, επαληθεύσιμη από τρίτους (δηλαδή, αν ένας κόμβος πειστεί μπορεί να πείσει και άλλους κόμβους) και λειτουργεί στο ασύγχρονο μοντέλο. Αυτό όμως έχει το κόστος ότι απαιτούμε μια έντιμη πλειοψηφία εκτελεστών που δεν μπορεί να δωροδοκηθεί ή να διαφθαρεί προσαρμοστικά (για την πιθανολογική λύση). Μπορούμε επίσης να χρησιμοποιήσουμε ένα fallback μηχανισμό στο πρωτόκολλο όπου οποιοσδήποτε πελάτης που δεν είναι ευχαριστημένος με την παραπάνω υπόθεση, αλλά που υποθέτει συγχρονισμό περιμένει για οποιονδήποτε από τους εκλεγμένους εκτελεστές να στείλει μια απόδειξη απάτης [10] ή να ζητήσει από έναν τίμιο πλήρη κόμβο την ορθότητα μιας κατάστασης-δέσμευσης μέσω bisection games. Και οι δύο προσεγγίσεις έχουν μια υπόθεση ειλικρινούς μειοψηφίας η οποία θα είναι πάντα αληθής με συντριπτική πιθανότητα. Ως αποτέλεσμα, η πρότασή μας μπορεί εύκολα να προσαρμοστεί σε ένα ευέλικτο μοντέλο [26] για ετερογενείς πελάτες.

1.2.2 Μοντέλο και παραδοχές (Model and Assumptions)

Μοντέλο επικοινωνίας (Communication model): Υποθέτουμε ένα ασύγχρονο περιβάλλον, όπου κάθε μήνυμα που αποστέλλεται μπορεί να καθυστερήσει για ένα απροσδιόριστο, αλλά πεπερασμένο, χρονικό διάστημα. Η σύνδεση μεταξύ κάθε δύο έντιμων κόμβων είναι αξιόπιστη, δηλαδή όταν ένας έντιμος κόμβος στέλνει ένα μήνυμα σε έναν άλλο έντιμο κόμβο, το μήνυμα θα φτάσει τελικά.

Κρυπτογραφικά Εργαλεία (Cryptographic Primitives): Χρησιμοποιούμε το σύμβολο κ για να δηλώσουμε την παράμετρο ασφαλείας. Υποθέτουμε ότι ο αντίπαλος είναι υπολογιστικά περιορισμένος, ότι τα κανάλια επικοινωνίας είναι κρυπτογραφικά ασφαλή και ότι υπάρχουν συναρτήσεις κατακερματισμού, υπογραφές και σχήματα κρυπτογράφησης. Χρησιμοποιούμε ένα computationally hiding και perfectly binding commitment scheme: $(Compute_{cmt}, Verify_{cmt})$. Απαιτούμε το σχήμα δέσμευσης να είναι ντετερμινιστικό και να παρέχει inclusion proofs, π.χ. μπορεί να είναι ένα δέντρο Merkle. Επιπλέον, οι χρήστες χρησιμοποιούν μια Verifiable Random function (VRF) [28].

Permissioned settings: Θεωρούμε έναν σταθερό αριθμό n κόμβων με τα δημόσια κλειδιά τους γνωστά σε κάθε συμμετέχοντα στο δίκτυο. Ένα μπλοκ γένεσης G το οποίο περιγράφει το αρχικό state του συστήματος παρέχεται τόσο στους εκτελεστές όσο και στους πελάτες. Ο adversary είναι στατικός και μπορεί να διαφθείρει μέχρι και $f \leq (1-\epsilon)\frac{n}{2}$ ($\epsilon > 0$ είναι μια σταθερά) κόμβους με βυζαντινό τρόπο πριν ξεκινήσει το πρωτόκολλο.

Proof-of-Stake: Με τον όρο κόμβος αναφερόμαστε σε κάθε οντότητα που έχει λογαριασμό στο σύστημα. Το σημείο αναφοράς στο σύστημα proof-of-stake είναι ένα μοναδιαίο νόμισμα, το οποίο είναι το μικρότερο υπάρχον χρηματικό ποσό. Κάθε νόμισμα είναι μια μοναδική συμβολοσειρά που συνδέεται με τον ιδιοκτήτη του. Υποθέτουμε ότι κάθε κόμβος είναι εξοπλισμένος με ένα ζεύγος ιδιωτικού-δημόσιου κλειδιού. Ένα μπλοκ γένεσης G το οποίο περιέχει την αρχική κατανομή του stake είναι προσβάσιμο σε όλους τους κόμβους. Η κατανομή των stake είναι δυναμική, δηλαδή οι κάτοχοι των νομισμάτων μπορεί να αλλάζουν με την πάροδο του χρόνου. Υποθέτουμε ότι το συνολικό ποσό του stake είναι σταθερό και ίσο με W σε κάθε γύρο. Ο adversary είναι στατικός και μπορεί να διαφθείρει ένα μέρος των stakeholders που κατέχουν το πολύ f νομίσματα, έτσι ώστε $f \leq (1-\epsilon)\frac{W}{2}$ όπου ϵ είναι μια σταθερά, με βυζαντινό τρόπο.

1.2.3 Διατύπωση προβλήματος

Διατυπώνουμε το πρόβλημα της State Machine Replication (SMR), χωρίζοντάς το σε ένα ordering layer και ένα execution layer. Οι λύσεις για το order layer περιλαμβάνουν πρωτόκολλα Blockchain και πρωτόκολλο για το πρόβλημα Byzantine Atomic Broadcast (BAB) [34, 15, 14, 24], στα οποία οι κόμβοι συμφωνούν μόνο για τη σειρά των μπλοκ χωρίς να τα εκτελούν. Τα πρωτόκολλά μας αποτελούν λύσεις για το επίπεδο εκτέλεσης.
State Machine Replication (SMR): Μια state machine αποτελείται από ένα σύνολο μεταβλητών κατάστασης που κωδικοποιούν το τρέχον state. Εξωτερικές οντότητες, οι χρήστες του συστήματος, μπορούν δίνουν εντολές στη state machine. Η state machine εκτελεί τις εντολές διαδοχικά χρησιμοποιώντας μια συνάρτηση μετάβασης για να ενημερώσει το state του συστήματος. Επιπλέον, η state machine μπορεί να παράγει μια έξοδο μετά την εκτέλεση κάθε εντολής. Για να παρέχει συμπεριφορά ανοχής σε σφάλματα, η μηχανή κατάστασης αναπαράγεται σε πολλαπλά αντίγραφα. Ένα πρωτόκολλο SMR αποσκοπεί στη διατήρηση του συγχρονισμού μεταξύ των αντιγράφων. Στην παρούσα εργασία, παρουσιάζουμε μια λύση SMR σαν την μια σύνθεση ενός πρωτοκόλλου Π_1 για το *ordering layer* και ενός πρωτοκόλλου Π_2 για το *execution layer*. Παρακάτω, ορίζουμε τα *ordering* και *execution* layers.
Ordering Layer: Θεωρούμε έναν αριθμό κόμβων, μερικοί από τους οποίους μπορεί να είναι κακόβουλοι, οι οποίοι λαμβάνουν συναλλαγές από εξωτερικές οντότητες. Οι κόμβοι οργανώνουν τις συναλλαγές σε μπλοκ. Επιπλέον, χρησιμοποιούν ένα πρωτόκολλο Π_1 για να συμφωνήσουν σε μια σειρά των μπλοκ. Κάθε κόμβος i δεσμεύεται τοπικά στο δικό του finalized ledger. Συμβολίζουμε το ledger στο οποίο ο κόμβος i δεσμεύεται στο γύρο r με T_r^i. Η έξοδος του ordering layer διάταξης, δηλαδή η σειρά των μπλοκ στα οποία καταλήγουν οι κόμβοι, είναι ένα ledger $T = b_0 \leftarrow b_1 \leftarrow ... \leftarrow b_i$. Παρουσιάζουμε τις ιδιότητες που πρέπει να ικανοποιεί ένα ordering πρωτόκολλο:

- *O-Safety*: Δεν υπάρχει γύρος r για τον οποίο να υπάρχουν δύο τίμιοι κόμβοι i, j έτσι ώστε $T_r^i \neq T_r^j$.
- *O-Liveness*: Εάν ένας τίμιος κόμβος λάβει μια είσοδο tx, τότε όλοι οι τίμιοι κόμβοι θα συμπεριλάβουν τελικά το tx σε ένα μπλοκ του τοπικού τους ledger.

Execution Layer: Θεωρήστε έναν αριθμό κόμβων όπου κάποιοι από αυτούς μπορεί να είναι αντίπαλοι. Επιπλέον, θεωρήστε το ledger από μπλοκ $T = b_0 \leftarrow b_1 \leftarrow ... \leftarrow b_i$ που εξάγεται

από το ordering layer και είναι προσβάσιμο σε όλους. Κάθε μπλοκ μπορεί να περιέχει invalid συναλλαγές. Η εγκυρότητα μιας συναλλαγής εξαρτάται από τη λογική της εφαρμογής. Οι κόμβοι είναι υπεύθυνοι για την εφαρμογή μόνο των έγκυρων συναλλαγών εντός των μπλοκ που έχουν δεσμευτεί στο ledger T. Οι άκυρες συναλλαγές εντός των μπλοκ δεν λαμβάνονται υπόψη. Κάθε κόμβος ενημερώνει το state του συστήματος τοπικά. Συμβολίζουμε το state στο γύρο r σύμφωνα με την άποψη i του κόμβου με S_r^i.

State: Ως state ενός blockchain, εννούμε μια δομή που αντιστοιχεί σε κάθε χρήστη τα υπάρχοντα του. Το περιεχόμενο του state εξαρτάται από τον τύπο των συναλλαγών που έχουν δεσμευτεί στο ledger. Για παράδειγμα, στο Ethereum, το state αποτυπώνει τους λογαριασμούς υπολοίπου των χρηστών, ενώ στο Bitcoin υιοθετείται το μοντέλο UTXO. Επιπλέον, το state μπορεί να περιέχει τμήματα κώδικα, π.χ. smart contracts.

Ideal functionality Π*:* Παρουσιάζουμε την ορθότητα της κατάστασης του συστήματος ορίζοντας την ιδεατή οντότητα Π. Η οντότητα Π δέχεται ως είσοδο το dirty ledger $T = b_0 \leftarrow b_1 \leftarrow ... \leftarrow b_i$ το οποίο αποτελεί έξοδο από το ordering layer. Η Π ενημερώνει το state εφαρμόζοντας όλες (και μόνο) τις έγκυρες συναλλαγές εντός των μπλοκ που έχουν δεσμευτεί στο ledger T. Συμβολίζουμε το state του συστήματος που εξάγεται από την οντότητα Π για το γύρο r με S_r^*. Η αρχική κατάσταση του συστήματος S_0^* ισούται με το genesis block, $S_0^* = G$. Για την ενημέρωση του state σε κάθε γύρο, η Π χρησιμοποιεί μια ντετερμινιστική συνάρτηση μετάβασης $apply$. Οι είσοδοι είναι το state του προηγούμενου γύρου και το μπλοκ που θα εκτελεστεί στον τρέχοντα γύρο. Πιο συγκεκριμένα, στον γύρο r, $S_r^* \leftarrow apply(b_r, S_{r-1}^*) = S_{r,len(b_r)}$ όπου

$$S_{r,j} = \begin{cases} S_{r-1}^* & \text{if } j = 0 \\ \text{apply\_tx}(S_{r,j-1}, tx_j) & \text{if } 1 \leq j \leq len(b_r) \end{cases} \quad \text{και } b_r = [tx_1, ..., tx_{len(b_r)}].$$

Το state που εφαρμόζεται στη συνάρτηση *apply_tx* παραμένει αμετάβλητο όταν η είσοδος tx είναι μια invalid συναλλαγή, δηλαδή *apply_tx(S,tx)=S*. Επομένως, για το ledger T, υπάρχει μια μοναδική ακολουθία από state $S_0^*, S_1^*, ..., S_i^*$ που ορίζεται από την παραπάνω συνάρτηση μετάβασης καταστάσεων.

Στην πράξη, οι κόμβοι μπορούν να χρησιμοποιούν οποιαδήποτε μηχανή εκτέλεσης M που προσομοιώνει την ιδεατή οντότητα Π. Όταν λαμβάνει ως εισόδους το ορθό state του γύρου r και το μπλοκ $r+1$, η μηχανή M εξάγει το ορθό state του γύρου $r+1$.

Ένα επίπεδο εκτέλεσης εγγυάται ότι οι τίμιοι κόμβοι προσομοιώνουν την ιδεατή οντότητα Π. Συνεχίζουμε με τον ορισμό των ιδιοτήτων ενός execution layer:

- *E-Safety*: Δεν υπάρχει γύρος r για τον οποίο υπάρχει ένας τίμιος κόμβος i που να δεσμεύεται στο state S_r^i τέτοια ώστε $S_r^i \neq S_r^*$.

- *E-Liveness*: Για κάθε γύρο r όπου ένας τίμιος κόμβος i δεσμεύεται στο state S_r^i, υπάρχει ένας γύρος $r' > r$ όπου ο κόμβος i δεσμεύεται τελικά ένα state $S_{r'}^i$, έτσι ώστε $S_r^i \neq S_{r'}^i$.

Εφόσον οι κόμβοι συνεχίζουν να ενημερώνουν το state τους χωρίς να αποκλίνουν από την ιδεατή οντότητα Π, ένα πρωτόκολλο εκτέλεσης είναι *Censorship Resistant*, δηλαδή ικανοποιεί την ακόλουθη ιδιότητα:

- *Censorship Resistance:* Κάθε έγκυρη συναλλαγή tx που δεσμεύεται στο ledger T θα εφαρμοστεί τελικά στο state.

Σημειώστε ότι η ιδιότητα του liveness εξασφαλίζει μόνο ότι κάθε τίμιος κόμβος θα ενημερώσει τελικά το state του. Δεδομένου ότι δεν εκτελούν όλοι οι κόμβοι για κάθε γύρο ουσιαστικά, δεν απαιτούμε πως οι τίμιοι κόμβοι ενημερώνουν το state στους ίδιους γύρους.
State Machine Replication: Τέλος, διατυπώνουμε το πρόβλημα SMR πάνω από τα ordering και execution layers. Πιο συγκεκριμένα, οι συναλλαγές που εκδίδονται από εξωτερικές οντότητες αποτελούν την είσοδο του SMR. Ένα πρωτόκολλο SMR αποτελείται από ένα ordering layer πρωτόκολλο Π_1 και ένα execution layer πρωτόκολλο Π_2 που ικανοποιούν τις ιδιότητες *O-Safety*, *O-Liveness* και *E-Safety*, *E-Liveness* αντίστοιχα. Οι κόμβοι που συμμετέχουν σε αυτά τα πρωτόκολλα μπορεί να είναι ή να μην είναι οι ίδιοι. Η έξοδος του Π_1 που είναι ένα dirty ledger T είναι η είσοδος του Π_2. Η έξοδος της μηχανής είναι η έξοδος του Π_2, δηλαδή μια συνεχώς αυξανόμενη ακολουθία από states $S_0, S_1, S_2, ..., S_i$.
Light Client. Θεωρήστε ένα execution layer πρωτόκολλο Π_e με είσοδο ένα ledger T και το κατά μέσο όρο μέγεθος του state που η ιδεατή οντότητα Π εξάγει σε κάθε γύρο $|S|$. Το execution layer υποστηρίζει light clients. Οι light clients ζητούν συνοπτικές (succinct) αποδείξεις από τους συμμετέχοντες κόμβους για να μάθουν τις επιθυμητές πληροφορίες σχετικά με το state του συστήματος. Αποτυπώνουμε αυτή την ιδέα ορίζοντας τα πιστοποιητικά *απόδειξη κατάστασης (state proof)*.

- Μια *απόδειξη κατάστασης (state proof)* π_S για το γύρο r είναι μια συνοπτική (succinct) απόδειξη που δείχνει ότι το state S είναι το σωστό state του γύρου r. Η απόδειξη π_S είναι σωστή αν και μόνο αν $S = S_r^*$.

- Μια *απόδειξη κατάστασης (state proof)* π για το γύρο r είναι συνοπτική (succinct) αν περιέχει ασυμπτωτικά λιγότερα δεδομένα από την ιστορία των states, δηλαδή αν $\frac{len(\pi)}{r|S|} = o(1)$

Ας υποθέσουμε ότι ο light client lc_i λαμβάνει μια *απόδειξη κατάστασης (state proof)* π_S για τον γύρο r χωρίς απαραίτητα να λαμβάνει το state S. Ο Lc_i αξιολογεί αν η απόδειξη είναι σωστή, κατά την αντίληψή του, χρησιμοποιώντας ένα κατηγόρημα $accept_{lc_i}(\pi_S, r)$ το οποίο δίνει είτε True είτε False. Ο light client lc_i αποδέχεται την απόδειξη π_S εάν και μόνο εάν $accept_{lc_i}(\pi_S, r) = True$. Οι ιδιότητες που πρέπει να ικανοποιεί ένα light client execution πρωτόκολλο είναι οι ακόλουθες:

- *LC-Safety*: Δεν υπάρχει γύρος r για τον οποίο υπάρχει ένας light client lc_i που λαμβάνει μια απόδειξη π_S για το γύρο r τέτοια ώστε $accept_{lc_i}(\pi_S, r) = True$ και $S \neq S_r^*$.

- *LC-Liveness*: Ένας light client που μπαίνει στο σύστημα στο γύρο r θα λάβει τελικά μια απόδειξη $\pi_{S'}$ για ένα γύρο $r^*, r^* \geq r$ έτσι ώστε $accept_{lc_i}(\pi_{S'}, r^*) = True$.

1.3 Πρωτόκολλα

Σε αυτή την ενότητα, παρουσιάζουμε τα πρωτόκολλα του ασύγχρονου επιπέδου εκτέλεσης πάνω σε ένα dirty ledger. Πρώτον, στα permissioned settings, παρουσιάζουμε το ντετερμινιστικό πρωτόκολλο στον Αλγόριθμο 1 το οποίο υποδεικνύει πώς να κατασκευάζουμε επαληθεύσιμα πιστοποιητικά που αντιστοιχούν στο σωστό state, δηλαδή τα valid state commitments. Το ντετερμινιστικό πρωτόκολλο υποδεικνύει ότι η πλειοψηφία των ειλικρινών κόμβων (honest nodes) είναι αναγκαία και ικανή συνθήκη για την κατασκευή valid state commitments. Ωστόσο, κάθε κόμβος εκτελεί για κάθε γύρο με αποτέλεσμα το κόστος ανά εκτέλεση μπλοκ (cost per block execution) να είναι O(1). Στη συνέχεια, ορίζουμε ένα πιθανοτικό πρωτόκολλο που ονομάζεται Horizontal Sampling, όπου σε κάθε γύρο επιλέγουμε μόνο έναν πολυλογαριθμικό αριθμό κόμβων για εκτέλεση, έτσι ώστε η πλειοψηφία αυτών να είναι ειλικρινείς με συντριπτική πιθανότητα. Ξεκινάμε με τα permissioned settings στον Αλγόριθμο 2 και στη συνέχεια επεκτείνουμε το πρωτόκολλο Horizontal Sampling σε proof-of-stake settings στους Αλγορίθμους 3,4.

1.3.1 Ντετερμινικό πρωτόκολλο

Το ντετερμινιστικό πρωτόκολλο περιγράφεται στον αλγόριθμο 1. Κάθε κόμβος είναι υπεύθυνος να εκτελέσει σε κάθε γύρο. Πιο συγκεκριμένα, κάθε κόμβος κατεβάζει τα δεδομένα που έχουν δεσμευτεί στο dirty ledger ξεκινώντας από το genesis block και τα εφαρμόζει διαδοχικά μέσω της μηχανής εκτέλεσης M, για να αποκτήσει το execution state κάθε γύρου. Σε κάθε γύρο, οι κόμβοι υπολογίζουν και υπογράφουν το αντίστοιχο state commitment. Στη συνέχεια, μεταδίδουν το state commitment στους υπόλοιπους κόμβους. Αυτή η διαδικασία απεικονίζεται στον αλγόριθμο 1 (γραμμές: 5-11). Οι υπόλοιποι κόμβοι αποδέχονται τα υπογεγραμμένα state commitments αφού επαληθεύσουν την υπογραφή του αποστολέα (Αλγόριθμος 1 Predicate AcceptCommitment, γραμμές: 3-4). Ένα state commitment είναι έγκυρο εάν είναι υπογεγραμμένο από τουλάχιστον $f + 1$ κόμβους.

