Ανάλυση πρωτοκόλλων αλυσίδων συναλλαγών κρυπτονομισμάτων

Ελένη Μακρή

Διπλωματική εργασία

Επιβλέπων: Αριστείδης Παγουρτζής, Καθηγητής
Συνεπιβλέπων: Πέτρος Ποτίκας, ΕΔΙΠ

Σχολή Εφαρμοσμένων Μαθηματικών και Φυσικών Επιστημών
Εθνικό Μετσόβιο Πολυτεχνείο
Αθήνα, Ιούλιος, 2022

Ευχαριστίες

Θα ήθελα να ευχαριστήσω κατ'αρχάς τον κύριο Αριστείδη Παγουρτζή, για την ευκαιρία να ασχοληθώ με ένα τόσο σύγχρονο και ενδιαφέρον θέμα, για την στήριξη και τις συμβουλές κατα την διάρκεια εκπόνησης της παρούσας διπλωματικής εργασίας. Επιπλέον, οφείλω ένα θερμό ευχαριστώ στον κύριο Πέτρο Ποτίκα για την πολύτιμη βοήθεια και την διαθεσιμότητα του.
Ένα μεγάλο ευχαριστώ στην οικογένεια μου για την συμπαράσταση, την κατανόηση και την πίστη τους σε εμένα καθ' όλη την διάρκεια των σπουδών μου. Επιπλέον, θα ήθελα να ευχαριστήσω την Δέσποινα Μαρκίδη και την Χρυστάλλα Καλυφομμάτου για τα όμορφα φοιτητικά χρόνια, την στήριξη και την ενθάρρυνση που έλαβα.

Ελένη Μακρή,
Αθήνα, Ιούλιος 2022

Στην αδερφή και φίλη μου, Αναστασία.

Περίληψη

Με την πάροδο των χρόνων από την εμφάνιση του Bitcoin, όλο και περισσότερα blockchain προτείνονται ως εναλλακτικές. Τα πρωτόκολλα που βασίζονται σε Proof-of-Stake ομοφωνία είναι η βασικότερη αντιπρόταση ως ενεργειακά αποδοτικές λύσεις με πολύ καλή απόδοση επικύρωσης συναλλαγών. Σε αυτή την εργασία γίνεται περιγραφή, ανάλυση και σύγκριση των πρωτοκόλλων συναλλαγών κρυπτονομισμάτων του Algorand και του Ouroboros. Σκοπός είναι ο αναγνώστης να μπορεί να κατανοήσει σε βάθος τους μηχανισμούς και τους κανόνες με τους οποίους λειτουργούν αυτά τα πρωτόκολλα, και να κινητοποιηθεί ενδιαφέρον για περαιτέρω έρευνα.

Λέξεις/φράσεις κλειδιά: Blockchain, Κατανεμημένα Συστήματα, Proof-of-Stake, Αλγόριθμοι Ομοφωνίας, Βυζαντινά Σφάλματα, Βυζαντινή Συμφωνία, Ψευδοτυχαιότητα, Σπόρος Τυχαιότητας, Verifiable Random Functions, Algorand, Ouroboros

Abstract

Over the years, since the emerge of Bitcoin, more and more blockchains have been proposed as alternatives. The main counter-proposals are protocols based on Proof-of-Stake consensus, as energy efficient solutions with great performance in transaction validation. In this thesis, we will describe and compare the Proof-of-Stake protocols of Algorand and Ouroboros. The aim is for the reader to be able to deeply understand the mechanisms and rules these protocols follow, and raise interest for future research.

Key words/phrases: Blockchain, Distributed Systems, Proof-of-Stake, Consensus Algorithms, Byzantine Faults, Byzantine Agreement, Pseudo-randomness, Seed, Verifiable Random Functions, Algorand, Ouroboros

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

Η εργασία αυτή είναι χωρισμένη σε τέσσερα μέρη.

Α) Στο πρώτο τμήμα της εργασίας παρουσιάζονται βασικές γνώσεις που πρόκειται να χρησιμοποιηθούν κατα την ανάλυση των πρωτοκόλλων. Πρόκειται για τα κεφάλαια 2 έως 5. Πιο συγκεκριμένα, δίνεται μια γενική ιδέα για κάποια βασικά κρυπτογραφικά εργαλεία, όπως η ασύμμετρη κρυπτογραφία, οι ψηφιακές υπογραφές και οι συναρτήσεις κατακερματισμού. Στην συνέχεια, παρουσιάζεται ένα θεωρητικό υπόβαθρο σχετικά με τα κατανεμημένα συστήματα, το οποίο καταλήγει στην περιγραφή της τεχνολογίας του blockchain σε διάφορα επίπεδα. Τέλος, παρουσιάζεται εκτενέστερα το κρυπτογραφικό εργαλείο των Ψευδοτυχαίων Συναρτήσεων Επαληθευόμενου Αποτελέσματος (VRFs), το οποίο χρησιμοποιείται και στα δύο πρωτόκολλα που θα αναφερθούν και έχει μεγάλη αξία για την ασφάλεια τους.

Β) Στο δεύτερο κομμάτι (κεφάλαιο 6) μελετάται το πρώτο πρωτόκολλο αλυσίδας συναλλαγών του Algorand, το οποίο υλοποιεί το κρυπτονόμισμα ALGO. Σε ένα πρώτο επίπεδο περιγράφεται ο ρόλος και η συνεισφορά του στο κομμάτι των τεχνολογιών blockchain, καθώς φαίνεται πως λύνει το τρίλημμα που θέτει αυτή η τεχνολογία. Στην συνέχεια ξεκινάει η ανάλυση του πρωτοκόλλου από τις υποθέσεις λειτουργίας, έπειτα στο μοντέλο επικοινωνίας, για να καταλήξουμε στον κορμό που πρωτοκόλλου που είναι η ανάλυση της μεθόδου τυχαίας επιλογής χρηστών με βάση τον αλγόριθμο Proof-of-Stake, η δημιουργία του σπόρου τυχαιότητας, και η περιγραφή του πρωτοκόλλου βυζαντινής συμφωνίας.

Γ) Στο τρίτο κομμάτι (κεφάλαιο 7) μελετάται το δεύτερο πρωτόκολλο αλυσίδας συναλλαγών του Ouroboros, το οποίο είναι μια οικογένεια πρωτοκόλλων. Για τον λόγο αυτό το τρίτο κομμάτι χωρίζεται σε δύο μέρη, στο πρωτόκολλο Ouroboros Classic που ήταν το πρώτο που δημιουργήθηκε, και στο πρωτόκολλο Ouroboros Praos, που είναι μια βελτίωση του πρώτου. Και στις δύο περιπτώσεις, η ανάλυση γίνεται ξεκινώντας από τις

υποθέσεις και μια γενική περιγραφή. Το ιδιαίτερο με τα πρωτόκολλα Ouroboros είναι η ανάλυση ασφαλείας τους, και για τον λόγο αυτό περιγράφονται πρώτα σε ένα μικρό χρονικό διάστημα που ονομάζεται εποχή ή στατικό πρωτόκολλο όπου και αποδεικνύονται ασφαλή. Έπειτα περιγράφονται στο δυναμικό πρωτόκολλο, που αποτελείται από διαδοχικές εποχές και γίνεται επέκταση της αποδειχθείσας ασφάλειας.

Δ) Στο τελευταίο μέρος της παρούσας εργασίας (κεφάλαια 8 και 9) γίνεται μια σύγκριση μεταξύ του Algorand και του Ouroboros, στα επίπεδα του blockchain που αναφέρθηκαν στο πρώτο τμήμα, δηλαδή στο επίπεδο δεδομένων, στο επίπεδο δικτύου, στο επίπεδο ομοφωνίας και στο επίπεδο αμοιβών. Τέλος, παρουσιάζονται τα συμπεράσματα και οι τελικές σκέψεις σχετικά με την ανάλυση που διεξήχθη.

Κεφάλαιο 2

Υπόβαθρο

Σκοπός του κεφαλαίου είναι να γίνει μια εισαγωγή στα βασικά κομμάτια της κρυπτογραφίας, όπως οι συναρτήσεις κατακερματισμού ή οι ψηφιακές υπογραφές που είναι οι πυλώνες ασφάλειας των αποκεντρωμένων τεχνολογιών, όπως το blockchain.