1.3.2 Horizontal Sampling πρωτόκολλο

Στο ντετερμινιστικό πρωτόκολλο, όλοι οι κόμβοι εκτελούν σε κάθε γύρο και μεταδίδουν τις δεσμεύσεις της κατάστασής τους. Τώρα, προχωράμε στην κατασκευή ενός αποτελεσματικού πιθανοτικού πρωτοκόλλου, που ονομάζεται *Horizontal Sampling*, το οποίο απεικονίζεται στον Αλγόριθμο 2. Υποθέτουμε ότι ο adversary ελέγχει το πολύ $f \leq (1-\epsilon)\frac{n}{2}$ κόμβους, όπου ϵ είναι κάποια σταθερά, και επιλέγουμε την παράμετρο ασφαλείας $\kappa = O(\log^2 n)$ για αυτή την ενότητα. Οι κόμβοι κατεβάζουν πρώτα το genesis block G που περιέχει την αρχική κατάσταση. Σε κάθε γύρο, κάθε κόμβος ελέγχει αν έχει εκλεγεί (Αλγόριθμος 2, γραμμή 28). Οι εκλεγμένοι κόμβοι προτείνουν state commitments κατά τη διάρκεια του *voting phase*. Η φάση *voting phase* εξάγει valid state commitments, τα οποία είναι state commitments υπογεγραμμένα από αρκετούς εκτελεστές.

Φάση εκλογής (Election phase): Σε κάθε γύρο, κάθε κόμβος υπολογίζει την VRF χρησιμοποιώντας το ιδιωτικό του κλειδί, τον αριθμό γύρου και τον αντίστοιχο τυχαίο σπόρο που παίρνει από το ledger. Αυτός ο υπολογισμός επιστρέφει δύο τιμές, μια τιμή κατακερματισμού

Algorithm 1: Deterministic execution protocol: Node p_i with public key pk_i and secret key sk_i

```
1  state(0) ← G, r_cur ← 1
2  threshold ← f + 1, state_com ← {}
   /* accept the state commitment cmt from the node with public key pk_j if the
      signature σ is valid                                                    */
3  Predicate AcceptCommitment(cmt, r, σ, pk_j) :
4      return Verify(σ, pk_j, cmt||r)
   /* executing for round r: applying the data committed to the ledger to update
      the state, computing and broadcasting the state commitment to everyone  */
5  Procedure Execute(r) :
6      download data(r)
7      state(r) ← apply(state(r − 1), data(r))
8      cmt ← Compute_cmt(state(r))
9      σ ← Sign(cmt||r, pk_i, sk_i)
10     state_com[(r, cmt)].add((σ, pk_i))
11     Send ("state commitment", cmt, r, σ) to all nodes
   /* Main loop.                                                              */
12 while True do
13     Execute(r_cur)
14     r_cur ← r_cur + 1
15 Upon receiving("state commitment", cmt, r, σ) from the node with public key pk_j for the first time in the
    round r do :
16     if AcceptCommitment(cmt, r, σ, pk_j) then
17         state_com[(r, cmt)].add((σ, pk_j))
```

μήκους $|h|$ και μια απόδειξη γνησιότητας που πιστοποιεί αυτή την τιμή κατακερματισμού (Αλγόριθμος 2, γραμμή 27). Αναφερόμαστε σε αυτή την απόδειξη ως *απόδειξη εκλογής (election proof)* των ηγετών. Όλοι οι κόμβοι με τιμή κατακερματισμού στο γύρο r μικρότερη από $X_r = \begin{cases} \kappa\frac{2^{|h|}}{n}, & \text{if } r = 1 \\ \frac{2^{|h|}}{n}, & \text{if } r > 1 \end{cases}$ εκλέγονται (Αλγόριθμος 2, γραμμή 28). Με αυτόν τον τρόπο, στον πρώτο γύρο αναμένονται κ εκλεγμένοι κόμβοι που αποτελούν το *bootstrap committee*, ενώ για $r > 1$ αναμένεται μόνο ένας κόμβος, ο οποίος καλείται αρχηγός(leader).

Validity of a commitment: Για τους πρώτους γύρους κ ψηφίζουν μόνο τα μέλη της επιτροπής *bootstrap committee*. Για κάθε γύρο $r \geq \kappa + 1$ όλοι οι εκλεγμένοι κόμβοι στο διάστημα $[r - \kappa + 1, r]$ υπολογίζουν τα state commitments. Στο Λήμμα 7.2.4, αποδεικνύουμε ότι το *bootstrap committee* αποτελείται από τουλάχιστον $\frac{\kappa}{2}$ ειλικρινείς κόμβους και το πολύ $\frac{\kappa}{2} - 1$ κακόβουλους κόμβους με συντριπτική πιθανότητα στο πλήθος των κόμβων n (επιλέγουμε $\kappa = O(\log^2 n)$). Η ίδια ιδιότητα ισχύει για τους εκλεγμένους κόμβους σε οποιοδήποτε διάστημα κ διαδοχικών γύρων σύμφωνα με το Λήμμα 7.2.5. Ως αποτέλεσμα, σε κάθε γύρο, τουλάχιστον $\frac{\kappa}{2}$ ειλικρινείς και το πολύ $\frac{\kappa}{2} - 1$ κακόβουλοι κόμβοι θα είναι υπεύθυνοι για την ψηφοφορία. Επομένως, μια δέσμευση κατάστασης που αντιστοιχεί σε ένα γύρο r μπορεί να θεωρηθεί έγκυρη αν υπογράφεται από τουλάχιστον $\frac{\kappa}{2}$ κόμβους μεταξύ εκείνων που έχουν επιλεγεί να εκτελέσουν κατά τη διάρκεια του διαστήματος των γύρων $[max(1, r - \kappa + 1), r]$ ή αν είναι το genesis block.

Φάση ψηφοφορίας (Voting phase): Για τους πρώτους γύρους κ, τα μέλη του *bootstrap committee* διαμορφώνουν τα αντίστοιχα valid state commitments. Για να ενημερώσουν το state, εφαρμόζουν τις συναλλαγές που έχουν δεσμευτεί στο dirty ledger για όλους αυτούς τους γύρους ξεκινώντας από το genesis block. Για κάθε γύρο, υπολογίζουν και μεταδίδουν τα υπογεγραμμένα state commitments μαζί με την απόδειξη εκλογής τους.

Ας θεωρήσουμε τώρα τον κόμβο p_i, έναν εκλεγμένο ηγέτη σε κάποιο γύρο $r \geq 2$ κατά τη διάρκεια της *φάσης ψηφοφορίας* (Αλγόριθμος 2, Διαδικασία Execute). Αρχικά, ο p_i περιμένει μέχρι να δει μια έγκυρη δέσμευση κατάστασης για τον γύρο $r-1$. Αφού λάβει την έγκυρη κατάσταση, ο αρχηγός (leader) αποκτά το αντίστοιχο state. Εάν το state δεν είναι διαθέσιμο από προηγούμενη εκτέλεση, ο p_i το ζητά από όλους τους κόμβους που έχουν υπογράψει το state commitment (Αλγόριθμος 2, γραμμές 13-14) (περισσότερα για τη διαθεσιμότητα δεδομένων στην ενότητα 2). Στη συνέχεια, ο p_i κατεβάζει τα δεδομένα που έχουν δεσμευτεί στο dirty ledger για τους ενδιάμεσους γύρους και τα εφαρμόζει διαδοχικά για να λάβει το state του γύρου $r+\kappa-1$. Για κάθε γύρο, κατασκευάζει και υπογράφει το αντίστοιχο state commitment. Τέλος, ο p_i μεταδίδει τα υπογεγραμμένα state commitments μαζί με την απόδειξη της εκλογής του στους υπόλοιπους κόμβους. Σημειώνουμε και πάλι ότι μόνο τα μέλη του *bootstrap committee* ψηφίζουν για τους πρώτους κ γύρους (Αλγόριθμος 2 γραμμή 21). Οι υπόλοιποι κόμβοι αποδέχονται τα ληφθέντα state commitments μόνο μετά την επιβεβαίωση της υπογραφής και της απόδειξης εκλογής του p_i (Αλγόριθμος 2, γραμμές 8-9).

1.3.3 Proof-of-Stake Πρωτόκολλο

Τώρα επεκτείνουμε το πρωτόκολλο *Horizontal Sampling* στα proof-of-stake settings. Οι συμμετέχοντες κόμβοι έχουν λογαριασμούς που κατέχουν stake/νομίσματα, και χρησιμοποιούμε το σύμβολο W για να δηλώσουμε το συνολικό ποσό του stake/κερμάτων στο σύστημα. Νέοι κόμβοι μπορούν να ενταχθούν δυναμικά στο σύστημα χρησιμοποιώντας τον αλγόριθμο 9. Υποθέτουμε ότι το πολύ $f \leq (1-\epsilon)\frac{W}{2}$ κέρματα μπορεί να ελέγχονται από έναν στατικό adversary, όπου ϵ είναι κάποια σταθερά, και επιλέγουμε την παράμετρο ασφαλείας $\kappa = O(\log^2 W)$ για αυτή την ενότητα.

Πρώτα, όλοι οι κόμβοι κατεβάζουν το μπλοκ γένεσης G που περιέχει την αρχική κατανομή του stake. Η κατανομή του stake μπορεί να αλλάξει με την πάροδο του χρόνου. Το πρωτόκολλο εκτέλεσης αποτελείται από τους αλγόριθμους 3, 4. Πιο συγκεκριμένα, αναλύουμε το πρωτόκολλο στις ακόλουθες φάσεις. Σε κάθε γύρο, κάθε κόμβος συμμετέχει στην Φάση *Εκλογής* (Election phase) για να ελέγξει αν κάποιο από τα νομίσματά του έχει εκλεγεί (Αλγόριθμος 3, Διαδικασία Election). Κατά τη διάρκεια της *φάσης ψηφοφορίας* (voting phase), οι κόμβοι με τουλάχιστον ένα εκλεγμένο νόμισμα υπολογίζουν τα state commitments για να σχηματίσουν τα αντίστοιχα valid state commitments, όπως στο permissioned πρωτόκολλο.

Παρακολουθώντας τον πλούτο: Η κατανομή του stake αλλάζει με την πάροδο του χρόνου και οι κόμβοι δεν αποκτούν απαραίτητα το state κάθε γύρου. Κάθε κόμβος εντάσσεται στο σύστημα με ένα συγκεκριμένο stake και ενεργοποιεί συναλλαγές, π.χ. για να πληρώσει έναν χρήστη, ή λαμβάνει συναλλαγές, π.χ. πληρώνεται από έναν χρήστη. Ως εκ τούτου, η πρόκληση για έναν κόμβο ώστε να ενημερώσει το stake του είναι να ελέγξει αν οι συναλλαγές που λαμβάνει είναι επιτυχείς ή όχι. Στο πρωτόκολλό μας, ο κόμβος ζητά ένα πιστοποιητικό *απόδειξη πληρωμής*

Algorithm 2: Horizontal Sampling: Node p_i with public key pk_i and secret key sk_i

$state(0) \leftarrow G$ // genesis block
$\texttt{threshold} \leftarrow \frac{\kappa}{2}, state\_com \leftarrow \{\}$
$r_{cur} \leftarrow 1$ // current round
/* threshold for the election process */
Procedure *Target*(r) **:**
    return $\frac{2^{|h|}}{n}\kappa$ **if** $r = 1$, **else return** $\frac{2^{|h|}}{n}$
/* verify the election proof with public key p_k in the round r */
Predicate *TimeToExecute*(p_k, r, u, π) **:**
    return *VerifyVRF*$_{p_k}(v, \pi, seed_r||r) \wedge v \leq$ *Target*(r)
/* check whether cmt comes from a valid leader of round r_l that is responsible for executing in round r */
Predicate *AcceptCommitment*$(p_k, r_l, u, \pi, r, \sigma, cmt)$ **:**
    return
      $\neg(r_l > 1 \wedge r \leq \kappa) \wedge (r_l \leq r \leq r_l + \kappa - 1) \wedge$ *Verify*$(\sigma, p_k, cmt||r) \wedge$ *TimeToExecute*(p_k, r_l, u, π)
/* acquiring the state of round r */
Procedure *AcquireState*(r) **:**
    Wait until $\exists(cmt, r)\ s.t. |state\_com[(cmt, r)]| \geq threshold$
    if $\texttt{state}(r) = null$ **then**
        request $\texttt{state}(r)$
        wait until receiving $state$ s.t. $Compute_{cmt}(state) = cmt$
        $\texttt{state}(r) \leftarrow state$
/* compute and broadcast the signed state commitments for all the intermediate rounds within the interval $[r_l, r_l + \kappa - 1]$ */
Procedure *Execute*$(r_l, (v, \pi))$ **:**
    AcquireState$(r_l - 1)$ if $r_l > 1$
    for $r = r_l, ..., r_l + \kappa - 1$ **do**
        download $data(r)$ // data within the block with height r
        $\texttt{state}(r) \leftarrow apply(\texttt{state}(r-1), data(r))$
        continue if $r_l > 1 \wedge r \leq \kappa$ // only bootstrap committee votes
        $cmt \leftarrow$ *Compute*$_{cmt}(\texttt{state}(r))$
        $\sigma \leftarrow Sign(cmt||r, pk_i, sk_i)$
        $state\_com[(r, cmt)].add((pk_i, r_l, v, \pi, \sigma))$
        Send *("state cmt"*$, cmt, r_l, r, \sigma, v, \pi)$ to all nodes
/* Main loop, run leader election for each round */
while *True* **do**
    $(v, \pi) \leftarrow VRF_{sk}(seed_{r_{cur}}||r_{cur})$
    if $v \leq$ *Target*(r_{cur}) **then**
        Execute$(r_{cur}, (v, \pi))$
    $r_{cur} \leftarrow r_{cur} + 1$
Upon *receiving("state cmt"*$, cmt, r_l, r, \sigma, u, \pi)$ *from the node with public key* pk_j *for the first time for round* r_l **do :**
    if *AcceptCommitment*$(pk_j, r_l, u, \pi, r, \sigma, cmt)$ **then**
        $state\_com[(r, cmt)].add((pk_j, r_l, u, \pi, \sigma))$

από τον πληρωτή, (βλ. ενότητα 1.5), για να επαληθεύσει ότι το state του έχει αλλάξει όπως αναμενόταν και επομένως η συναλλαγή ήταν επιτυχής.

Φάση εκλογής (Election phase): Σε κάθε γύρο, κάθε κόμβος υπολογίζει την VRF χρησιμοποιώντας τα νομίσματα που του ανήκουν και το τυχαίο seed που προέρχεται από το dirty ledger για να λάβει τις *αποδείξεις εκλογής* για τα εκλεγμένα νομίσματα. Ομοίως με το permissioned πρωτόκολλο, το πρωτόκολλο PoS εκλέγει ένα *bootstrap committee* για τον πρώτο γύρο και εκλέγει κατά μέσο όρο ένα νόμισμα ανά γύρο για κάθε γύρο $r > 1$. Για να διατηρηθεί το κατώφλι (threshold) μιας έγκυρης δέσμευσης κατάστασης ίδιο για κάθε γύρο, μόνο τα μέλη του *bootstrap committee* ψηφίζουν για τους πρώτους κ γύρους, ενώ για $r \geq 2$ ο ιδιοκτήτης ενός εκλεγμένου νομίσματος στο γύρο r μπορεί να ψηφίσει για κάθε γύρο στο διάστημα $[max(r, \kappa + 1), r + \kappa - 1]$. Ένα state commitment για τον γύρο r είναι έγκυρο (valid) εάν υπογράφεται από τους ιδιοκτήτες τουλάχιστον $\frac{\kappa}{2}$ των εκλεγμένων νομισμάτων κατά τη διάρκεια του διαστήματος γύρων $[max(1, r - t + 1), r]$ ή εάν είναι το genesis block.

Απόδειξη ιδιοκτησίας (Proof of Ownership): Δεδομένου ότι οι κόμβοι παρακολουθούν μόνο το δικό τους stake, οι εκλεγμένοι κόμβοι πρέπει να αποδείξουν ότι κατέχουν τα εκλεγμένα νομίσματα. Ως εκ τούτου, παρέχουν inclusion proofs για τα εκλεγμένα νομίσματά τους χρησιμοποιώντας το valid state commitment του προηγούμενου γύρου, π.χ. το state commitment θα ήταν το Merkle root σε μια Merkle proof. Ονομάζουμε αυτά τα πιστοποιητικά *αποδείξεις ιδιοκτησίας* και το αντίστοιχο state commitment *parent commitment*. Για να είναι σε θέση να επαληθεύσει τις *αποδείξεις πληρωμής* προκειμένου να παρακολουθήσει το stake του και να υπολογίσει τις *αποδείξεις ιδιοκτησίας* σε περίπτωση εκλογής, κάθε κόμβος περιμένει το valid state commitment του προηγούμενου γύρου πριν συμμετάσχει στη φάση της εκλογής.

Φάση ψηφοφορίας (Voting phase): Η *φάση ψηφοφορίας* είναι παρόμοια με το permissioned πρωτόκολλο. Αρχικά, τα μέλη του *bootstrap committee* υπολογίζουν και μεταδίδουν τα υπογεγραμμένα state commitments για κάθε γύρο $r \leq \kappa$ για να σχηματίσουν το αντίστοιχο valid state commitment. Στη συνέχεια, κάθε κόμβος με εκλεγμένο νόμισμα στο γύρο r, ξεκινά από το state που αντιστοιχεί στο valid state commitment του γύρου $r-1$. Επιπλέον, ο κόμβος χρησιμοποιεί το valid state commitment για να κατασκευάσει την *απόδειξη ιδιοκτησίας* για το εκλεγμένο νόμισμα του. Τέλος, ο εκλεγμένος κόμβος υπολογίζει και μεταδίδει τα υπογεγραμμένα state commitments για όλους τους ενδιάμεσους γύρους μαζί με την *απόδειξη εκλογής* και την *απόδειξη ιδιοκτησίας* στο γύρο r.

Οι κόμβοι αποδέχονται τα state commitments που προτείνονται από τους εκλεγμένους κόμβους μόνο μετά την επαλήθευση των πιστοποιητικών *proof of election* και *proof of ownership* (Αλγόριθμος 4 γραμμές 27-30). Όταν λαμβάνουν inclusion proof με ένα *parent commitment* το οποίο δεν είναι ακόμη έγκυρο (valid), αποθηκεύουν όλα τα πιστοποιητικά σε μια τοπική δομή δεδομένων (Αλγόριθμος 3 γραμμές 13-15). Όταν ένα state commitment γίνεται έγκυρο (valid) κατά την αντίληψη ενός έντιμου κόμβου p_i, ο κόμβος λαμβάνει υπόψη όλες τις ψήφους για τις οποίες είναι *parent commitment* (Αλγόριθμος 3 γραμμές: 21-24).

Bootstrapping: Θεωρήστε τον Bob, έναν κόμβο που επιθυμεί να ενταχθεί στο δίκτυο στο γύρο r. Υποθέτουμε ότι ο Bob είναι συνδεδεμένος με τουλάχιστον έναν τίμιο εκτελεστή. Ο Bob έχει λάβει από πολλούς κόμβους μια δομή δεδομένων που ονομάζεται $chain$ και η οποία περιέχει τα υπογεγραμμένα state commitments από τους εκλεγμένους κόμβους μαζί με τα αντίστοιχα πιστοποιητικά (υπογραφές, αποδείξεις εκλογής και αποδείξεις ιδιοκτησίας) για κάθε γύρο.

Ο Bob κατεβάζει πρώτα το μπλοκ γένεσης G. Για κάθε $chain$, ο Bob εφαρμόζει τον αλγόριθμο 9 για να αξιολογήσει αν είναι το σωστό. Για τον πρώτο γύρο, ο *Bob* επαληθεύει μόνο τις *αποδείξεις εκλογής* των μελών του *bootstrap committee*, καθώς η αρχική κατανομή του stake περιέχεται στο G. Στη συνέχεια, για κάθε ψήφο μέχρι το γύρο r, επαληθεύει τις υπογραφές, την *απόδειξη ιδιοκτησίας* και την *απόδειξη εκλογής* των εκλεγμένων κόμβων. Όταν ο Bob λάβει το σωστό $chain$, αποκτά το τελευταίο valid state commitment στο $chain$ και ζητά το αντίστοιχο state (ενότητα 1.5).

Algorithm 3: Functions for the election phase: node p_i with pair of keys (pk_i, sk_i)

`/* threshold for the election process */`

Procedure *Target*(r) **:**

  return $\frac{2^{|h|}}{n}\kappa$ **if** $r = 1$, **else return** $\frac{2^{|h|}}{n}$

`/*` p_i `evaluates whether a subset of its coins is elected in the round` r`, returning a list with the respective certificates */`

Procedure $Election(r, parent_{cmt})$ **:**

  $elected\_coins \leftarrow \{\}$

  for *each coin that* p_i *owns in the round r* **do**

    $(v, \pi) \leftarrow VRF_{sk_i}(coin||seed_r)$

    if $v \leq$ *Target(r)* **then**

      $\pi_m \leftarrow null$ if $r = 1$, else $\pi_m \leftarrow InclusionProof(parent_{cmt}, p_i, coin)$

      $elected\_coins.add((pk_i, r, coin, v, \pi, \pi_m, parent_{cmt}))$

  return $elected\_coins$

`/* verify the proofs of election and ownership coming from a valid leader that can vote for the round` r `*/`

Procedure *VerifyVotes*$(r, votes)$ **:**

  $valid\_votes \leftarrow \{\}$

  for $vote \leftarrow (pk_j, r_l, coin, u, \pi, \pi_m, parent_{cmt}) \in votes$ **do**

    if $|state\_com[(r_l - 1, parent_{cmt})]| < \frac{\kappa}{2}$ **then**

      $tmp\_certificates[(parent_{cmt}, r_l - 1)].add(cmt_r, r, votes)$

      continue `//` $parent_{cmt}$ `is not valid yet`

    if $(r_l = 1 \wedge pk_j$ *owns coin* $\wedge\, r \leq \kappa \wedge VerifyVRF_{pk_j}(u, \pi, coin||seed_1) \wedge u \leq$ *Target(1)*$) \vee (r_l > 1 \wedge r > \kappa \wedge r_l \leq r \leq r_l + \kappa - 1 \wedge$ *Verify*$_{InclusionProof}(\pi_m, parent_{cmt})) \wedge$ *VerifyVRF*$_{pk_j}(u, \pi, coin||seed_{r_l}) \wedge u \leq$ *Target*$(r_l))$ **then**

      $valid\_votes.add(vote)$

  return $valid\_votes$

`/* include the valid votes for` cmt`; if` cmt `becomes valid, update the votes for which` cmt `is the` ***parent commitment*** `*/`

Procedure *UpdateVotes*$(votes, cmt, r)$ **:**

  $\forall vote \in votes:\ state\_com[(cmt, r)].add(vote)$

  if $|state\_com[(cmt, r)]| \geq threshold$ **then**

    for $(cmt_{r'}, r', certificates) \in tmp\_certificates[(cmt, r)]$ **do**

      $induced\_votes \leftarrow$ *VerifyVotes*$(r', certificates)$

      UpdateVotes$(induced\_votes, cmt_{r'}, r')$ **if** $induced\_votes \neq null$

Algorithm 4: Functions for the voting phase: node p_i with pair of keys (pk_i, sk_i)

$state(0) \leftarrow G, state\_com \leftarrow \{\}, tmp\_certificates \leftarrow \{\}, elected\_coins \leftarrow \{\}, succesful\_rounds \leftarrow \{\}$
$r_{cur} \leftarrow 1$, `threshold` $\leftarrow \frac{\kappa}{2}$
$parent_{cmt} \leftarrow G$ `// valid state commitment of round` $r-1$
`/* acquire the state of the round` r `*/`
Procedure *AcquireState(r)* **:**
  if $state(r) = null$ **then**
    $cmt \leftarrow cmt'$ *s.t.* $state\_com[(cmt', r)] \geq threshold$
    request `state`(r)
    wait until receiving $state$ s.t. $Compute_{cmt}(state) = cmt$
    `state`$(r) \leftarrow state$
`/* node` p_i `computes and broadcasts its state commitments for the rounds` $[r_l, r_l + \kappa - 1]$ `to everyone along with the proofs of election and ownership of its coins in the round` r_l `*/`
Procedure *Execute*$(r_l, elected\_coins)$ **:**
  AcquireState$(r_l - 1)$ if $r_l > 1$
  for $r = r_l, ..., r_l + \kappa - 1$ **do**
    download $data(r)$
    `state`$(r) \leftarrow apply($`state`$(r-1), data(r))$
    continue if $r_l > 1 \wedge r \leq \kappa$ `// only the bootstrap committee votes`
    $cmt \leftarrow Compute_{cmt}($`state`$(r))$
    $\sigma \leftarrow Sign(cmt||r, pk_i, sk_i)$
    $\forall vote \in elected\_coins : state\_com[(r, cmt)].add(vote)$
    $Send($*"state cmt"*$, r_l, r, cmt_r, \sigma, elected\_coins)$ to all nodes
`/* Main loop.` `*/`
while *True* **do**
  Wait until $\exists(cmt, r_{cur} - 1)$ $s.t. |state\_com[(cmt, r_{cur} - 1)]| \geq threshold$ if $r_{cur} > 1$
  $parent_{cmt} \leftarrow cmt$ if $r_{cur} > 1$ `// else` $parent_{cmt} \leftarrow G$
  $elected\_coins \leftarrow Election(r_{cur}, parent_{cmt})$
  if $elected\_coins \neq null$ **then**
    Execute$(r_{cur}, elected_coins)$
  $r_{cur} \leftarrow r_{cur} + 1$
`/* Receiving` cmt `for round` r`, the respective certificates (proofs of election and ownership), and the signature` σ`.` `*/`
Upon *receiving("state cmt",* $r_l, r, cmt_r, \sigma, certificates$*) from node with public key* pk_j *for the first time for the round* r_l **do :**
  return if $Verify(\sigma, pk_j, cmt_r||r) = false$
  $valid\_votes \leftarrow$ *VerifyVotes*$(r, certificates)$
  UpdateVotes$(valid\_votes, cmt_r, r)$ **if** $valid\_votes \neq null$

1.4 Light clients

Μόλις φτιάξουμε ένα σύστημα όπου οι εκτελεστές μπορούν να επαληθεύουν ότι έχει γίνει μια πληρωμή, είναι απλό να να το μετατρέψουμε στον πρώτο μη-διαδραστικό, ασύγχρονο light-client για dirty ledgers. Σε αυτή την ενότητα, παρουσιάζουμε πώς ένας light-client μπορεί να μάθει το state του συστήματος. Πρώτον, συζητάμε πώς ένας light client μπορεί να αποκτήσει και να επαληθεύσει ένα *state proof*. Στη συνέχεια, χρησιμοποιούμε τα *state proof* ως δομικό στοιχείο για να αποδείξουμε ότι συνέβη μια αλλαγή στο state.

Υποθέσεις (Assumptions): Κάθε light client έχει πρόσβαση στον τυχαίο σπόρο (random seed) για κάθε γύρο μέσω του dirty ledger, προκειμένου να επαληθεύσει τις εκλογές των ηγετών (proof of election). Επιπλέον, κάθε light client είναι συνδεδεμένος με τουλάχιστον έναν τίμιο εκτελεστή. Ένας εκτελεστής χρησιμοποιεί ένα gossip-protocol για να αποκτήσει πληροφορίες που είναι απαραίτητες για να αντιδράσει στα αιτήματα ενός light client, όπως το state που αντιστοιχεί σε ένα valid state commitment.

Bootstrapping: Ας υποθέσουμε ότι το ύψος του dirty ledger είναι ίσο με h και ένας light client lc_i μπαίνει στο σύστημα στο γύρο $r \leq h$. Αρχικά, παρουσιάζουμε πώς ο lc_i μπορεί να επαληθεύσει ένα *state proof*. Μια απόδειξη εγκυρότητας ενός state commitment που αντιστοιχεί στην κατάσταση S αποτελεί το *state proof* π_S. Στη συνέχεια, ο light client επιλέγει πώς θα συνδεθεί στο δίκτυο. Μια επιλογή είναι να λάβει το αντίστοιχο state και να αντλήσει τις επιθυμητές πληροφορίες αφού κατεβάσει και εφαρμόσει μόνος του τα δεδομένα που έχουν δεσμευτεί στο dirty ledger. Διαφορετικά, ο lc_i μπορεί να συνδεθεί εκ νέου στο δίκτυο όποτε χρειάζεται ένα πιστοποιητικό *απόδειξη πληρωμής*.

State Proof- Permissioned settings: Για να ξεκινήσει στον γύρο r, ο lc_i περιμένει να λάβει ένα valid state commitment για κάποιο γύρο μεγαλύτερο ή ίσο με τον γύρο r. Στο *Ντετερμινιστικό πρωτόκολλο*, ο lc_i επαληθεύει ότι ένα state commitment έχει υπογραφεί από τουλάχιστον $f + 1$ κόμβους χρησιμοποιώντας το κατηγόρημα *AcceptStateProof* στον Αλγόριθμο 10. Στο πρωτόκολλο *Horizontal Sampling*, ένα valid state commitment σε ένα γύρο r' ψηφίζεται από τουλάχιστον $\frac{\kappa}{2}$ εκλεγμένους ηγέτες (leaders) στο διάστημα $[max(1, r' - \kappa + 1), r']$. Η ψήφος κάθε ηγέτη περιλαμβάνει την υπογραφή του και την απόδειξη της εκλογής του. Για να επαληθεύσει το state proof, ο Lc_i χρησιμοποιεί το κατηγόρημα *AcceptStateProof* στον αλγόριθμο 11.

State Proof - Ρυθμίσεις Proof-of-stake: Ο light client μπαίνει στο σύστημα χρησιμοποιώντας τον αλγόριθμο 9. Με λίγα λόγια, για κάθε γύρο, ο lc_i απαιτεί και επαληθεύει τις υπογραφές, την *απόδειξη ιδιοκτησίας* και την *απόδειξη εκλογής* που προέρχονται από τους ιδιοκτήτες των εκλεγμένων νομισμάτων που έχουν ψηφίσει για τα valid state commitments.

Αποδείξεις πληρωμής: Τώρα επεξηγούμε τον τρόπο παροχής πιστοποιητικών για επιτυχείς συναλλαγές. Ας θεωρήσουμε την Alice και τον Bob, δύο light clients που χρησιμοποιούν το σύστημά μας. Ο Bob επιθυμεί να αγοράσει ένα προϊόν από την Alice, προκαλώντας μια συναλλαγή που θα καταγραφεί στο dirty ledger. Η Alice χρειάζεται μια απόδειξη που πιστοποιεί ότι η πληρωμή είναι επιτυχής πριν παραδώσει το εμπόρευμα στον Bob.

Υποθέστε ότι η συναλλαγή κατά την οποίο ο Bob πληρώνει την Alice δεσμεύεται στο γύρο r. Το πιστοποιητικό με το οποίο ο Bob αποδεικνύει ότι το state της Alice έχει αλλάξει στο γύρο r ονομάζεται *απόδειξη πληρωμής*. Πιο συγκεκριμένα, το πιστοποιητικό αποτελείται από ένα valid state commitment για κάποιο γύρο μεγαλύτερο από r και ένα inclusion proof (π.χ. Merkle proof)

που δείχνει το νέο state της Alice. Η Alice χρησιμοποιεί το Predicate PaymentProof στον Αλγόριθμο 11 για να επαληθεύσει πρώτα την εγκυρότητα του state commitment και στη συνέχεια το inclusion proof, χρησιμοποιώντας το valid state commitment, για να εξάγει το νέο της state. Εάν η συναλλαγή είναι επιτυχής, το state της Alice έχει αλλάξει κατά τη διάρκεια αυτού του διαστήματος.

1.5 Διαθεσιμότητα Δεδομένων

State availability: Οι κόμβοι που είναι υπεύθυνοι για την εκτέλεση ενός συγκεκριμένου γύρου πρέπει πρώτα να αποκτήσουν το state του προηγούμενου γύρου. Το state απαιτείται επίσης από την *Απόδειξη πληρωμής* και το bootstrapping στις ρυθμίσεις proof-of-stake.

Αρχικά αφήνουμε τους εκτελεστές να αποθηκεύσουν κάθε state για τους γύρους που εκτέλεσαν. Σε όλα τα προτεινόμενα πρωτόκολλα, κάθε έγκυρη δέσμευση κατάστασης υπογράφεται από τουλάχιστον έναν τίμιο κόμβο που έχει αποθηκεύσει το state (το επιχείρημα ισχύει με συντριπτική πιθανότητα στα πιθανοκρατικά πρωτόκολλα). Ένας εκτελεστής ζητά το state που αντιστοιχεί σε ένα *valid* state commitment από όλους τους κόμβους που έχουν υπογράψει το state commitment. Ο ειλικρινής κόμβος που έχει υπογράψει το state commitment θα το στείλει τελικά στον εκτελεστή. Ο εκτελεστής θα επαληθεύσει ότι το state πράγματι αντιστοιχεί στο valid state commitment.

Certificate availability: Για το bootstrapping, οι εκτελεστές αποθηκεύουν τα πιστοποιητικά που σχετίζονται με τα valid state commitments. Στο ντετερμινιστικό πρωτόκολλο, αποθηκεύουν μόνο τα υπογεγραμμένα state commitments (Αλγόριθμος 1 γραμμές: 10, 17). Στο πρωτόκολλο Horizontal Sampling, οι εκτελεστές αποθηκεύουν τα υπογεγραμμένα state commitments μαζί με τις αποδείξεις εκλογής των ηγετών (proofs of election) (Αλγόριθμος 2 γραμμές: 24, 33), και στις ρυθμίσεις proof-of-stake, διατηρούν επιπλέον τις *αποδείξεις ιδιοκτησίας (proofs of ownership)* των εκλεγμένων νομισμάτων (Αλγόριθμος 3 γραμμή 20, Αλγόριθμος 4 γραμμή 18).

Κείμενο στα αγγλικά

Chapter 1

Cryptographic Background

1.1 Negligible Function and Security Parameter

When defining cryptographic protocols we must quantify the level of security. To this end, we use the notion of a security parameter, which is a parameter of a cryptographic scheme, such as the length of the private key in an encryption scheme. The rationale behind the security parameter is that the system becomes more secure as long as we increase this parameter. In that way, we formulate the security of a system probabilistically by requiring a set of properties to hold except with negligible probability in the security number.

A function f is negligible in the variable x if and only if for any fixed polynomial function $p(x)$, there exists x_0 such that for any $x > x_0$ it holds that $f(x) < \frac{1}{p(x)}$. In our proposed protocols in this thesis, we require the properties defined to hold except with a probability of $e^{-log^2 k}$ where k is the security parameter.

1.2 Digital Signatures

Consider Alice and Bob, two users interacting in a network. Alice sends the message m to Bob. Bob requires a certificate for message authentication that binds the message m to the sender, to make sure that the message indeed comes from Alice and that Alice will not claim later that she had sent a message $m' \neq m$. To this end, we use Digital Signature Schemes.

A Digital Signature Scheme takes as input a security parameter κ and consists of the algorithms: Gen,Sign, Vf. Each user u_i, first uses the function Gen which generates a public key pk and a corresponding secret key sk. The function Sign takes as input a secret key sk and a message m and outputs a signature σ. Every time that user u_i wishes to send a message m to another user, it sends m along with the corresponding the signature σ. Finally, to verify a signature coming from another user, users employ the function Vf, which takes as input the message m, the signature σ, and the public key pk. Vf outputs $true$ only if the signature σ for the message m was constructed by the user with the public key pk. The correctness of the functions Sign and Vf hold with overwhelming probability in the security parameter κ.

1.3 Commitment Scheme

Consider, two users, Alice and Bob, in the following scenario. Alice has a private input v, which she does not reveal to Bob immediately ($hiding$ property). Alice wants to commit to the value v, producing a certificate which corresponds uniquely to v ($binding$ property). Therefore, when she reveals v to Bob later, Bob will be sure that it was indeed Alice's input.

To achieve that, Bob and Alice can leverage a Commitment Scheme consisting of the algorithms Setup, Commit, and Reveal. Initially, users employ the function Setup with a security parameter κ to define the parameters of the system, e.g., the public and secret key pairs of the users, random coins in case that the commitment scheme is deterministic. Alice uses the function Commit with her private input v which outputs the commitment c. Bob can verify that the commitment c corresponds to the value v, when Alice sends the value, using the function Reveal. As an example of a Commitment Scheme, we present *Merkle Trees* in Section 1.4.

1.4 Merkle trees

1.4.1 Definition of Merkle trees

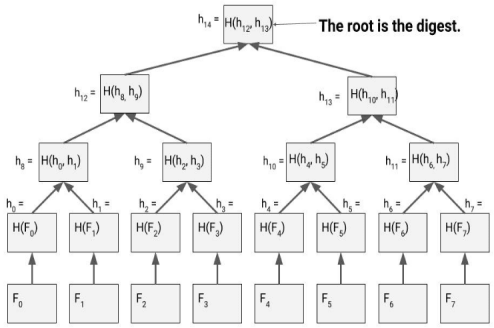
A Merkle tree is a type of data structure that takes as input a number of files and outputs a hash, the *Merkle root hash*. *Merkle trees* can be used as a vector commitment scheme, since the *Merkle root hash* of any two sequences of data S, S' equals only if the sequences are identical. Merkle trees are broadly used because it is feasible to provide *proof of inclusion*. In particular, we can prove that a file is contained in a Merkle tree by providing a proof of logarithmic size on the total number of files stored on the tree. Moreover, no one can offer a faulty *proof of inclusion*, i.e. claim that a file that is not saved on the tree actually exists.

Consider that we want to store a number of n files $F_i,\ i \in \{1, 2, ..., n\}$ in a Merkle tree. To construct the Merkle tree we assume the existence of a collision-resistant hash function H. The hash of the file x_i $(H(F_i))$ is stored on the $i-th$ leaf of the Merkle tree. On each internal node of the tree is stored the hash that results from concatenating the hashes of its two children. By repeating this process until reaching the root node of the tree, we acquire the Merkle root hash. This process is illustrated in image 1.1a.

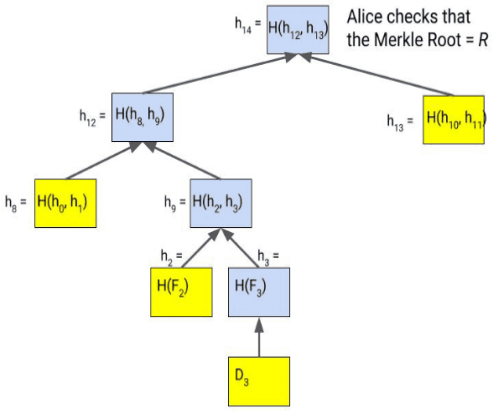
1.4.2 Merkle proof

Assume that we have stored n files on a Merkle tree. Now we want to prove that the file D_i is contained in the Merkle tree. Each node that wants to verify the *Merkle proof* must know the Merkle root hash in advance. To provide a *Merkle inclusion proof* for the file D_i, it is necessary to provide only the file and the hashes needed to reconstruct the Merkle root hash. For instance, in image 1.1b, Alice verifies that the file D_3 is included in the tree by having only the hashes needed along the path to the Merkle root.

To construct the *Merkle proof* we just include one hash per level. Merkle trees are balanced trees and therefore the size of a Merkle proof is $O(logn)$, where n is the number of the stored files. It is impossible for someone to offer a *Merkle proof* for a file that is not present in the Merkle tree, since we use collision-resistant hash functions.



(a) Merkle tree defined by the data F_i.



(b) Merkle proof

1.5 Verifiable Random Functions

A Verifiable Random Function (VRF) is a cryptographic function that produces a pseudo-random value along with a proof so that anyone can verify non-interactively the occurring value.

Assume that a user with private-public key pair (sk, pk) computes the VRF for the message m. The result $VRF(m, sk) = (hash, \pi)$ corresponds to two values. First, a $hash$ value which is in fact a pseudorandom value. The second value π, is a proof of authenticity so that anyone can verify that the $hash$ value indeed occurs after the user with public key pk computed the VRF with the message m. The users verify the proof π by calling a function $Verify(m, \pi, pk)$.

Chapter 2

From single-shot Consensus to Blockchains

2.1 Byzantine Broadcast

The *consensus* problem was introduced by Lamport et al. [25], called as *Byzantine Generals problem*, in the following variant:
"*Imagine that several divisions of the Byzantine army are camped outside an enemy city, each division commanded by its own general. The generals can communicate with one another only by messenger. After observing the enemy, they must decide upon a common plan of action. However, some of the generals may be traitors, trying to prevent the loyal generals from reaching agreement.*"

Consider a number of n generals, where one of them is the commanding general. The commanding general proposes to all the generals either to ATTACK or to RETREAT. A solution to the *Byzantine Generals problem* problem should guarantee that:

- All loyal generals reach the same decision
- If the commanding general is loyal, then all loyal generals will obey the commanding general's order.