2.1 Ασύμμετρη κρυπτογραφία

Η ασύμμετρη κρυπτογραφία ή αλλιώς κρυπτογραφία δημοσίου κλειδιού, είναι το κομμάτι της κρυπτογραφίας που χρησιμοποιείται περισσότερο, και ήταν μια ιδέα που προήλθε από τους W. Diffie και M. Hellman στο [45]. Σχεδόν την ίδια χρονική περίοδο, οι RL. Rivest, A. Shamir, και L. Adleman [44] πρότειναν το πρώτο κρυπτοσύστημα δημοσίου κλειδιού *RSA*. Η ασύμμετρη κρυπτογράφηση περιλαμβάνει την χρήση δύο κλειδιών, ενός δημοσίου και ενός ιδιωτικού κλειδιού, τα οποία με την βοήθεια συναρτήσεων μονής κατεύθυνσης (εύκολο να υπολογιστεί το αποτέλεσμα τους αλλά δύσκολο να ακολουθηθεί η αντίθετη πορεία), καθιστούν εύκολη και ασφαλή την κρυπτογράφηση και την αποκρυπτογράφηση.

Για να γίνουν αντιληπτά τα παραπάνω θα γίνει μια σύντομη ανασκόπηση της κρυπτογράφησης RSA. Η κρυπτογράφηση RSA βασίζει την ασφάλεια της στην μονόδρομη συνάρτηση $E(x) = x^e mod n$. Η αντιστοφή της $E(x)$ αποτελεί το πρόβλημα RSA:

Ορισμός 2.1.1 (**Πρόβλημα RSA** [9])**.** Δίνονται $N = pq$, e με $gcd(e, \varphi(N)) = 1$ και $c \in \mathbb{Z}_N^*$. Να βρεθεί η τιμή $c^{\frac{1}{e}}(mod n) = c^d mod n$, δηλαδή $m \in \mathbb{Z}_N^*$

ώστε $m^e mod N = c$.

Η εύρεση λοιπόν της e-οστής ρίζας $\mathrm{mod} n$ είναι δύσκολη εκτός και αν είναι γνωστά τα p, q, d. Αξιοποιώντας αυτή την πληροφορία, φτιάχτηκε το κρυπτοσύστημα RSA ως εξής :

- Δημιουργία ζεύγους κλειδιών:
  - Επιλογή μεγάλων πρώτων αριθμών p, q
  - Εύρεση $N = p \cdot q$
  - Εύρεση e ώστε $gcd(e, \varphi(N)) = 1$
  - Υπολογισμός $d = e^{-1}(mod\varphi(N))$
  - Δημόσιο κλειδί: (e, N) , Ιδιωτικό κλειδί: (p, q, d)
- Κρυπτογράφηση:
  - Κωδικοποίηση στο $\mathbb{Z}_N$
  - Υπολογισμός $c = m^e mod n$
- Αποκρυπτογράφηση :
  - Υπολογισμός $m = c^d mod n$

2.2 Ψηφιακές Υπογραφές

Οι ψηφιακές υπογραφές είναι πολύ βασικές για την επικύρωση πληροφοριών, καθώς δίνουν την δυνατότητα επικύρωσης και της πληροφορία ως προς την τροποποίηση της αλλά και του δημιουργού της. Στην ουσία μια ψηφιακή υπογραφή είναι ένα αλφαριθμητικό (string) που προκύπτει από ένα σχήμα υπογραφής μετασχηματίζοντας την αρχική πληροφορία. Το σχήμα ψηφιακών υπογραφών αποτελείται από έναν αλγόριθμο παραγωγής της ψηφιακής υπογραφής και έναν αλγόριθμο επαλήθευσης δωσμένης της υπογραφής, του μηνύματος και του δημοσίου κλειδιού. Πιο συγκεκριμένα, ο δημιουργός της υπογραφής, φτιάχνει ένα ζευγάρι δημοσίου και ιδιωτικού κλειδιού. Με την βοήθεια του ιδιωτικού κλειδιού και του αλγόριθμου παραγωγής, παράγεται η υπογραφή του μηνύματος. Αυτό το μήνυμα μπορεί να επαληθευτεί από οποιονδήποτε γνωρίζει το δημόσιο κλειδί του υπογράφοντα.