Problem definition:

That problem is commonly known as *Byzantine Broadcast* (BB). We will now proceed with the definition of the *Byzantine Generals* problem according to [30].

We refer to the generals as $nodes$ and to the commanding general as the *designated sender*. We call the nodes that follow the protocol honest nodes and those that do not follow it corrupted nodes. A corrupt node acts arbitrarily, e.g. it might omit messages or send arbitrary messages at arbitrary times. We assume that a pair of keys, a secret and the respective public key, corresponds to each node. The public keys of the nodes are accessible to everyone and each node can communicate with each other.

The designated sender receives an input bit $b \in \{0, 1\}$ before the protocols starts. When the protocol ends each honest node outputs a bit. The protocol for the Byzantine Broadcast problem must satisfy the following properties:

- Consistency : If two honest nodes output b and b' respectively, then b = b'.
- Validity : If the sender is honest and receives the input bit b, then all honest nodes should output b

Consistency guarantees that there will not be a disagreement on the output of any two honest nodes. Validity ensures that the output of each honest node will be a function of the inputs. Otherwise, a trivial solution like having all the nodes outputting the bit "0" would be acceptable.

2.2 State Machine Replication and Blockchains

In section 2.1, we introduced the problem of single-shot consensus, i.e. all the honest nodes should agree on a single value. Consensus is one of the core functionalities for many applications. In practice, multiple deployed systems need to reach consensus repeatedly over time. At any point in time, the inputs on which the nodes should agree might depend on past decisions. To this end, these applications maintain a public ledger keeping track of the inputs (transactions) that have been applied so far. In the distributed systems literature, this notion is called *state machine replication*. After Nakatomo introduced Bitcoin [29] in 2008, the notion of such an ever-growing public ledger is broadly called *blockchain*.

State machine: A state machine receives inputs from clients and is responsible for providing the corresponding outputs. At any point in time, the state machine stores the state of the system. That is done by updating the state after applying each received input sequentially.

Blockchain: A Blockchain is just a chain of blocks as its name indicates. Each block contains a set of transcations. Batching is used, instead of agreeing on a single value in each block, to improve throughput. Each block extends a previous block. That is done by referencing a previous block using cryptographic tools. In that way, the order of the transactions committed to the ledger is uniquely determined by a chain of blocks, the *blockchain*.

Note that a Blockchain is in fact a state machine. The inputs received from the users are the transactions committed to the ledger. The output of the state machine is the ledger. The state of the system attributes to each user its possessions. For instance, in cryptocurrencies that use blockchain technology the state of the system might correspond to the balance accounts of the users. As soon as a new valid block is committed to the ledger, the nodes participating in the network update the state by applying all the transactions contained in the block.

Blockchain - Problem definition:

Consider a number of n nodes. The nodes receive transactions from an external environment. Each node maintains an ordered log of transactions. We call LOG_i^r the finalized log of node i in the round r. We will give an abstraction of the properties that a Blockchain (and a state machine) should satisfy, again according to [30].

As discussed in single-shot consensus, any protocol dealing with the *byzantine broadcast* problem should guarantee that: i) there will not be any disagreement between any two honest nodes ($consistency$), ii) if an honest leader proposes a value, every honest node will output this value ($validity$). In a blockchain protocol, due to delays in the network, the finalized log of two nodes might differ in length. $Consistency$ now requires that if we take any two honest nodes in any round, the finalized log of one of them will be a prefix of the finalized log of the other. In that way, we guarantee that the two nodes will not disagree on the ordering of the transactions. Furthermore, instead of the validity, which takes into account only values proposed by honest leaders, now we want to ensure that every transaction seen by one honest node will eventually be contained in the finalized log of all of the other honest nodes. We call this property $liveness$.

Combining $consistency$, i.e. honest nodes do not disagree on the order of the transactions, and $liveness$, i.e. honest nodes will eventually add to their finalized logs the same transactions, leads to the fact that every two honest nodes will eventually have identical finalized logs. Every blockchain protocol must satisfy the following properties, even under the presence of corrupted parties:

- Consistency: for any honest nodes i and j, and for any round numbers t and r, it must be that $LOG_i^t \preceq LOG_j^r$ or $LOG_i^t \preceq LOG_j^r$. Here $LOG \preceq LOG^0$ means that the former log is a prefix of the latter or they are the same
- $T_{conf} - liveness$: If an honest node receives some transaction tx as input in some round r, then by the end of round $r + T_{conf}$, all honest nodes' local logs must include tx.

2.3 Bitcoin

In 2008, Satoshi Nakamoto revolutionized the way we perceive financial systems by proposing Bitcoin: the first peer-to-peer electronic cash system [29]. Bitcoin introduced the notion of blockchains, which is now well established. Bitcoin is in fact the first secure and verifiable open distributed system. It is open/permissionless, meaning that users can join and leave the network anytime without even the need for identities. Each transaction committed to the ledger confirmed according to the consensus rules cannot be changed. Any user can download and verify the ledger according to the consensus mechanism. We discuss briefly some of the basic tools that are used in Bitcoin, such as the way in which transactions are performed and the consensus mechanism.

Adressess and transactions: To receive and spend funds, each user generates a number of key pairs consisting of a private and the respective public key. A Bitcoin address corresponds to a public key and is used as a username of its owner. Whoever wants to pay the user, can do that at one of his Bitcoin addresses. The private key corresponding to a Bitcoin address authorizes the owner of the address to spend the funds associated with this address. The funds associated with an address are the unspent transactions (UTXOs) previously sent to this address.

Consider that a user initiates a transaction to pay some users. The transaction consists of a subset of inputs and a subset of outputs. The inputs reference the unspent transactions previously sent to the sender's address. To spend these transactions, the user has to provide a signature that acts as a witness of knowing the private key associated with an address. The output includes the addresses of the receivers and the number of bitcoins that each of them will be paid.

Proof of Work and Longest chain rule: Nodes extending the ledger are called *miners*. The miners that are eligible to propose a new block are the ones that manage to solve the proof-of-work puzzle. More specifically, a miner that wants to propose a batch of transactions, creates a block including these transactions, its public key, a reference to the previous valid block on the chain, and repeatedly hash that info by changing a nonce. If the result is a sufficiently small value, the *target*, the miner is eligible to propose a new block. By including a reference to the previous valid block on the chain, the miner commits (or votes) on a chain of blocks, known as the *blockchain*. A block is considered to be valid if and only if the transactions included in it are valid, the hash header is less than the target, and it is a part of the longest chain.

Since the system is permissionless, the adversary can create multiple accounts (addresses) to participate in the network. This attack is known as *Sybil attack*. Bitcoin defends against this attack since no matter how many accounts the adversary creates, its contribution to the network will always be proportional to its computational power. Moreover, the total ordering of the transactions is uniquely determined by the ledger. The chain that determines the total ordering, acceptable by the honest users, contains only valid blocks and thus the adversary cannot *double spend*, i.e., spend the same funds to different transactions. To *double spend*, the adversary has to construct a longer chain than the one adopted by that time by the rest of the users. To do that, the adversary would need to hold the majority of the computational power.

2.4 Scalability of Blockchains

2.4.1 The scalability Problem

A fundamental challenge of blockchain technologies is known as *blockchain trilemma*. The *blockchain trilemma* illustrates the difficulty of constructing a blockchain system that is secure, scalable, and decentralized. First, we discuss why these properties are desired. Then, we briefly outline why defining a blockchain protocol that satisfies all these properties simultaneously is challenging. Lastly, we proceed with mentioning broadly accepted solutions to the scalability problem.

Decentralization: In centralized networks, there is a master node that has a greater impact on the protocol than the rest of the nodes. On the contrary, in decentralized networks, all the nodes participating should have an equal contribution. Decentralized networks ensure transparency, since all the nodes validate the data flowing in the network. Moreover, decentralized systems are more corruption-resistant since an adversary requires a large portion of the resources to corrupt enough nodes, to attack the system. Instead, in centralized networks, there is a single point of failure. The adversary can focus on attacking only the master node of the network.

Scalability: With the term *scalability* we refer to the capability of a blockchain: i) confirming transactions fast, ii) committing transactions to the ledger fast. Confirming transactions fast is quantified by having low confirmation latency, i.e. the time that a user would have to wait to be sure that a transaction of its interest is confirmed according to the consensus rules. Moreover, throughput measures the number of transactions committed to the ledger per second and is usually measured to tps (transactions per second). When the above properties are satisfied, we have a ”fast” ledger, namely a ledger in which transactions are committed often with a low confirmation latency.

Security: Finally, we capture *security* by requiring that to damage the system, an adversary should possess a large portion of the resources (that ideally will be an infeasible scenario).

We will now discuss intuitively the trade-offs when trying to satisfy the three properties simultaneously. First, to achieve decentralization, all the nodes participating in the network should reach a consensus before each transaction is confirmed. Moreover, the amount of resources that an adversary should have to attack the system is proportional to the number of nodes in the network. For instance, in Bitcoin, to be feasible for the adversary to double-spend it would have to possess the majority of the computational power in the network. Therefore, to achieve security,

we need a large number of participating nodes so that it would be infeasible for the adversary to attack the system. However, the more nodes distributed in the network, the more challenging it becomes to scale the system since all of them need to reach a consensus on confirming blocks of transactions.

2.4.2 Scalability Problem: Proposed solutions

We can categorize the proposed solutions to i) layer 1 scalability solutions: focusing on altering the main blockchain, ii) layer 2 scalability solutions: defining off-chain solutions.

Layer 1 solutions

Changing the parameters of the protocol: First, we can change the core components of the main network, such as increasing the block size or reducing the time required for creating a block. However, these actions might not be sufficient to improve significantly either the throughput or the confirmation latency.

For instance, in Bitcoin, confirmation latency is proportional to the time it takes to commit new blocks in the ledger. Decreasing the time for creating a block decreases security though. Intuitively, due to the delays on the network, users might have different views at any point in time. The faster that miners produce blocks, the more possible it is for users to adopt different chains ($fork$). Therefore, to improve throughput, we can only increase the block size. Yet, empirical results indicate that increasing the block size increases the network delay [17].

Sharding: In sharding the state of the system is divided in smaller parts, the *shards*. Each shard is maintained by a small portion of nodes following the same consensus rules. The shards run in parallel managing only a fraction of the transactions load.

Sharding faces multiple challenges though. First, the nodes should be assigned to the shards randomly. Otherwise, the adversary could focus on corrupting the nodes responsible for a single one shard. Solutions providing distributed randomness generation are heavy in terms of communication complexity. Due to the attack stated above, sharding cannot deal with a fully adaptive adversary, i.e. an adversary that can corrupt nodes immediately. To this end, sharding protocols usually assume a slowly adaptive adversary and reassign nodes to the shard often which is a costly process. Moreover, another difficulty to deal with is to handle transactions that involve multiple shards. Avarikioti et al. [12] formalizes the notion of sharding blockchains, outlying the underlying possibilities and the limitations.

DAG blockchains: The difference between DAG and simple blockchains lays in the data structure used. More specifically, in DAG blockchains, instead of forming a chain of blocks, blocks are organized as a Direct Acyclic Graph to parallelize the proccessing and the validation of data [33].

Layer 2 solutions

State Channels: In state channels, two nodes can perform transactions off-chain and commit on chain only the final state. The off-chain channels can offer faster transactions with limited fees. The participants on a state channel must be well known and lock a portion of their funds before opening the channel. To commit on chain in a valid final state, we can deploy either a network of nodes or a smart contract monitoring the transactions performed between the two nodes. Unfortunately, the notion of a state channel works only in specific applications and cannot be used to scale general-purpose smart contracts.

Sidechains: A sidechain is an independent blockchain that runs on the side of the main blockchain. The sidechain operates under its own consensus mechanism. The users can transfer data between the sidechain and the main chain using utility tokens. Sidechains remove the transaction workload off-chain. However, that comes with the challenge that nodes performing transactions on the sidechain are not under the security of the main blockchain.

Rollups: The core idea of rollups is executing transactions off-chain and then posting a compressed form of relevant data on the main chain to ensure security. To achieve this, there must be a way to prove the validity of the compressed data. To this end, there are two different directions, *optimistic rollups* and *ZK-rollups*.

In *optimistic rollups*, anyone can post a batch of data in a compressed form on the main chain. There is a dispute period in which anyone can claim that a rollup of data posted on the main chain is not valid. That can be done by providing a *proof of fraud* to a smart contract. An obvious challenge is data availability issues. In particular, to verify a *proof of fraud*, a smart contract has to re-execute the transactions by the state of the system as it was before the transactions were applied. Furthermore, there are synchrony assumptions needed so that the users manage to provide *proofs of fraud* during the dispute periods.

Optimistic rollups provide reactive security, meaning that actions are performed only when nodes committing rollups on-chain misbehave. Instead, *ZK rollups* provide proactive security. A user posting a rollup of data has to do it accompanied with a Zero Knowledge Proof (STARK) which will be verified by a smart contract. In practice, *ZK-rollups* seem to be hard to adopt so far, since computing ZK-STARKs is a computationally heavy process for the users.

Chapter 3

Problem Introduction

3.1 Introduction

The rise of blockchain technology has lead to the rapid development of a variety of solutions for the State Machine Replication (SMR) problem. Nodes running an SMR algorithm need to both order a set of transactions as well as execute them to update their local state, two separate responsibilities that are usually conflated into a single consensus protocol. Recently, the idea of separating the total ordering of transactions from the execution has shown tremendous promise on increasing the scalability of blockchains [11, 16, 31] however all existing research focuses on the ordering layer assuming that after ordering every participant can locally execute the transactions and update the state.

In this work, we investigate the question of "how to scale execution after the ordering is done". In other words, given that transactions are ordered, how scalable can an execution protocol be. Currently there exist two proposed solutions. The first and most prevalent is that every consensus-node also executes and adds a commitment to the new-state on a succeeding block [5, 11, 14]. The second relies on a semi-trusted executor node that runs a "rollup" protocol [7]. The executor proposing a new state after locally executing the ordered transactions either provides a sufficiently large dispute window for some honest executor to challenge the proposal with a fraud proof [9, 4, 2], or a zk-proof [13, 1] of correct execution. Neither of these solutions are built from first principles, the former is merely a synchrony assumption breaking the model of the underlying ordering-layer [34, 15, 14, 24] whereas the later is proposed as a remedy to that assumption which forces mostly inefficient and non-general purpose zero-knowledge proof usage as well as allows for the executor to censor transactions.

In this paper, we take a step back and design from first principles. As a first contribution we merely point out that decoupling of ordering from execution is nothing more than taking a Byzantine Atomic Broadcast (BAB) [18], i.e. ensuring the total ordering of sent messages, and a deterministic execution engine [19, 21, 3] to solve the SMR problem. In blockchain systems, the BAB layer is called a *dirty* ledger because transactions are not checked for validity. The nodes taking part in the network, which we call *consensus nodes*, commit transactions without validation and only make sure the ledger is growing consistently.

Once we define our problem we propose a novel protocol for the execution layer of an underlying dirty ledger for both the permissioned and the Proof-of-Stake settings. Our protocol works in an asynchronous environment, making no extra assumptions and does not use zk-proving ma-

chinery. We merely assume the existence of a dirty ledger, that ensures both the total ordering and the availability of the transactions committed to it. Then, for the execution layer, we use a set of nodes that we call $executors$. They validate transactions and update the state of the system and can be a subset of the consensus nodes or external. Surprisingly an honest majority is sufficient for the executors even though we have no timing assumptions and only a logarithmic number of them needs to execute every block. As a result our solution provides both better fault tolerance ($f \leq (1-\epsilon)\frac{n}{2}$ instead of $f < \frac{n}{3}$) and significantly better scalability in two dimensions, execution and storage, with expected $O(\frac{log^2 n}{n})$ (instead of $O(1)$) cost per executor per block meaning that the system can be truly scalable and decentralized.

Our Approach

Our protocol can be roughly split into two steps. In the first step, we *elect* on expectation one executor per round by computing a VRF [28]. Then the executor *votes* by computing state commitments for the next $O(log^2 n)$ rounds. Hence every round will have an $O(log^2 n)$ number of executors. Their task is to construct verifiable certificates of the state such that a user ($executor$ or light client) can be convinced about the state without execution. At first glance, that problem seems related to the consensus problem [25] since executors need to agree on the state, but unlike the consensus problem, in each round all honest nodes have the same input, an ordered list of transactions. Therefore, as long as honest executors bootstrap in the correct state, a state commitment can be considered valid if and only if at least one honest executor has voted on it.

Since nodes update the state in a distributed fashion, we must guarantee its availability. More specifically, in each round, only the elected nodes obtain the state. However, to vote for the following $O(log^2 n)$ rounds, elected executors must acquire the state of the previous round. For that reason, every node stores the state of the rounds it has executed and provides it upon request.

A final general challenge for dirty ledgers is to define light client constructions. Straightforward solutions such as providing inclusion proofs for the transactions committed to the ledger are not sufficient since the transactions can be invalid. To solve this challenge, we present the first non-interactive light client construction for dirty ledgers in the asynchronous model. In a nutshell, light clients learn information about the state by verifying the certificates produced by the executors, i.e., the valid state commitments, along with inclusion proofs.

Our paper has the following contributions.

- We formalize the SMR problem by separating it into ordering and execution.
- We propose a solution for the SMR execution layer with $O(\frac{log^2(n)}{n})$ cost per executor and near-optimal fault tolerance $f < (1-\epsilon)n/2$, assuming the existence of the ordering layer.
- We extend our protocol for the proof-of-stake settings.
- We introduce the first non-interactive light client for dirty ledgers.

Structure of the paper. In Section 3.2 we present the related work, model and assumptions, as well as the problem definition of scaling execution. In Section 3.3, we overview our solutions for different settings, including the Deterministic, the Horizontal Sampling, and the Proof-of-Stake protocols and we provide the full solutions in Chapter 4. Chapter 5 discusses the light

client protocol and Chapter 6 discusses the data availability problem. We provide a summary of terminologies in table 4.1, and all proofs of the protocols in section 7.

3.2 Preliminaries

3.2.1 Related work

The natural way to separate the ordering from the execution is to let each node execute every round and add the state commitment to a subsequent block, leading to an average cost per block execution $O(1)$ [5, 11, 14]. The other promising approach to moving the computation of the state off-chain is employing a *rollup* protocol where a coordinator, updates the state of the system locally and only posts the state commitment on the main chain. There are two directions to verify the state commitments, *optimistic rollups* [6] and *ZK-rollups* [8].

In *optimistic rollups*, there is a dispute period during which executors can prove that a state commitment posted on the main chain is invalid. However, this technique requires synchrony assumptions and average cost per block execution $O(1)$ to guarantee that only valid state commitments are posted on the main chain. On the other hand, in *ZK-rollups*, the coordinator commits the state commitment along with a zero-knowledge proof (ZK-STARK) indicating that a specific set of transactions has been applied to the state. Nevertheless, *ZK-rollups* are not *censorship-resistant* since the coordinator can just not include some valid transactions in this set. Furthermore, computing ZK-STARKs is a computationally heavy for the users, and scaling general-purpose applications is challenging due to the difficulty to express general computation.

Finally, on the light-client for dirty ledgers domain, Tas et al. [32] proposed the first such solution in the synchronous model. In that work, a number of nodes, which are called $full\ nodes$, are in charge of updating the state of the system and providing state commitments to the light clients, proving their validity through an interactive game (bisection game). Unlike this solution we propose a light-client construction that is non-interactive, third-party verifiable (i.e., if a node is convinced it can convince other nodes as well) and works in the asynchronous model. This however, comes at the cost that we require an honest majority of executors that cannot be bribed or adaptively corrupted (for the probabilistic solution). We can also employ a fall-back mode in the protocol where any client not happy with the assumption above, but who assumes synchrony waits for any of the elected executors to provide a fraud-proof [10] or ask an honest full-node for the correctness of a state-commitment through bisection games. Both approaches have an honest minority assumption which will always be true with overwhelming probability. As a result, our proposal can easily be adapted to a flexible model [26] for heterogeneous clients.

3.2.2 Model and Assumptions

Communication model: We assume an asynchronous environment, where any message sent can be delayed for an unspecified, but finite, amount of time. The link between every two honest nodes is reliable, namely when an honest node sends a message to another honest node, the message will eventually arrive.

Cryptographic Primitives: We use κ to denote the security parameter. We assume the adversary is computationally bounded, the communication channels are cryptographically secure, and

the existence of hash functions, signatures, and encryption schemes. We use a computationally hiding and perfectly binding commitment scheme: $(Compute_{cmt}, Verify_{cmt})$. We require the commitment scheme to be deterministic and provide inclusion proofs, e.g., it can be a Merkle tree. Moreover, users employ a Verifiable Random function (VRF) [28].
Permissioned setting: We consider a fixed number of n nodes with their public keys known to every participant in the network. A genesis block G which describes the initial state of the system is provided both to the executors and to the clients. The adversary is static and can corrupt up to $f \leq (1-\epsilon)\frac{n}{2}$ ($\epsilon > 0$ is a constant) nodes in a Byzantine fashion before the protocol starts.
Proof of stake: With the term node we refer to each identity that has an account on the system. The point of reference in the proof-of-stake system is a unit coin, which is the smallest amount of money existing. Each coin is a unique string linked to its owner. We assume each node is equipped with a private-public key pair. A genesis block G which contains the initial stake distribution is accessible to all nodes. The stake distribution is dynamic, namely the coins might change hands over time. We assume that the total amount of stake is fixed and equal to W in every round. The adversary is static and can corrupt a portion of the stake holders holding at most f coins, such that $f \leq (1-\epsilon)\frac{W}{2}$ where ϵ is a constant, in a Byzantine fashion.

3.2.3 Problem Definition

We formulate the State Machine Replication (SMR) problem by diving it into an ordering and an execution layer. Solutions for the ordering layer include Blockchain protocols such as Byzantine Atomic Broadcast (BAB) [34, 15, 14, 24], in which the nodes only agree on the order of the blocks without executing them. Our protocols are solutions for the execution layer.
State Machine Replication (SMR): A state machine consists of a set of state variables that encode its current state. External identities, users of the system, can issue commands to the state machine. The state machine executes the commands sequentially using a transition function to update the state of the system. Furthermore, the state machine might generate an output after executing each command. To provide fault-tolerant behavior, the state machine replicates in multiple copies. An SMR protocol aims to maintain synchronization between the replicas. In this paper, we illustrate that an SMR solution can be a composition of a protocol Π_1 for the *ordering layer* and a protocol Π_2 for the *execution layer*. Below, we define the *ordering* and the *execution* layers.
Ordering layer: Consider a number of nodes, some of which can be adversarial, receiving transactions from external identities. The nodes organize the transactions in blocks. Furthermore, they employ a protocol Π_1 to agree on an order of the blocks. Each node i commits locally to a finalized ledger of blocks. We denote the ledger to which node i commits in the round r by T_r^i. The output of the ordering protocol, i.e. the order of the blocks in which the nodes reach, is a ledger $T = b_0 \leftarrow b_1 \leftarrow ... \leftarrow b_i$. We introduce the properties that an ordering protocol must satisfy:

- *O-Safety*: There is no round r for which exist two honest nodes i, j s.t. $T_r^i \neq T_r^j$.
- *O-Liveness*: If an honest node receives an input tx, then all honest nodes will eventually include tx in a block of their local ledger.

Execution Layer: Consider a number of nodes where some of them can be adversarial. Moreover, consider the ledger of blocks $T = b_0 \leftarrow b_1 \leftarrow ... \leftarrow b_i$ output by the ordering layer, accessible to everyone. Each block might contain invalid transactions. The validity of a transaction depends on the logic of the application. The nodes are responsible for applying only the valid transactions within the blocks committed to the ledger T. The invalid transactions within the blocks are disregarded. Each node updates the state of the system. We denote the state of the system in the round r according to node's i view by S_r^i.

State: As a blockchain state, we denote a structure keeping track of each user's possessions. The content of the state depends on the type of transactions committed to the ledger. For instance, in Ethereum, the state captures the balance accounts of the users, while in Bitcoin the UTXO model is adopted. Furthermore, the state can contain fragments of code, e.g., smart contracts.

Ideal functionality Π*:* We illustrate the correctness of the state of the system by introducing an ideal functionality Π. The functionality Π receives as input the ledger $T = b_0 \leftarrow b_1 \leftarrow ... \leftarrow b_i$ which is an output by the ordering layer. Π updates the state by applying all (and only) the valid transactions within the blocks committed to the ledger T. We denote the state of the system stored by Π for the round r by S_r^*. The initial state of the system S_0^* equal to the genesis block, $S_0^* = G$. To update the state in each round, Π uses the deterministic transition function $apply$. The inputs are the state of the previous round and the block to be executed in the current round. More specifically, in the round r, $S_r^* \leftarrow apply(b_r, S_{r-1}^*) = S_{r,len(b_r)}$ where

$$S_{r,j} = \begin{cases} S_{r-1}^* & \text{if } j = 0 \\ \text{apply\_tx}(S_{r,j-1}, tx_j) & \text{if } 1 \leq j \leq len(b_r) \end{cases} \quad \text{and } b_r = [tx_1, ..., tx_{len(b_r)}].$$

The state applied to the function *apply_tx* remains unchanged when the input tx is an invalid transaction, namely *apply_tx(S,tx)=S*. Therefore, for the ledger T, there exists a unique sequence of states $S_0^*, S_1^*, ..., S_i^*$ defined by the state transition function above.

In practice, nodes can employ any execution engine M that simulates the ideal functionality Π. When receiving as inputs the correct state of r and the block $r + 1$, the engine M outputs the correct state of $r + 1$.

An execution layer guarantees that the honest nodes simulate the ideal functionality Π. We proceed with defining the properties of an execution layer:

- *E-Safety*: There is no round r for which exists an honest node i that commits on a state S_r^i s.t. $S_r^i \neq S_r^*$.
- *E-Liveness*: For any round r where an honest node i commits a state S_r^i, there exists a round $r' > r$ where node i eventually commits a state $S_{r'}^i$ s.t. $S_r^i \neq S_{r'}^i$.

Since nodes keep updating their state without deviating from the ideal functionality Π, an execution protocol is *Censorship Resistant*, namely it satisfies the following property:

- *Censorship Resistance:* Every valid transaction tx committed to the ledger T will eventually be applied in the state.

Note that the liveness property ensures only that each honest node will eventually update its state. Since not all nodes execute for every round essentially, we do not require that the honest nodes update their states in the same rounds.

State Machine Replication: Finally, we formulate the SMR problem on top of the ordering and execution layers. More specifically, transactions issued by external identities constitute the input of the SMR. An SMR protocol consists of an ordering layer protocol Π_1 and an execution layer protocol Π_2 satisfying the properties *O-Safety*, *O-Liveness* and *E-Safety*, *E-Liveness* respectively. Nodes participating in those protocols may or may not be the same. The output of Π_1 which is a ledger of blocks T is the input of Π_2. The output of the machine is the output of Π_2, namely an ever-growing state sequence $S_0, S_1, S_2, ..., S_i$.
Light Client. Consider an execution layer Π_e with input a ledger T and the average size of the state that the ideal functionality Π outputs in any round $|S|$. The execution layer supports light client constructions. Light clients request succinct proofs from the participating nodes to learn desired information about the state of the system. We capture this idea by defining the *state proof* certificates.

- A *state proof* π_S for the round r is a succinct proof indicating that the state S is the correct state of the round r. Proof π_S is correct if and only if $S = S_r^*$.
- A *state proof* π for the round r is succinct if it contains asymptotically less data than the history of states, namely if $\frac{len(\pi)}{r|S|} = o(1)$

Assume that the light client lc_i receives a *state proof* π_S for the round r without necessarily receiving the state S. Lc_i evaluates whether the proof is correct, in its perception, using a predicate $accept_{lc_i}(\pi_S, r)$ which yields either True or False. The light client lc_i accepts π_S if and only if $accept_{lc_i}(\pi_S, r) = True$. The properties which a light client execution layer protocol must satisfy are the following:

- *LC-Safety*: There is no round r for which exists a light client lc_i that receives a proof π_S for the round r s.t. $accept_{lc_i}(\pi_S, r) = True$ and $S \neq S_r^*$.
- *LC-Liveness*: A light client bootstrapping in the round r will eventually receive a proof $\pi_{S'}$ for a round $r^*, r^* \geq r$ s.t. $accept_{lc_i}(\pi_{S'}, r^*) = True$.

Additional Assumptions: We assume the existence of an underlying ledger T as an output of an *ordering layer* Π that satisfies the properties of *O-Safety*, *O-Liveness*. The ledger T is accessible to every node. Furthermore, we assume that there are no duplicate transactions committed to T. Finally, we assume that for each round a random seed is provided by the dirty ledger, similarly to Algorand [22], or DAG-based BFT protocols [24, 16, 23].

3.3 Overview of the Protocols

In our proposed protocols, executors update the state of the system in a distributed fashion. We decompose the protocols in two phases, an *election phase* and a *voting phase*. The *election phase* will select a set of executors for every round. Then, in the *voting phase* elected executors for the round compute and broadcast their signed state commitments. The *voting phase* outputs *valid* state commitments, as defined:

- A state commitment is considered to be valid if and only if either it is signed by at least one honest node or it is the genesis block.

The goal is to ensure that only *correct* state commitments, defined below, will become *valid*.

- A state commitment cmt is the *correct* state commitment of round r if and only if $cmt = compute_{cmt}(S_r^*)$, where S_r^* is the state of round r defined by the ideal functionality of the execution layer introduced in Section 3.2.

There are two challenges when solving the problem. The first is to ensure that there is provably at least one honest node that has voted a state commitment to guarantee its validity. The second is to ensure that when elected nodes enter the *voting phase*, they have the state of the previous round available. Below we explain how we tackle these challenges for different settings.

Permissioned and Deterministic. First, we present a straightforward deterministic protocol for the permissioned settings to lay the foundation of our other solutions. We consider a total number of $n = 2f + 1$ executors (instead of $n > 3f$), with f executors corrupted by a static adversary. Every node executes for each round, i.e., each executor starts from the genesis block and updates the state by applying the valid transactions of the dirty ledger. For every round, executors compute, sign, and broadcast the corresponding state commitment. A state commitment is valid if signed by at least $f + 1$ nodes so that at least one honest node is included.

Probabilistic solutions. The straightforward deterministic solution requires every executor to run for every round, which is not scalable. For better scalability, we propose probabilistic protocols for the permissioned and the Proof-of-Stake settings. We assume up to $f \leq (1 - \epsilon)\frac{n}{2}$ executors can be corrupted by a static adversary, where ϵ is some constant. Our protocols guarantee the validity of a state commitment by requiring a threshold of executors to sign the state commitment. To ensure safety, the number of adversarial nodes executing in each round must be less than this threshold. To ensure liveness, in each round, there must be enough honest nodes executing to form a valid state commitment. We set the threshold for the valid state commitment to be $1/2$ of the number of elected executors, and demonstrate that the aforementioned property is satisfied with a overwhelming probability in the security parameter by electing only a logarithmic number of nodes per round.

Vertical vs Horizontal Sampling. The straightforward probabilistic solution is to elect a *committee* of poly-logarithmic size per round who broadcasts signed state commitments. A state commitment is considered valid if it is signed by at least half of the committee members. We call this approach *vertical sampling*. Each node is elected on average once per $O(\frac{n}{polylogn})$ rounds and executes for only the respective rounds. Instead, we adopt an approach we call *Horizontal Sampling*, in which only expected constant number (e.g. one) of nodes are elected per round. In that solution, every node is elected on average every n rounds and executes for $O(polylogn)$ rounds. In both cases the cost per block execution is $O(\frac{polylogn}{n})$. However, since nodes update their execution states in a distributed fashion, elected nodes may need to retrieve the previous execution state from other nodes in order to execute the current round, which incurs high communication overhead. In *Horizontal Sampling*, in comparison to the vertical sampling, nodes request the state less frequently, resulting in a more scalable solution.

Permissioned and Randomized. First, we present the *Horizontal Sampling* protocol for the permissioned settings. During the *election phase*, each executor computes the VRF locally

in each round. Only one node on average is elected per round. The elected node starts from the state of the previous round, computes and broadcasts state commitments for the following $O(polylogn)$ rounds. Hence, with only one executor elected per round, a poly-logarithmic number of nodes will vote for each round. State commitments signed by at least half of the elected nodes are considered valid.

Proof-of-stake (PoS). We then extend the *Horizontal Sampling* protocol for the Proof-of-Stake settings. In the permissioned settings, each node computes a VRF for the *election phase*. In PoS, the adversary can create numerous accounts to increase the probability of being elected. To make the protocol Sybil Resistant, each node's election probability is proportional to its stake. Concretely, nodes compute the VRF for all of their coins in the *election phase*. In the *voting phase*, elected nodes compute and broadcast their signed state commitments, as in the permissioned protocol.

An extra challenge in the PoS protocol is that the stake distribution changes over time. In every round, each node keeps track of its own stake and only the elected nodes execute the state. Therefore, elected nodes must prove the ownership of elected coins to the rest of the nodes. To this end, they construct and broadcast inclusion proofs along with their signed state commitments.

State availability. In the probabilistic protocols, not all nodes execute for every round to acquire the respective the state of every round. For liveness, our protocol must guarantee state availability, i.e., any node is able to acquire the state of the previous round when it executes the current round. Since any valid state commitment is signed by at least one honest node, the corresponding state will eventually be available to any node requesting it.

Light clients. Lastly, we introduce a non-interactive light client construction for our protocols. We assume that at any given time, each light client is connected to at least one honest executor. Briefly, a non-interactive light client can learn information about the state of the system after receiving a valid state commitment from an executor, along with an inclusion proof (e.g. Merkle proof).

Chapter 4

Protocols

In this section, we present our asynchronous execution layer protocols on top of an underlying dirty ledger. First, in the permissioned settings, we present the deterministic protocol 5 demonstrating how to construct verifiable certificates that correspond to the correct state, i.e., the valid state commitments. The deterministic protocol suggests that a majority of honest nodes is a necessary and sufficient condition to construct valid state commitments. However, every node executes for every round resulting in cost per block execution O(1). Next, we define a probabilistic scalable protocol called Horizontal Sampling, where in every round we select only a poly-logarithmic number of nodes to execute so that the majority of them are honest with overwhelming probability. We start with the permissioned settings in Section 4.2 and then extend the Horizontal Sampling protocol to the proof-of-stake settings in Section 4.3.

4.1 Deterministic Protocol

The Deterministic protocol is described in Algorithm 5. Every node is responsible for executing in each round. More specifically, every node downloads the data committed to the ledger starting from the genesis block and applies it sequentially via the execution machine M, to acquire the execution state of each round. In every round, the nodes compute and sign the respective state commitment. Then, they broadcast the state commitment to the rest of the nodes. This process is illustrated in Algorithm 5 (Procedure Execute lines: 5-11). The remaining nodes accept signed state commitments once they have verified the sender's signature (Algorithm 5 Predicate AcceptCommitment, lines: 3-4). A state commitment is valid when it is signed by at least $f + 1$ nodes.

4.2 Horizontal Sampling

In the deterministic protocol, all nodes execute in every round and broadcast their state commitments. Now, we proceed with building an efficient probabilistic protocol, called *Horizontal Sampling*, illustrated in Algorithm 6. We assume up to $f \leq (1 - \epsilon)\frac{n}{2}$ executors can be corrupted by a static adversary where ϵ is some constant, and we choose the security parameter $\kappa = O(\log^2 n)$ for this section. Nodes first download the genesis block G which holds the initial state. In each round, every node checks whether it is elected (Algorithm 6, line 28). Elected

Algorithm 5: Deterministic execution protocol: Node p_i with public key pk_i and secret key sk_i

1 $state(0) \leftarrow G, r_{cur} \leftarrow 1$
2 $threshold \leftarrow f + 1, state\_com \leftarrow \{\}$
`/* accept the state commitment` cmt `from the node with public key` pk_j `if the signature` σ `is valid */`
3 **Predicate** *AcceptCommitment*(cmt, r, σ, pk_j) **:**
4 ⌊ **return** *Verify*$(\sigma, pk_j, cmt||r)$
`/* executing for round r: applying the data committed to the ledger to update the state, computing and broadcasting the state commitment to everyone */`
5 **Procedure** *Execute*(r) **:**
6 **download** $data(r)$
7 $\texttt{state}(r) \leftarrow apply(\texttt{state}(r-1), data(r))$
8 $cmt \leftarrow Compute_{cmt}(\texttt{state}(r))$
9 $\sigma \leftarrow Sign(cmt||r, pk_i, sk_i)$
10 $state\_com[(r, cmt)].add((\sigma, pk_i))$
11 ⌊ Send ("state commitment", cmt, r, σ) to all nodes
`/* Main loop. */`
12 **while** *True* **do**
13 *Execute*(r_{cur})
14 ⌊ $r_{cur} \leftarrow r_{cur} + 1$
15 **Upon** *receiving("state commitment", cmt, r,* σ*) from the node with public key* pk_j *for the first time in the round* r **do :**
16 **if** *AcceptCommitment(*cmt, r, σ, pk_j*)* **then**
17 ⌊ $state\_com[(r, cmt)].add((\sigma, pk_j))$

nodes propose state commitments during the *voting phase*. The *voting phase* outputs valid state commitments, which are state commitments signed by enough executors.

Election phase: In each round, every node computes the VRF using its private key, the round number, and the corresponding random seed. This computation returns two values, a hash value of length $|h|$ and a proof of authenticity certifying this hash value (Algorithm 6, line 27). We refer to this proof as the *proof of election* of the leaders. All nodes with hash value in round r of less than $X_r = \begin{cases} \kappa\frac{2^{|h|}}{n}, & \text{if } r = 1 \\ \frac{2^{|h|}}{n}, & \text{if } r > 1 \end{cases}$ are elected (Algorithm 6, line 28). In that way, in the first round there will be expected κ elected nodes constituting the *bootstrap committee*, while for $r > 1$ there will be only one node in expectation, which is called the leader.

Validity of a commitment: For the first κ rounds only the members of the *bootstrap committee* are voting. For any round $r \geq \kappa+1$ all the elected nodes in the interval $[r-\kappa+1, r]$ compute the state commitments. In Lemma 7.2.4, we prove that the *bootstrap committee* consists of at least $\frac{\kappa}{2}$ honest nodes and at most $\frac{\kappa}{2} - 1$ adversarial nodes with overwhelming probability in n (we choose $\kappa = O(\log^2 n)$). The same property holds for the elected nodes in any interval of κ consecutive rounds according to Lemma 7.2.5. As a result, in each round, at least $\frac{\kappa}{2}$ honest and at most $\frac{\kappa}{2} - 1$ adversarial nodes will be responsible for voting. Therefore, a state commitment corresponding to a round r can be considered as valid if it is signed by at least $\frac{\kappa}{2}$ nodes among

Algorithm 6: Horizontal Sampling: Node p_i with public key pk_i and secret key sk_i

$state(0) \leftarrow G$ // genesis block
$\texttt{threshold} \leftarrow \frac{\kappa}{2}, state\_com \leftarrow \{\}$
$r_{cur} \leftarrow 1$ // current round
/* threshold for the election process */
Procedure *Target*(r) **:**
    return $\frac{2^{|h|}}{n}\kappa$ **if** $r = 1$, **else return** $\frac{2^{|h|}}{n}$
/* verify the election proof with public key p_k in the round r */
Predicate *TimeToExecute*(p_k, r, u, π) **:**
    return *VerifyVRF*$_{p_k}(v, \pi, seed_r||r) \wedge v \leq$ *Target*(r)
/* check whether cmt comes from a valid leader of round r_l that is responsible for executing in round r */
Predicate *AcceptCommitment*$(p_k, r_l, u, \pi, r, \sigma, cmt)$ **:**
    return
      $\neg(r_l > 1 \wedge r \leq \kappa) \wedge (r_l \leq r \leq r_l + \kappa - 1) \wedge$ *Verify*$(\sigma, p_k, cmt||r) \wedge$ *TimeToExecute*(p_k, r_l, u, π)
/* acquiring the state of round r */
Procedure *AcquireState*(r) **:**
    Wait until $\exists(cmt, r)\ s.t. |state\_com[(cmt, r)]| \geq threshold$
    if $\texttt{state}(r) = null$ **then**
        request $\texttt{state}(r)$
        wait until receiving $state$ s.t. $Compute_{cmt}(state) = cmt$
        $\texttt{state}(r) \leftarrow state$
/* compute and broadcast the signed state commitments for all the intermediate rounds within the interval $[r_l, r_l + \kappa - 1]$ */
Procedure *Execute*$(r_l, (v, \pi))$ **:**
    AcquireState$(r_l - 1)$ if $r_l > 1$
    for $r = r_l, ..., r_l + \kappa - 1$ **do**
        download $data(r)$ // data within the block with height r
        $\texttt{state}(r) \leftarrow apply(\texttt{state}(r-1), data(r))$
        continue if $r_l > 1 \wedge r \leq \kappa$ // only bootstrap committee votes
        $cmt \leftarrow$ *Compute*$_{cmt}(\texttt{state}(r))$
        $\sigma \leftarrow Sign(cmt||r, pk_i, sk_i)$
        $state\_com[(r, cmt)].add((pk_i, r_l, v, \pi, \sigma))$
        Send *("state cmt"*$, cmt, r_l, r, \sigma, v, \pi)$ to all nodes
/* Main loop, run leader election for each round */
while *True* **do**
    $(v, \pi) \leftarrow VRF_{sk}(seed_{r_{cur}}||r_{cur})$
    if $v \leq$ *Target*(r_{cur}) **then**
        Execute$(r_{cur}, (v, \pi))$
    $r_{cur} \leftarrow r_{cur} + 1$
Upon *receiving("state cmt"*$, cmt, r_l, r, \sigma, u, \pi)$ *from the node with public key* pk_j *for the first time for round* r_l **do :**
    if *AcceptCommitment*$(pk_j, r_l, u, \pi, r, \sigma, cmt)$ **then**
        $state\_com[(r, cmt)].add((pk_j, r_l, u, \pi, \sigma))$

those that are elected to execute during the interval of rounds $[max(1, r-\kappa+1), r]$ or if it is the genesis block. In figure 4.1, on the left side we present an example of the leaders' votes in the interval $[r, r+3]$ where the malicious leader L_{r+2} votes for incorrect state commitment for rounds $r+2, r+3$; on the right side we present the resulting fork in the round $r+2$, where there are less than $\frac{\kappa}{2}$ votes for the incorrect state commitment coming from the adversarial leaders in the interval $[r-\kappa+3, r+2]$, and at least $\frac{\kappa}{2}$ votes for the correct state commitment coming from the honest leaders in the same interval.

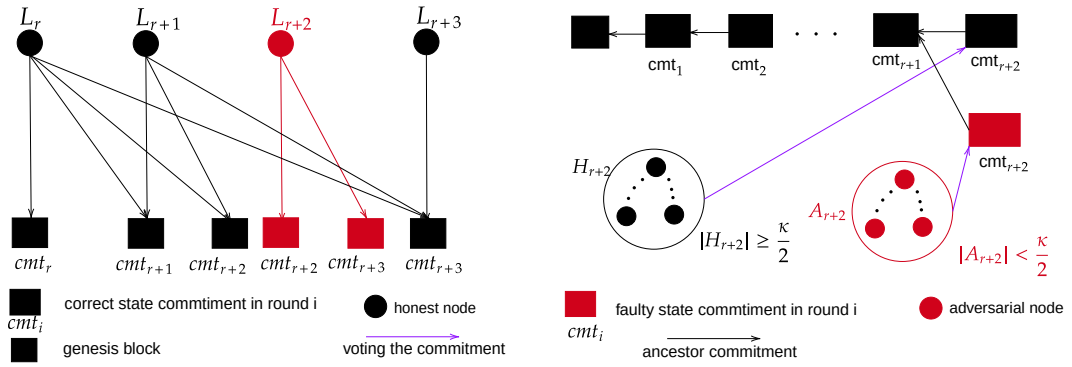


Figure 4.1: The left figure illustrates the elected leaders' votes in the round interval $[r, r+3]$, resulting in the fork in the chain of the proposed state commitments illustrated on the right side. The set H_{r+2} (or A_{r+2}) consists of the votes of the honest (or adversarial) elected leaders in the interval $[r-\kappa+3, r+2]$.

Voting phase: For the first κ rounds, the *bootstrap committee* members form the respective valid state commitments. To update the state, they apply the transactions committed to the ledger for all these rounds starting from the genesis block. For every round, they compute and broadcast their signed state commitments along with their proof of election.

Now consider node p_i, an elected leader in some round $r \geq 2$ during the *voting phase* (Algorithm 6, Procedure Execute). First, p_i waits until witnessing a valid state commitment for the round $r-1$. After receiving the valid state, the leader acquires the corresponding state. If the state is not available from a previous execution, p_i requests it from all the nodes that have signed the commitment (Algorithm 6, lines 13-14) (more on data availability in section 6). Then, p_i downloads the data committed to the ledger for the intermediate rounds and applies it sequentially to obtain the state of the round $r+\kappa-1$. For each round, it constructs and signs the respective state commitment. Finally, p_i broadcasts the signed state commitments along with the proof of its election to the rest of the nodes. We note again that only *bootstrap committee* members vote for the first κ rounds (Algorithm 6 line 21). The rest of the nodes accept the received commitments only after confirming $p_i's$ signature and proof of election (Algorithm 6, lines 8-9).

4.3 Proof-of-stake settings

Now we extend the *Horizontal Sampling* protocol to the proof-of-stake setting. Participating nodes have accounts holding stake/coins, and we use W to denote the total amount of the stake in the system. New nodes can dynamically join the system, and we demonstrate bootstrapping in Algorithm 9. We assume up to $f \leq (1-\epsilon)\frac{W}{2}$ stake can be corrupted by a static adversary, where ϵ is some constant, and we choose the security parameter $\kappa = O(\log^2 W)$ for this section.

First, all nodes download the genesis block G which contains the initial stake distribution. The stake distribution can change over time. The execution protocol consists of the Algorithms 7, 8. More specifically, we decompose the protocol into the following phases. In each round, every node participates in the *election phase* to check whether any of its coins is elected (Algorithm 7, Procedure Election). During the *voting phase*, nodes with at least one elected coin compute state commitments to form the respective valid state commitments, like in the permissioned protocol.

Tracking wealth: The stake distribution changes over time and the nodes do not necessarily acquire the execution state of each round. Every node joins the system with a specific stake, and it triggers transactions, e.g., to pay a user, or it receives transactions, e.g., it is paid by a user. Hence the challenge for a node to update its stake is to check whether the transactions it receives are successful or not. In our protocol, the node requests a *proof of payment* certificate from the payer, (see section 5), to verify that its state has changed as expected and therefore the transaction was successful.

Election phase: In each round, every node computes the VRF using its owned coins and the randomness seed coming from the dirty ledger to see whether it is elected and generate the *proofs of election* for the elected coins. Similarly to the permissioned protocol, the PoS protocol elects a *bootstrap committee* for the first round, and elects on average one coin per round for every round $r > 1$. To keep the threshold of a valid state commitment identical for every round, only the *bootstrap committee* members are voting for the first κ rounds, while for $r \geq 2$ the owner of an elected coin in round r can vote for every round in the interval $[max(r, \kappa+1), r+\kappa-1]$. A state commitment for the round r is valid if it is signed by the owners of at least $\frac{\kappa}{2}$ of the elected coins during the interval of rounds $[max(1, r-t+1), r]$ or if it is the genesis block.

Proof of ownership: Since nodes track only their own stake, the elected nodes must prove that they own the elected coins. Hence, they provide inclusion proofs for their elected coins using the valid state commitment of the previous round, e.g., the commitment would be the Merkle root in a Merkle proof. We call these certificates *proofs of ownership* and the corresponding state commitment *parent commitment*. To be able to verify the *proofs of payment* in order to track its stake, and to compute the *proofs of ownership* in case of election, each node waits for the valid state commitment of the previous round before participating in the election phase.

Voting phase: The *voting phase* is similar to the permissioned protocol. First, the *bootstrap committee* members compute and broadcast their signed state commitments for every round $r \leq \kappa$ to form the respective valid state commitment. Then, every node with an elected coin in the round r, starts from the state corresponding to the valid state commitment of the round $r-1$. Moreover, the node uses the valid state commitment to construct the *proof of ownership* for its elected coin. Finally, the elected node computes and broadcasts the signed state commitments

for all the intermediate rounds along with the *proof of election* and the *proof of ownership* in the round r.

Nodes accept the state commitments proposed by the elected nodes only after verifying the *proof of election* and the *proof of ownership* certificates (Algorithm 8 lines 27-30). When receiving an inclusion proof with a *parent commitment* which is not valid yet, they store all the certificates in a local data structure (Algorithm 7 lines 13-15). When a state commitment becomes valid in the view of an honest node p_i, the node takes into account all the votes for which it is a *parent commitment* (Algorithm 7 lines: 21-24).

Bootstrapping: Consider Bob, a node that wishes to join the network in the round r. We assume that Bob is connected to at least one honest executor. Bob has received from many nodes a data structure called $chain$ that contains the state commitments signed by the elected nodes along with the respective certificates (signatures, proofs of election, and proofs of ownership) for each round.

Bob downloads the genesis block G first. For each $chain$, Bob applies Algorithm 9 to evaluate whether it is the correct one. For the first round, *Bob* verifies only the *proofs of election* of the *bootstrap committee* members since the initial stake distribution is contained in G. Then, for each vote up to round r, he verifies the signatures, the *proof of ownership*, and the *proof of election* of the elected nodes. When Bob receives the correct $chain$, it acquires the last valid state commitment in the $chain$ and requests the corresponding state (Section 6). As an example, in figure 4.2 we present the correct chain up to the round h.

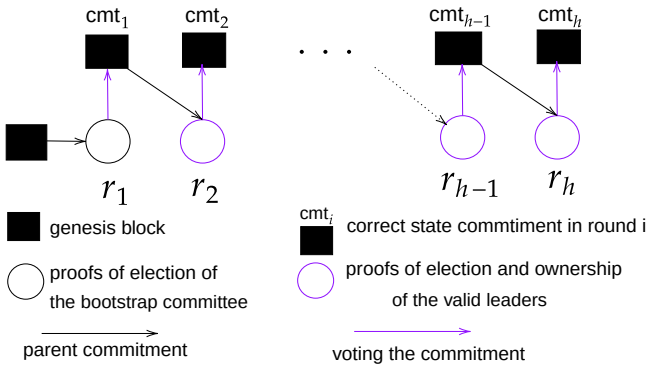


Figure 4.2: A correct $chain$ up to the round h. Only nodes elected during the interval of rounds $[i, i + \kappa - 1]$ are allowed to vote for a commitment in the round i.

.

Algorithm 7: Functions for the election phase: node p_i with pair of keys (pk_i, sk_i)

```
   /* threshold for the election process                                        */
1  Procedure Target(r) :
2      return (2^|h| / n) κ if r = 1, else return 2^|h| / n
   /* p_i evaluates whether a subset of its coins is elected in the round r,
      returning a list with the respective certificates                         */
3  Procedure Election(r, parent_cmt) :
4      elected_coins ← {}
5      for each coin that p_i owns in the round r do
6          (v, π) ← VRF_{sk_i}(coin||seed_r)
7          if v ≤ Target(r) then
8              π_m ← null if r = 1, else π_m ← InclusionProof(parent_cmt, p_i, coin)
                 elected_coins.add((pk_i, r, coin, v, π, π_m, parent_cmt))
9      return elected_coins
   /* verify the proofs of election and ownership coming from a valid leader that
      can vote for the round r                                                  */
10 Procedure VerifyVotes(r, votes) :
11     valid_votes ← {}
12     for vote ← (pk_j, r_l, coin, u, π, π_m, parent_cmt) ∈ votes do
13         if |state_com[(r_l − 1, parent_cmt)]| < κ/2 then
14             tmp_certificates[(parent_cmt, r_l − 1)].add(cmt_r, r, votes)
15             continue // parent_cmt is not valid yet
16         if (r_l = 1 ∧ pk_j owns coin ∧ r ≤ κ ∧ VerifyVRF_{pk_j}(u, π, coin||seed_1) ∧ u ≤
             Target(1)) ∨ (r_l > 1 ∧ r > κ ∧ r_l ≤ r ≤ r_l + κ − 1 ∧ Verify_{InclusionProof}(π_m, parent_cmt)) ∧
             VerifyVRF_{pk_j}(u, π, coin||seed_{r_l}) ∧ u ≤ Target(r_l)) then
17             valid_votes.add(vote)
18     return valid_votes
   /* include the valid votes for cmt; if cmt becomes valid, update the votes
      for which cmt is the parent commitment                                    */
19 Procedure UpdateVotes(votes, cmt, r) :
20     ∀vote ∈ votes : state_com[(cmt, r)].add(vote)
21     if |state_com[(cmt, r)]| ≥ threshold then
22         for (cmt_{r'}, r', certificates) ∈ tmp_certificates[(cmt, r)] do
23             induced_votes ← VerifyVotes(r', certificates)
24             UpdateVotes(induced_votes, cmt_{r'}, r') if induced_votes ≠ null
```

Algorithm 8: Functions for the voting phase: node p_i with pair of keys (pk_i, sk_i)

$state(0) \leftarrow G, state\_com \leftarrow \{\}, tmp\_certificates \leftarrow \{\}, elected\_coins \leftarrow \{\}, succesful\_rounds \leftarrow \{\}$
$r_{cur} \leftarrow 1$, `threshold` $\leftarrow \frac{\kappa}{2}$
$parent_{cmt} \leftarrow G$ `// valid state commitment of round` $r-1$
`/* acquire the state of the round` r `*/`
Procedure *AcquireState(r)* **:**
  if $state(r) = null$ **then**
    $cmt \leftarrow cmt'$ *s.t.* $state\_com[(cmt', r)] \geq threshold$
    request `state`(r)
    wait until receiving $state$ s.t. $Compute_{cmt}(state) = cmt$
    `state`$(r) \leftarrow state$
`/* node` p_i `computes and broadcasts its state commitments for the rounds` $[r_l, r_l + \kappa - 1]$ `to everyone along with the proofs of election and ownership of its coins in the round` r_l `*/`
Procedure *Execute*$(r_l, elected\_coins)$ **:**
  AcquireState$(r_l - 1)$ if $r_l > 1$
  for $r = r_l, ..., r_l + \kappa - 1$ **do**
    download $data(r)$
    `state`$(r) \leftarrow apply($`state`$(r-1), data(r))$
    continue if $r_l > 1 \wedge r \leq \kappa$ `// only the bootstrap committee votes`
    $cmt \leftarrow$ *Compute*$_{cmt}$(`state`$(r))$
    $\sigma \leftarrow Sign(cmt||r, pk_i, sk_i)$
    $\forall vote \in elected\_coins : state\_com[(r, cmt)].add(vote)$
    $Send($*"state cmt"*$, r_l, r, cmt_r, \sigma, elected\_coins)$ to all nodes
`/* Main loop.` `*/`
while *True* **do**
  Wait until $\exists(cmt, r_{cur} - 1)$ $s.t. |state\_com[(cmt, r_{cur} - 1)]| \geq threshold$ if $r_{cur} > 1$
  $parent_{cmt} \leftarrow cmt$ if $r_{cur} > 1$ `// else` $parent_{cmt} \leftarrow G$
  $elected\_coins \leftarrow Election(r_{cur}, parent_{cmt})$
  if $elected\_coins \neq null$ **then**
    Execute$(r_{cur}, elected_coins)$
  $r_{cur} \leftarrow r_{cur} + 1$
`/* Receiving` cmt `for round` r`, the respective certificates (proofs of election and ownership), and the signature` σ`.` `*/`
Upon *receiving("state cmt",* $r_l, r, cmt_r, \sigma, certificates$*) from node with public key* pk_j *for the first time for the round* r_l **do :**
  return if $Verify(\sigma, pk_j, cmt_r||r) = false$
  $valid\_votes \leftarrow$ *VerifyVotes*$(r, certificates)$
  UpdateVotes$(valid\_votes, cmt_r, r)$ **if** $valid\_votes \neq null$

Algorithm 9: Bootstrapping: user U_i

```
/* U_i aims to bootstrap in a round greater or equal than round r_e.          */
Procedure Bootstrap(chain, r_e) :
    r ← 1, state_cmt ← {}, valid_state_cmt ← {}
    while chain[r] ≠ null do
        for (node, cmt, σ, votes) ∈ chain[r] do
            continue if Verify(σ, node, cmt||r) = false
            valid_votes ← VerifyVotes(r, votes)
            if valid_votes ≠ null then
                ∀vote ∈ valid_votes : state_cmt[(cmt, r)].add(vote)
                if |state_cmt[(cmt, r)]| ≥ κ/2 then
                    valid_state_cmt[r] ← cmt
                    break
        return null if ∄cmt s.t. |state_cmt[(cmt, r)]| ≥ κ/2
        r ← r + 1
    return valid_state_cmt if r ≥ r_e
    return null
```

Notation	Description
n	total number of nodes in the permissioned settings
f	number of adversarial nodes (or coins held by adversarial nodes) in the permissioned settings (or in PoS)
W	total amount of stake in PoS
proof of election	proofs coming from the VRF computation of the elected nodes in the probabilistic protocols
proof of ownership with parent commitment cmt	inclusion proof with hash header cmt demonstrating that a node p_i owns a particular coin in PoS

Table 4.1: Summary of Terminologies

Chapter 5

Light clients

Once we have a system where executors can verify that a payment has been made, it is simple to transform it to the first non-interactive, asynchronous light-client for dirty ledgers. In this section, we demonstrate how a light client can learn the state of the system. First, we discuss how a light client can acquire and verify a *state proof*. Then, we use *state proofs* as a building block to prove that a change in the state occurred.

Assumptions: Each light client has access to the random seed for each round through the dirty ledger, in order to verify the leader election. In addition, each light client is connected to at least one honest executor. An executor uses a gossip protocol to obtain information necessary to react to a light client's requests, such as the state that corresponds to a valid state commitment.

Bootstrapping: Assume that the height of the dirty ledger equals h and a light client lc_i bootstraps in the round $r \leq h$. First, we illustrate how lc_i can verify a *state proof*. A validity proof of the state commitment corresponding to the state S constitutes the *state proof* π_S. The light client then chooses how to connect to the network. One option is to receive the corresponding state and derive the desired information after downloading and applying the data committed to the ledger on its own. Otherwise, lc_i can reconnect to the network whenever it needs a *proof of payment* certificate.

State Proof - Permissioned settings: To bootstrap in the round r, lc_i waits to receive a valid state commitment for some round greater than or equal to the round r. In the *Deterministic protocol*, lc_i verifies that a state commitment is signed by at least $f + 1$ nodes using the Predicate *AcceptStateProof* in Algorithm 10. In the *Horizontal Sampling protocol*, a valid state commitment in a round r' is voted by at least $\frac{\kappa}{2}$ elected leaders in the interval $[max(1, r' - \kappa + 1), r']$. Each leader's vote includes their signature and proof of election. Lc_i verifies this using the Predicate *AcceptStateProof* in Algorithm 11.

State Proof - Proof-of-stake settings: The light client bootstraps using Algorithm 9. In a nutshell, for each round, lc_i requires and verifies the signatures, the *proof of ownership*, and the *proof of election* coming from the owners of the elected coins that have voted for the valid state commitments.

Proofs of payment: We now demonstrate how to provide certificates for successful transactions. Consider Alice and Bob, two light clients using our system. Bob wishes to purchase a product from Alice, triggering a transaction that will be logged in the dirty ledger. Alice needs a proof that the payment is successful before providing the merchandise to Bob.

Assume that the transaction of Bob paying Alice is committed at round r. The certificate with which Bob proves that Alice's state is changed in round r is called *proof of payment*. More

Algorithm 10: Light Client protocol - Deterministic Protocol

```
threshold ← f + 1
/* check whether there are at least f + 1 signatures for cmt in
   Σ                                                          */
Predicate AcceptStateProof(cmt, r, Σ) :
    Remove duplicates in Σ
    return |AcceptCommitment(cmt, r, σ, p_j) : (σ, p_j) ∈ Σ| ≥ threshold
Predicate PaymentProof(cmt, r, Σ, π_inclusion_proof) :
    return AcceptStateProof (cmt, r, Σ)∧ state change has occurred according to
     π_inclusion_proof
```

Algorithm 11: Light Client protocol - Horizontal Sampling

```
threshold ← κ/2
/* check whether there are at least κ/2 signatures for cmt by
   leaders of rounds [r − t + 1, r] in Σ                      */
Predicate AcceptStateProof(cmt, r, Σ)) :
    Remove duplicates in Σ
    return
     |AcceptCommitment(p_k, r_l, u, π, r, σ, cmt) : (p_k, r_l, u, π, σ) ∈ Σ| ≥ threshold
Predicate PaymentProof(cmt, r, Σ, π_inclusion_proof ) :
    return AcceptStateProof(cmt, r, Σ)∧ state change occurred according to the
     π_inclusion_proof
```

specifically, the certificate constitutes of a valid state commitment for any round greater than r and a short inclusion proof (e.g. a Merkle proof) indicating Alice's new state. Alice uses the Predicate PaymentProof in Algorithm 11 to verify first the validity of the state commitment and then the inclusion proof, using the valid state commitment, to extract her new state. If the transaction is successful, Alice's state is changed during this interval.

Proofs of being a payer: An issue which we call *proof of being a payer* arises. Assume that Alice consents to sell the same type of product to Bob and Eve. Consider the transactions tx_1, tx_2 which are supposed to be Alice's payment from Bob and Eve respectively. Assume that both transactions are committed to the block with height r, and tx_1 is valid while tx_2 is invalid. Now, Bob and Eve in turn provide a *proof of payment* certificate to Alice. Alice cannot distinguish whether it is Bob or Eve that has paid her.

To deal with that problem, executors compute and sign a state commitment after applying each transaction committed to the block for which they execute. For instance, consider the round r, the block $B_r = [tx_1, tx_2, ..., tx_i, ..., tx_{len(B)}]$ and S^*_{r-1} the state of the round $r-1$. Executors must compute and broadcast the state commitments $cmt_{r,i} \leftarrow compute_{cmt}(S_{r,i}), i = 1, ..., len(B)$, where $S_{r,i} \leftarrow apply(S^*_{r-1}, [tx_1, ..., tx_i])$ and $cmt_{r,0} = compute_{cmt}(S^*_{r-1})$. In that way, they form valid state commitments after applying each transaction. Now, Bob can

provide to Alice the valid state commitments $cmt_{r,0}$ and $cmt_{r,1}$ along with inclusion proofs to extract her state. Alice in turn will verify that her state has changed in that round and that tx_1 was successful.

For instance, in [32], *full nodes* organize this data into a binary Merkle tree. Each transaction is contained in a leaf of that tree accompanied with the state commitment that occurs by either applying the transaction, if it is valid, or neglecting it otherwise. According to our protocols, the executors must agree on each leaf of the Merkle tree.

Chapter 6

Data availability

State availability: Nodes responsible for executing in a particular round need to acquire first the state of a previous round. It is also required by the *Proof of payment* and bootstrapping in the proof-of-stake settings (section 4.3).

We first let the executors store every state they executed. In all of the proposed protocols, each valid state commitment is signed by at least one honest node which has stored the state (the argument holds with overwhelming probability in the probabilistic protocols). An executor requests the state that corresponds to a *valid* state commitment from all the nodes that have signed the respective state commitment. The honest node that has signed the state commitment will eventually provide it to the executor. The executor will verify that the state indeed corresponds to the valid state commitment.

Certificate availability: To support bootstrapping protocols, executors store the certificates related to the valid state commitments. In the Deterministic protocol, they only store the signed state commitments (Algorithm 5 lines: 10, 17). In the Horizontal Sampling protocol, executors store the signed valid state commitments along with the leaders' proofs of election (Algorithm 6 lines: 24, 33), and in the proof-of-stake settings, they additionally keep the *proofs of ownership* of the elected coins (Algorithm 7 line 20, Algorithm 8 line 18).

Chapter 7

Proofs

Structure of the Proofs section: First, for each protocol, we will formulate helpful lemmas to prove safety and liveness of the execution protocols. For safety, we must argue first that there are some state commitments that the nodes can trust. The objective of our protocols is that only a correct state commitment can be valid. For liveness, we must ensure that as long as the ledger grows the nodes will be generating valid state commitments.

Then we will prove the properties *E-Safety*, *E-Liveness* for all the protocols together using the aforementioned Lemmas as a foundation. Additionally, we prove the properties *LC-Safety*, *LC-Liveness* for the light client execution layer protocol together using these Lemmas. In the probabilistic protocols, namely the horizontal sampling and the PoS protocol, the properties hold except with negligible probability in the security parameter.

7.1 Deterministic Protocol

Definition 7.1.1. A state commitment is considered to be $valid$ if and only if either it is signed by at least $f + 1$ nodes or it is the genesis block.

Definition 7.1.2. *Successful round*: A round r is *successful* if and only if a valid state commitment visible to every honest node exists for this round.

Lemma 7.1.3. *Every valid state commitment is correct.*

Proof. We will prove the Lemma by reaching a contradiction. Assume that the state commitment cmt_r corresponding to the round $r \geq 1$ is valid but not correct. Since cmt_r is valid, it is signed by at least $f + 1$ nodes and therefore from at least an honest node. We call the existing honest node p_i. Node p_i has started updating the system from the genesis block, applying the transactions committed to the ledger via the execution engine M. Thus, p_i must have obtained the correct state corresponding to the round r. So, we conclude that p_i must have misbehaved and thus we reach a contradiction. □

Lemma 7.1.4. *State availability: The state corresponding to a valid state commitment will be available to any user u_i.*

Proof: Consider the valid state commitment cmt_r for the round r. The commitment cmt_r is signed by at least $f + 1$ nodes and therefore from at least an honest one, which we call p_i. Node

p_i has stored the corresponding state. A user u_i requests the state from all the nodes that have signed the commitment cmt. Since p_j is honest, it will eventually send the corresponding state to u_i. □

Lemma 7.1.5. *Every round r corresponding to a block committed to the dirty ledger will eventually be* successful.

Proof: Consider the round $r \geq 1$. There are at least $f + 1$ honest nodes executing in each round (Algorithm 5 Procedure Execute). All these honest nodes download the data committed to the dirty ledger up to round r and apply only the valid transactions to obtain the updated state. For each one of these rounds, the honest nodes compute the respective state commitment. Since the commitment scheme is deterministic the $f + 1$ honest nodes output the same result forming a valid commitment for all the rounds up to the round r. □

7.2 Horizontal Sampling protocol

7.2.1 Permissioned Protocol

In the following proofs: $\kappa = O(log^2 n)$.

Lemma 7.2.1. *Consider a number of n nodes with $f \leq (1-\epsilon)\frac{n}{2}$ Byzantine nodes and $\geq (1+\epsilon)\frac{n}{2}$ honest nodes, where $0 < \epsilon < \frac{1}{2}$. Let κ denote the security parameter. A sampling is taking place, where each node is chosen independently with a probability of $\frac{\kappa}{n}$. The following arguments hold except a negligible probability in κ:*
1. There will be at least $(1 + \frac{\epsilon}{2})\frac{\kappa}{2}$ honest nodes chosen.
2. There will be at most $(1 - \epsilon)\frac{\kappa}{2}$ Byzantine nodes chosen.

Proof. 1. Let H denote the number of selected honest nodes. Since each node is chosen with probability $\frac{\kappa}{n}$, we have $E[H] \geq (1 + \epsilon)\frac{n}{2} \cdot \frac{\kappa}{n} = (1 + \epsilon)\frac{\kappa}{2}$. From Chernoff bounds, we have $P[H \leq (1 - \delta)E[H]] \leq e^{-\frac{\delta^2 E[H]}{2}}$ where $0 \leq \delta \leq 1$. Since $E[H] \geq (1 + \epsilon)\frac{\kappa}{2}$, we have $P[H \leq (1 - \delta)(1 + \epsilon)\frac{\kappa}{2}] \leq P[H \leq (1 - \delta)E[H]] \leq e^{-\frac{\delta^2 E[H]}{2}}$. Choosing $\delta = \frac{\epsilon}{2(1+\epsilon)}$, we have $P[H \leq (1 + \frac{\epsilon}{2})\frac{\kappa}{2}] \leq e^{-\frac{\epsilon^2 \kappa}{16(1+\epsilon)}}$.

2. Let F denote the number of selected Byzantine nodes. Since each node is chosen with probability $\frac{\kappa}{n}$, we have $E[F] \leq (1-\epsilon)\frac{\kappa}{2}$. Again using Chernoff bounds, $P[F \geq (1+\delta)E[F]] \leq e^{-\frac{\delta^2 E[F]}{2+\delta}}, 0 < \delta < 1$, and the fact that $P[F \geq (1 + \delta)(1 - \epsilon)\frac{\kappa}{2}] \leq P[F \geq (1 + \delta)E[F]]$, and choosing $\delta = \frac{\epsilon}{1-\epsilon}$ leads to $P[F \geq \frac{\kappa}{2}] \leq e^{-\frac{\epsilon^2 \kappa}{4-2\epsilon}}$. □

Definition 7.2.2. A majority commitment corresponding to some round r is a state commitment voted by at least $\frac{\kappa}{2}$ nodes among the ones elected during the interval of rounds $[max(1, r - \kappa + 1), r]$.

Definition 7.2.3. A state commitment is considered to be valid if and only if either it is a majority commitment or it is the genesis block.

Lemma 7.2.4. *The* bootstrap committee *consists of at least* $\frac{\kappa}{2}$ *honest and at most* $\frac{\kappa}{2} - 1$ *adversarial nodes except with negligible probability in* n*.*

Proof: In the first round each node is elected with a probability of $p = \frac{\kappa}{n}$ after computing the VRF. Since the majority of the nodes are honest and considering the VRF computations as independent Bernoulli trials, the argument holds by applying Lemma 7.2.1. Since $\kappa = O(log^2 n)$ the probabilities are overwhelming in n. □

Lemma 7.2.5. *In any interval of rounds* $[r, r + \kappa - 1], r \geq 1$*, there will be at least* $\frac{\kappa}{2}$ *honest and at most* $\frac{\kappa}{2} - 1$ *adversarial, not necessarily distinct, elected nodes with a high probability in* n*.*

Proof: Case 1, $r = 1$: Only votes coming from *bootstrap committee* members' are taken into account for this interval. Following from Lemma 7.2.4, there are at least $\frac{\kappa}{2}$ honest and at most $\frac{\kappa}{2} - 1$ adversarial *bootstrap committee* members except with negligible probability in n.
Case 2, $r \geq 2$: Without loss of generality, consider the interval of rounds $[r, r + \kappa - 1]$. In every round within that interval, each node computes the VRF for the election phase (Algorithm 6 line 27), which is a Bernoulli trial with a probability of success $p = \frac{1}{n}$. Let H (or F) denote the number of honest (or adversarial) elected nodes within that interval. Then $E[H] \geq \sum_{i=r}^{r+\kappa-1}(1+\epsilon)\frac{n}{2}\frac{1}{n} = (1+\epsilon)\frac{k}{2}$. Similarly, $E[F] \leq \sum_{i=r}^{r+\kappa-1}(1-\epsilon)\frac{n}{2}\frac{1}{n} = (1-\epsilon)\frac{k}{2}$. Since computing the VRF results in independent Bernoulli trials, we can apply Chernoff bounds for $E[H]$ and $E[F]$. With the same analysis done in Lemma 7.2.1: $P[H \leq (1-\delta)E[H]] \leq e^{-\frac{\epsilon^2\kappa}{16(1+\epsilon)}}$ and $P[F \geq \frac{\kappa}{2}] \leq e^{-\frac{\epsilon^2\kappa}{4-2\epsilon}}$. Since $\kappa = O(\log^2 n)$, we have these probabilities negligible in n. □

Lemma 7.2.6. *Every valid state commitment is correct except with negligible probability in* n*.*

Proof. We will prove this theorem by induction on the rounds for which a valid state commitment exists.
Base step: Genesis block is the valid and correct state commitment for round 0.
Induction step: Consider the round $r \geq 1$ for which the valid state commitment cmt_r exists. Assume that the theorem holds for every round $r' < r$. First, if $1 \leq r \leq \kappa$, there are at most $\frac{\kappa}{2} - 1$ adversarial *bootstrap committee* members except with negligible probability in n according to Lemma 7.2.4. If $r \geq \kappa + 1$, the elected adversarial nodes in the interval $[r - \kappa + 1, r]$ are at most $\frac{\kappa}{2} - 1$ with overwhelming probability in n according to Lemma 7.2.5. So, for any $r \geq 1$, cmt_r must be signed by at least one honest node, which we call p_i. P_i was a leader in some round $r_l, max(1, r - \kappa + 1) \leq r_l \leq r$. Node p_i started the execution from the state corresponding to a valid state commitment $cmt_{r_{l-1}}$ of the round r_{l-1}. It verified that the state corresponds to $cmt_{r_{l-1}}$ by computing the respective commitment. According to the induction assumption, $cmt_{r_{l-1}}$ is correct. Finally, p_i applied the valid transactions committed to the ledger for all the intermediate rounds via the execution engine M. Therefore, since cmt_r is signed by p_i it must be correct. □

Lemma 7.2.7. *State availability: Consider a round for which a valid state commitment exists. Each node that requests the corresponding state will eventually receive it except with negligible probability in* n*.*

Proof: Consider node p_i that needs the state corresponding to a valid state commitment of a round r. P_i requests the state from all the nodes that have signed the state commitment. Each valid state commitment is signed by at least one honest node except with negligible probability in n. Otherwise, the adversarial nodes could form a valid state commitment on their own contradicting Lemma 7.2.6. Let us call p_j an existing honest node that have signed the valid state commitment. Node p_j has stored the corresponding state and will eventually deliver it to p_i. □

Definition 7.2.8. We call the interval of rounds $[r-\kappa+1, r]$ *well formed* if and only if every honest elected node for a round $r_l \in [r-\kappa+1, r]$ has witnessed a valid state commitment for the round $r_l - 1$.

Lemma 7.2.9. *Consider a* well formed *interval of rounds* $[r-\kappa+1, r]$. *Round* r *will eventually become successful except with negligible probability in* n.

Proof: Consider a round $r \geq \kappa$ such that the interval $[r-\kappa+1, r]$ is well formed. So, each honest node elected in a round $r_l \in [r-\kappa+1, r]$ has witnessed a valid state commitment for the round $r_l - 1$. Following from Lemma 7.2.5 there are at least $\frac{\kappa}{2}$ honest nodes elected in that interval with overwhelming probability, constituting the subset H. Every node in H will acquire the state corresponding to the valid state commitment it has seen according to Lemma 7.2.7. The valid state commitment, and therefore the state, is correct according to Lemma 7.2.6. Nodes in H will next apply the valid transactions committed to the ledger and compute the state commitments up to the round r (Algorithm 6 lines: 18-22). The leaders will then broadcast the signed state commitment along with their proof of election to every node (Algorithm 6 line 25). Every node will verify their signatures and their proofs of election (Algorithm 6 line 32). Since the commitment scheme is deterministic and the nodes in H start from a correct state, they will output the same result forming a valid state commitment for the round r. □

Lemma 7.2.10. *Every round* r *corresponding to a block committed to the dirty ledger will eventually be* successful *except with negligible probability in* n.

Proof: The interval of rounds $[2, \kappa+1]$ will eventually become well formed, since there are at least $\frac{\kappa}{2}$ honest *bootstrap committee* members, according to Lemma 7.2.4, that will compute and broadcast to every node the state commitments for every round $r \in [1, \kappa]$ along their *proofs of election*. Since the commitment scheme is deterministic their outputs will be identical, forming the respective valid state commitments for these rounds.

Applying Lemma 7.2.9 for the *well formed* interval $[r_c, r_c + \kappa - 1]$ results in a valid state commitment $cmt_{r_c+\kappa-1}$ visible to every honest node for the round $r_c + \kappa - 1$. Therefore, the interval $[r_c+1, r_c+\kappa]$ becomes *well formed*. By applying this argument recursively for $r_c \geq 2$, every round r will eventually become *successful*. □

Lemma 7.2.11. *Efficiency guarantees: The expected number of rounds for which each node executes equals* $\frac{h\kappa}{n}$, *where* h *is the number of the blocks committed to the dirty ledger and* $\kappa = O(log^2 n)$.

Proof: In the first round, nodes are elected with a probability of $p = \frac{\kappa}{n}$ and execute for κ rounds. For the rounds $r \geq 2$, consider the random variable N depicting the number of

times that the honest node p_i is elected in $h-1$ rounds. N follows the binomial distribution $N \sim B(h-1, \frac{1}{n})$. Each time that p_i is elected, it executes for κ rounds. Thus, the expected value of the random variable X capturing the number of rounds for which p_i executes in h rounds is equal to $E[X] = \frac{\kappa^2}{n} + \kappa E[N] = \frac{(h+\kappa-1)\kappa}{n}$. □

7.2.2 Proof-of-stake protocol

In the following proofs: $\kappa = O(log^2 W)$.

Definition 7.2.12. We say that an interval of rounds $[r, r+\kappa-1]$ is *honest prevalent* if at least $\frac{\kappa}{2}$ elected coins within that interval are held by honest nodes and at most $\frac{\kappa}{2}-1$ elected coins by adversarial nodes.

Definition 7.2.13. Consider the node p_i. We call and denote by $C_{p_i,r}$ the stake of p_i in the round r according to its view as *relative stake*, and the stake of p_i according to the correct state S_r^*, denoted by $C_{p_i,r}^*$, as *actual stake*.

Definition 7.2.14. A *majority commitment* corresponding to some round r is a state commitment voted by the owners of at least $\frac{\kappa}{2}$ of the elected coins during the interval of rounds $[max(1, r-t+1), r]$.

Definition 7.2.15. A state commitment is considered to be valid if and only if either it is a majority commitment or it is the genesis block.

Lemma 7.2.16. *Bootstrap committee: Consider the interval of rounds* $[1, \kappa]$. *The following arguments hold except with negligible probability in* W*: i)* $[1, \kappa]$ *is* honest prevalent*, ii) for every round* $r \in [1, \kappa]$ *there will eventually exist a valid state commitment which is correct and visible to every honest node, iii) every node requesting a state corresponding to any of the valid state commitments will eventually receive it.*

Proof: i) Only votes coming from nodes elected in the first round are valid for this interval. In the first round, every node computes the VRF for each coin it owns, events that are independent Bernoulli trials with a probability of success $p = \frac{X_1}{2^{|h|}} = \frac{\kappa}{W}$. Since the majority of the coins in the initial stake distribution are held by honest nodes, we can apply Lemma 7.2.1 where instead of nodes we sample their coins, to conclude that there will be at least $\frac{\kappa}{2}$ elected coins owned by honest nodes and at most $\frac{\kappa}{2}-1$ elected coins held by adversarial nodes.
ii) Since the interval is *honest prevalent*, there are at least $\frac{\kappa}{2}$ elected coins held by honest nodes. These honest nodes will apply the valid transactions committed to the ledger to update the state for every round $r \in [1, \kappa]$ and broadcast the corresponding state commitments to every node (Algorithm 8 Procedure Execute). Since the commitment scheme is deterministic and they all start from the genesis block their output will be identical, forming the corresponding valid state commitments which must be correct.
iii) There are at most $\frac{\kappa}{2}-1$ coins held by adversarial nodes within that interval. Therefore, each valid state commitment is signed by at least one honest node which has stored the state and will eventually provide it to every node requesting it. □

Lemma 7.2.17. *Bootstrapping: Consider two nodes p_i, p_j that have bootstrapped in the system in the rounds $r_i, r_j \geq 1$ respectively. If $r_i \leq r_j$ and p_j have accepted the chain $chain_j$ to bootstrap, p_i would consider as valid every state commitment included in $chain_j$.*

Proof: For every round $r \geq 1$, p_j has access to the random seed via the dirty ledger for the proofs of election of the elected nodes. $Chain_j$ starts from the genesis block and continues with the valid state commitments that the *bootstrap committee* forms according to Lemma 7.2.16. In any round $r > 1$, p_j verifies the respective proofs of ownership coming from nodes with elected coins only when the parent commitment is the valid state commitment of the previous round. For the *voting phase*, p_j takes into account only votes coming from elected nodes. Since p_i and p_j perform the same verifications for the *election* and the *voting* phase, p_i would accept as valid every state commitment included in $chain_2$. □

According to Lemma 7.2.17 we cannot distinguish bootstrapping nodes from nodes that have been participating from the first round. From now on, we refer to both as nodes.

Lemma 7.2.18. *Consider the interval of rounds $[r, r + \kappa - 1], r \geq 2$. If for every round $r' \in [r-1, r+\kappa-2]$ and for each honest node p_i: i) p_i eventually receives a $valid$ state commitment which is $correct$, ii) $C_{p_i,r'} = C^*_{p_i,r'}$, then the interval $[r, r+\kappa-1]$ will eventually become* honest prevalent *in every honest node's view except with negligible probability in W.*

Proof: Each coin is elected with a probability of $p = \frac{1}{W}$ after its owner computes the VRF for it. Adversarial nodes cannot construct a proof of ownership for coins they do not own and every honest node with elected coins in any round $r' \in [r, r + \kappa - 1]$ will provide a proof of inclusion with parent commitment the valid state commitment of round $r' - 1$ that will be accepted by everyone. Since the valid state commitments are correct and in each round the majority of the coins are held by honest nodes by assumption, the argument holds with the analysis done in Lemma 7.2.5 (case 2) where the independent Bernoulli trials are the coins' election. □

Lemma 7.2.19. *Consider the* honest prevalent *interval of rounds $[r, r + \kappa - 1]$ and that i) every honest node eventually receives a valid state commitment which is correct for every round in the interval $[r-1, r+\kappa-2]$, ii) every honest node with an elected coin in a round $r' \in [r, r+\kappa-1]$ will eventually receive the corresponding state of the round $r' - 1$. There will eventually exist a valid state commitment for the round $r + \kappa - 1$ which will be correct and visible to every honest node except with negligible probability in W.*

Proof: i) Consider the set H including the honest nodes that own elected coins with that interval. Now, consider the node $p_i \in H$ with elected coins in a round $r' \in [r, r + \kappa - 1]$. P_i starts executing from the correct state of the round $r' - 1$. Then, it applies the data committed to the ledger up to the round $r' + \kappa - 1 \geq r$ via the execution engine M, computes and broadcast to every node the state commitments for all the intermediate rounds, which must be correct since p_i started from a correct state. Node p_i provides the *proof of election* and the *proof of ownership* with the parent commitment $cmt_{r'-1}$, which is the valid and correct state commitment of the round $r' - 1$, for all of its elected coins along with the signed state commitment. All honest nodes will verify the *proof of election*, the signature, and the *proof of ownership* since $cmt_{r'-1}$ will eventually be visible to everyone by assumption. Since the nodes in H own at least $\frac{\kappa}{2}$ elected

coins and the commitment scheme is deterministic, they will form a valid state commitment for the round r which will be visible to every node. □

Lemma 7.2.20. *State availability: Consider the* honest prevalent *interval of rounds* $[r, r+\kappa-1]$. *If the valid state commitment* cmt *for the round* $r+\kappa-1$ *exists, it will be available to all the nodes requesting it except with negligible probability in* W.

Proof: Consider node p_i that has seen a valid state commitment for the round r and requests the corresponding state. P_i requests the state from all the nodes with elected coins in that interval that have signed for the valid state commitment. Since $[r, r+\kappa-1]$ is *honest prevalent*, there are not enough adversarial members with elected coins to form a $valid$ state commitment on their own. Thus, cmt must be signed by at least one honest node. This honest node has stored the corresponding state and will eventually send it to p_i. □

Lemma 7.2.21. *For every* $r \geq 1$*, the interval of rounds* $[r, r+\kappa-1]$ *will eventually become* honest prevalent *in every honest node's view except with negligible probability in* W.

Proof: The interval $[1, \kappa]$ is *honest prevalent* and for every round within that interval there is a corresponding valid state commitment which is correct according to Lemma 7.2.16. Since there is a valid state commitment visible to every node for each round, every honest node that was paid within that interval requesting a *proof of payment* will eventually receive it by the payer. Therefore, each honest node p_i can determine whether the transactions it receives are successful and it knows which coins it has spent, so for every round $r \in [1, \kappa]$ $C_{p_i,r} = C^*_{p_i,r}$.

Since there is a valid and correct state commitment for every round in $[1, \kappa]$ and each node tracks its *actual stake*, the interval $[2, \kappa+1]$ will become *honest prevalent* according to Lemma 7.2.18. For each round in $[1, \kappa]$, the state corresponding to the valid commitment will be available according to Lemma 7.2.16. Finally, we can apply Lemma 7.2.19 to conclude that a valid state commitment which is correct will eventually exist for the round $\kappa+1$. For the intervals $[r, r+\kappa-1], r \geq 3$, we apply the same argument recursively where for the state availability of the round $r+\kappa-2$ we use Lemma 7.2.20. □

Lemma 7.2.22. *Every valid state commitment is correct except with negligible probability in* W.

Proof: We will prove the Lemma by reaching a contradiction. Assume that there is a round r to which the $valid$ state commitment cmt_r that is not $correct$ corresponds. Then, there must be a round $r^*, 1 \leq r^* \leq r-\kappa+1$ which is the first round s.t. there are at least $\frac{\kappa}{2}$ elected coins within the interval $[r^*, r^*+\kappa-1]$ held by adversarial nodes. By Lemma 7.2.21 we conclude that the interval $[r^*, r^*+\kappa-1]$ is *honest prevalent* and thus we reach a contradiction. □

We say that an interval of rounds $[r, r+\kappa-1]$ is *adversarial prevalent* if at least $\frac{\kappa}{2}$ elected coins within that interval are held by adversarial nodes.

Lemma 7.2.23. *Every round r corresponding to a block committed to the dirty ledger will eventually be* successful *except with negligible probability in* W.

Proof: For every round $r \in [1, \kappa]$ the *bootstrap committee* forms the respective valid state commitments following from Lemma 7.2.16. For every round $r \geq \kappa+1$ the interval $[r-\kappa+1, r]$

is *honest prevalent* and the honest nodes with elected coins within that interval form the valid state commitment for round r according to Lemma 7.2.21. □

Lemma 7.2.24. *The expected number of rounds for which an honest node p_i executes equals $\kappa(\kappa W_{p_i,1} + \sum_{r=2}^{h} W_{p_i,r})/W$, where $W_{p_i,r}$ is $p_i's$ stake in the round r, W is the total stake distribution, h is the number of the blocks committed to the dirty ledger, and $\kappa = O(log^2 W)$.*

Proof: The probability with which at least one of $p_i's$ coin is elected in the round r equals $p_{i,r} = \begin{cases} \frac{\kappa W_{p_i,r}}{W}, & \text{if } r = 1 \\ \frac{W_{p_i,r}}{W} & \text{if } r > 1 \end{cases}$. When p_i is elected, it executes for κ rounds. We denote the number of the rounds for which node p_i will execute by the random variable X, expressed as the sum of independent Poisson trials, with excepted value: $E[X] = \sum_{r=1}^{h} \kappa p_{i,r} = \kappa \frac{\kappa W_{p_i,1}}{W} + \sum_{r=2}^{h} \kappa \frac{W_{p_i,r}}{W}$. □

7.3 Execution protocol

We refer to the Lemmas 7.1.3, 7.2.6, 7.2.22 and 7.1.5, 7.2.10, 7.2.23 that guarantee, respectively, that valid state commitments are correct and that executors keep producing them as long as the ledger grows. We prove the properties $E - Safety, E - Liveness$ on top of these Lemmas.

The rationale behind the following definition, which indicates when nodes $commit$ to a state, is that we only need to guarantee the generation of valid state commitments and the corresponding states' availability. When those conditions are satisfied, nodes can acquire the state whenever they wish.

Definition 7.3.1. We say that node p_i commits to the state S in the round r, if and only if p_i has received a valid state commitment cmt for that round such that $cmt = compute_{cmt}(S)$.

Theorem 7.3.2. *All of our protocols (Deterministic 4.1, Horizontal Sampling 4.2, PoS 4.3) satisfy the E-Safety property.*

Proof: Nodes commit only to valid state commitments. Valid state commitments are correct according to Lemmas 7.1.3, 7.2.6, 7.2.22 for each respective protocol (Deterministic, Horizontal Sampling, PoS). Therefore, if node p_i commits to the state S in the round r it must hold that $S = S_r^*$. □

Theorem 7.3.3. *All of our protocols (Deterministic 4.1, Horizontal Sampling 4.2, PoS 4.3) satisfy the E-Liveness property.*

Proof: In all the protocols (Deterministic, Horizontal Sampling, PoS), nodes form a valid state commitment visible to every node for every round according to Lemmas 7.1.5,7.2.10,7.2.23 respectively. Therefore, in all protocols, for every round r and every node p_i there will a round $r' > r$ for which p_i commits to a different state. □

Theorem 7.3.4. *All of our protocols (Deterministic 4.1, Horizontal Sampling 4.2, PoS 4.3) satisfy the* Censorship resistance *property.*

Consider a valid transaction tx committed to the ledger in the block with height r. Enough honest nodes will eventually execute the block in the round r, apply all the valid transactions, and therefore tx, to the state, and form the corresponding valid state commitment according to Lemmas 7.1.5,7.2.10,7.2.23 for each protocol (Deterministic, Horizontal Sampling, PoS) respectively. Following from Lemmas 7.1.3, 7.2.6, 7.2.22, the valid state commitment will be correct, namely every valid transaction, including tx, must have been applied to the state.□

7.4 Light clients

Again we use Theorems 7.1.3, 7.2.6, 7.2.22 and 7.1.5, 7.2.10, 7.2.23 concerning the validity and availability of state commitments.

Lemma 7.4.1. *Light nodes perform the same verifications with the executors for valid state commitments in the: i)Deterministic protocol, ii) Horizontal Sampling protocol, iii) PoS protocol.*

Proof: i) In the Deterministic protocol, the light clients know in advance the public keys of the executors.
ii) In the Horizontal Sampling, light clients know the public keys of the executors before the protocol starts as well as they have access to the random seed for each round through the dirty ledger, to verify the proofs of the election of the nodes.
iii) Following from Lemma 7.2.17. □

Theorem 7.4.2. *All of the proposed protocols (Deterministic 4.1, Horizontal Sampling 4.2, PoS 4.3) satisfy the LC-Safety property.*

Proof: A light node joins the network after having received a *state proof* consisting of a valid state commitment and the respective certificates. Assume that there exists a round r in which the adversary provides a state proof $\pi_{\tilde{S}}$ corresponding to a faulty state $\tilde{S}$ to a light client lc_i s.t. lc_i outputs $accept_{lc_i}(\pi_{\tilde{S}}, r) = True$. Since light clients perform the same verifications as honest nodes, according to Lemma 7.4.1, the adversary would convince the executors about the validity of the faulty commitment $cmt = Compute_{cmt}(\tilde{S})$ as well. That is contradicting Lemmas 7.1.3, 7.2.6, 7.2.22 for each respective protocol (Deterministic, Horizontal Sampling, PoS). □

Theorem 7.4.3. *All of the proposed protocols (Deterministic 4.1, Horizontal Sampling 4.2, PoS 4.3) satisfy the LC-Liveness property.*

Proof: Consider the light client lc_i that requests a *state proof* for a round greater or equal than r. Each round will eventually become *successful* in all protocols (Deterministic, Horizontal Sampling, PoS) as follows from the Lemmas 7.1.5, 7.2.10, 7.2.23. The honest executors store all the certificates related to the valid state commitments. Lc_i is connected to at least one honest node, let us call it p_i. Since round r will become successful, p_i will eventually see a valid state commitment corresponding to a round $r^* \geq r$. The *state proof* π_S which p_i will construct and send to lc_i, contains the state commitment and the respective certificates indicating its validity. The light client will verify π_S, since the honest executor had already verified the certificates, and output $accept_{lc_i}(\pi_S, r^*) = True$. □

For the following Lemma, we denote the size of the nodes' signatures by c_1, the size of the *proof of election* (proof coming from the VRF) by c_2, and the average size of the *proof of ownership* (inclusion proof) by $I(|S|)$. We assume that $\frac{I(|S|)}{|S|} = o(1)$, where $|S|$ is the average size of the state. For example, in Merkle proofs $I(|S|) = O(log(|S|))$.

Lemma 7.4.4. *A* state proof π *for a round* $r \geq 1$*, is succinct in all of our proposed protocols: i) Deterministic 4.1, ii) Horizontal Sampling 4.2, iii) PoS 4.3.*

Proof: i) In the Deterministic protocol, π constitutes of $f + 1$ signatures and therefore $len(\pi) = (f + 1)c_1$.
ii) In the Horizontal Sampling, π consists of $O(log^2 n)$ signatures and the respective *proofs of election* and thus $len(\pi) = (c_1 + c_2)O(log^2 n)$.
iii) In the Proof-of-Stake protocol, for every round up to the round r there are $O(log^2 W)$ votes along with the signatures and the proofs of election and ownership. Consequently, $len(\pi) = r(c_1 + c_2 + I(S))O(log^2 W)$. □

Chapter 8

Extensions and future work

8.1 Extending the horizontal sampling protocol in the permissionless settings

Now we extend the horizontal sampling protocol in the permissionless settings. More specifically, the protocol can be decomposed into the following operations: i) first, there is the election phase, in which we assign leaders to rounds, and ii) second, there is the voting phase, in which the elected leaders propose state commitments as described in section 4.2.

Leader election phase To defend against Sybil attacks, we employ the PoW puzzle for the election phase, similarly to the Nakamoto consensus. In a nutshell, participating nodes mine blocks that contain a reference to a previous block, their public key, and a nonce value. The block's header occurs by hashing these inputs together. A block is valid only if its hash header is less than a target value, known to all participants, and if it extends the longest chain. This approach is similar to the Bitcoin protocol, where instead of transactions the nodes put their public keys within the blocks. According to the *common prefix* property which is defined in [20] and is parameterized by κ, honest nodes will always be following identical chains with the exception of the last κ blocks. We call this mutual chain *leader election sequence*. Furthermore, following from the properties of *chain quality, chain growth* the *leader election sequence* will be ever-growing. For each round, the elected leader is the one to which the public key committed to the respective block corresponds. Note that if the executors are the same nodes forming the dirty ledger T, they can put their keys in the blocks committed to the ledger T.

Voting phase For the execution phase, nodes follow the horizontal sampling protocol 4.2. The leader of the round r is responsible for executing in the interval of rounds $[r - 2\kappa + 1, r]$, where κ is the parameter stated in the *common prefix property*. Between an interval of 2κ rounds, the majority of the leaders will be honest with a high probability in κ otherwise the adversary would violate the *common prefix* property. Leaders compute state commitments and sign them with the private key corresponding to the key they put in the dirty ledger.

8.2 Towards an adaptive adversary

All of the protocols we defined previously apply when considering a $static$ adversary. But at what cost can we withstand an adaptive adversary?

Why is it infeasible to deal with an adaptive adversary in our protocols? Corrupting nodes during the run: Corrupting nodes during the run is not a problem for the deterministic protocol. The adversary can corrupt at most f executors, hence each valid state commitment is signed by at least one node that will always behave truthfully. Additionally, in each round, there will always be at least $f + 1$ honest executors responsible for executing and storing the respective state. As a result, in the deterministic protocol, safety and liveness are ensured even in the presence of an adaptive adversary.

In the probabilistic protocols, the adversary can corrupt all of the elected nodes for one round (or interval of rounds in the horizontal sampling protocol) as soon as it learns their identity. In our protocols, the elected nodes request the state of a previous round before starting the execution. That acts as a witness to their election. As a result, the adaptive adversary can identify which nodes were elected and corrupt them immediately to form a faulty valid state commitment.

Withstanding the adaptive adversary: To deal with the adaptive adversary, we have to address two challenges. First, the elected nodes should not announce their election to the other nodes before having completed their execution. In our protocols, the executors do that when requesting the state needed to execute. To avoid that, each elected node should have access to the state required to execute without requesting it. To this end, when the elected leaders (or the committee members in the permissionless settings) perform the execution, they broadcast the signed state commitment along with the updated state. In that way, the state will eventually be available to all the honest executors. Now, the storage cost per round per executor is O(1).

The second challenge is that it must be infeasible for the adversary to re-execute, i.e. sign another state commitment, after corrupting an elected node. To defend against this attack, we could use ephemeral keys that can be utilized by Merkle Trees [27] or as was proposed by Micali et al. [22]. In a nutshell, each node has a pair of masterkeys which it uses to produce a pair of keys for each round. After executing for a round, the honest nodes delete the corresponding private key. Thus, even if the adversary corrupts an elected node immediately, it will not manage to make it sign a faulty commitment.

To summarize, when an elected leader (or the committee in the permissionless settings) executes, it signs the state commitment and then deletes the private key that corresponds to that round. Afterwards, it broadcasts the signed state commitment accompanied with the *proof of its election* and the updated state. Sending all the data in batches is essential. If the executors sent the information one by one, the adversary would learn their identity and corrupt them before sending the state to the rest of them. The issue that this approach faces is that having all the elected leaders sending the state to all of the nodes creates delays in the network.

8.3 Implementation

In this theory paper, we first lay the foundations of the problem of diving the SMR in the ordering and execution layers and then introduce our proposed protocols for the SMR execution layer.

However, interesting questions arise when it comes to implementing our protocols. The most interesting problem is how to extend our protocols to work with merely an honest minority if we assume synchrony assumptions. In a nutshell, we will allow the probability of failure to be low, but not necessarily negligible, and we will define a fallback mechanism, such as providing fraud proofs in case of system failure. Once that is done, we must not necessarily guarantee that an honest majority of nodes are elected per round during the election phase.

Bibliography

[1] Bringing the World to Ethereum | Polygon. https://polygon.technology.

[2] Fuel Network. https://fuel.network.

[3] Neon Team. Neon evm. https://neon-labs.org/Neon_EVM.pdf. Accessed: 3-8-2022.

[4] Optimism. https://www.optimism.io.

[5] The DiemBFT Team. State machine replication in the diem blockchain, 2021. https://developers.diem.com/docs/technical-papers/state-machine-replication-paper.

[6] Optimistic rollups: How they work and why they matter. https://medium.com/stakefish/optimistic-rollups-how-they-work-and-why-they-matter-3f677a504fcf, 2021.

[7] Rollups, data availability layers & modular blockchains: introductory meta post. https://polynya.medium.com/rollups-data-availability-layers-modular-blockchains-introductory-\meta-post-5a1e7a60119d, 2021.

[8] What is a zero-knowledge (zk) rollup? https://zebpay.com/blog/what-is-a-zero-knowledge-rollup-zk, 2022.

[9] Mustafa Al-Bassam. Lazyledger: A distributed data availability ledger with client-side smart contracts. *arXiv preprint arXiv:1905.09274*, 2019.

[10] Mustafa Al-Bassam, Alberto Sonnino, and Vitalik Buterin. Fraud proofs: Maximising light client security and scaling blockchains with dishonest majorities. *arXiv preprint arXiv:1809.09044*, 160, 2018.

[11] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the thirteenth EuroSys conference*, pages 1–15, 2018.

[12] Georgia Avarikioti, Eleftherios Kokoris-Kogias, and Roger Wattenhofer. Divide and scale: Formalization of distributed ledger sharding protocols. *arXiv preprint arXiv:1910.10434*, 2019.

[13] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. On the indifferentiability of the sponge construction. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 181–197. Springer, 2008.

[14] Ethan Buchman. *Tendermint: Byzantine fault tolerance in the age of blockchains*. PhD thesis, University of Guelph, 2016.

[15] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OsDI*, volume 99, pages 173–186, 1999.

[16] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: a dag-based mempool and efficient bft consensus. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 34–50, 2022.

[17] Christian Decker and Roger Wattenhofer. Information propagation in the bitcoin network. In *IEEE P2P 2013 Proceedings*, pages 1–10. IEEE, 2013.

[18] Xavier Défago, André Schiper, and Péter Urbán. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Computing Surveys (CSUR)*, 36(4):372–421, 2004.

[19] Jose M Faleiro and Daniel J Abadi. Rethinking serializable multiversion concurrency control. *arXiv preprint arXiv:1412.2324*, 2014.

[20] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 281–310. Springer, 2015.

[21] Rati Gelashvili, Alexander Spiegelman, Zhuolun Xiang, George Danezis, Zekun Li, Yu Xia, Runtian Zhou, and Dahlia Malkhi. Block-stm: Scaling blockchain execution by turning ordering curse to a performance blessing. *arXiv preprint arXiv:2203.06871*, 2022.

[22] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th symposium on operating systems principles*, pages 51–68, 2017.

[23] Neil Giridharan, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Bullshark: Dag bft protocols made practical. *arXiv preprint arXiv:2201.05677*, 2022.

[24] Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All you need is dag. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 165–175, 2021.

[25] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. In *Concurrency: the Works of Leslie Lamport*, pages 203–226. 2019.

[26] Dahlia Malkhi, Kartik Nayak, and Ling Ren. Flexible byzantine fault tolerance. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, pages 1041–1053, 2019.

[27] Ralph C Merkle. A digital signature based on a conventional encryption function. In *Conference on the theory and application of cryptographic techniques*, pages 369–378. Springer, 1987.

[28] Silvio Micali, Michael Rabin, and Salil Vadhan. Verifiable random functions. In *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*, pages 120–130. IEEE, 1999.

[29] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, page 21260, 2008.

[30] Elaine Shi. Foundations of distributed consensus and blockchains. book manuscript, 2020. Available at https://www.distributedconsensus.net.

[31] Chrysoula Stathakopoulou, Tudor David, Matej Pavlovic, and Marko Vukolić. Mir-bft: High-throughput robust bft for decentralized networks. *arXiv preprint arXiv:1906.05552*, 2019.

[32] Ertem Nusret Tas, Dionysis Zindros, Lei Yang, and David Tse. Light clients for lazy blockchains. *arXiv preprint arXiv:2203.15968*, 2022.

[33] Qin Wang, Jiangshan Yu, Shiping Chen, and Yang Xiang. Sok: Diving into dag-based blockchain systems. *arXiv preprint arXiv:2012.06128*, 2020.

[34] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356, 2019.