Το πιο σημαντικό κομμάτι στις ψηφιακές υπογραφές είναι ότι είναι αδύνατο να πλαστογραφηθούν. Δηλαδή είναι αδύνατο να παραχθεί μια ίδια υπογραφή για το ίδιο μήνυμα αλλά από διαφορετικό δημόσιο κλειδί, και η υπογραφή να επαληθευτεί από τον αντίστοιχο αλγόριθμο. Επιπλέον, τα σχήματα ψηφιακής υπογραφής είναι φτιαγμένα ώστε να είναι πολύ εύκολο κάποιος και να πράξει υπογραφή, αλλά και να επαληθεύσει.

Ένα σχήμα υπογραφής μπορεί να φτιαχτεί με βάση την κρυπτογράφηση RSA που περιγράφτηκε νωρίτερα. Για υπογραφή s και μήνυμα m, έχουμε :

- Δημιουργία ζεύγους κλειδιών:
  - Επιλογή μεγάλων πρώτων αριθμών p, q
  - Εύρεση $N = p \cdot q$
  - Εύρεση e ώστε $gcd(e, \varphi(N)) = 1$
  - Υπολογισμός $d = e^{-1}(mod\varphi(N))$
  - Δημόσιο κλειδί: (e, N) , Ιδιωτικό κλειδί: (p, q, d)
- Δημιουργία υπογραφής :
  Υπολογίζεται η υπογραφή s που είναι η συνάρτηση αποκρυπτογράφησης RSA: $s = m^d modn$ και διαμοιράζεται.
- Επαλήθευση υπογραφής :
  Χρησιμοποιείται ο αλγόριθμος κρυπτογράφησης και το δημόσιο κλειδί για να ανακτηθεί το μήνυμα m: $m = s^e modn$. Μπορεί πολύ εύκολα να επαληθευτεί ότι $s^e = m^{de} = m$.

2.3 Συναρτήσεις κατακερματισμού

Η συνάρτηση κατακερματισμού ή συνάρτηση σύνοψης (hash function), είναι ένας μαθηματικός αλγόριθμος H με είσοδο ένα σύνολο δεδομένων αυθαίρετου μήκους M, και έξοδο ένα αλφαριθμητικό καθορισμένου μήκους h. Η έξοδος του αλγορίθμου h=H(M) ονομάζεται σύνοψη ή κατακερματισμός (hash) [46].

Οι συναρτήσεις κατακερματισμού χρησιμοποιούνται ευρέως στην κρυπτογραφία κυρίως για την σύνοψη μεγάλων δεδομένων, καθώς έχουν πολύ καλές ιδιότητες:

- Για την ίδια είσοδο, δίνουν την ίδια τιμή hash value.
- Για οποιαδήποτε είσοδο, είναι εύκολο να υπολογιστεί η hash value.
- Είναι μη αντιστρέψιμες, δωσμένης της hash value δεν μπορεί να υπολογιστεί η είσοδος.
- Οποιαδήποτε αλλαγή στην είσοδο, δίνει μια τελείως διαφορετική τιμή hash value.

Και οι ιδιότητες τους ως προς την ασφάλεια έναντι κρυπτογραφικών επιθέσεων:

- Pre-image resistance: Από μια τιμή hash(m) είναι δύσκολο να βρεθεί το αρχικό μήνυμα m.
- Second pre-image resistance: Δεδομένης μιας εισόδου m_1, είναι δύσκολο να βρεθεί δεύτερο m_2 ώστε να έχουν την ίδια τιμή κατακερματισμού

hash(m_1)=hash(m_2).

- Collision resistance: Είναι δύσκολο να βρεθούν δύο διαφορετικά m_1, m_2 ώστε hash(m_1)=hash(m_2).

Κεφάλαιο 3

Κατανεμημένα συστήματα

3.1 Ορισμός

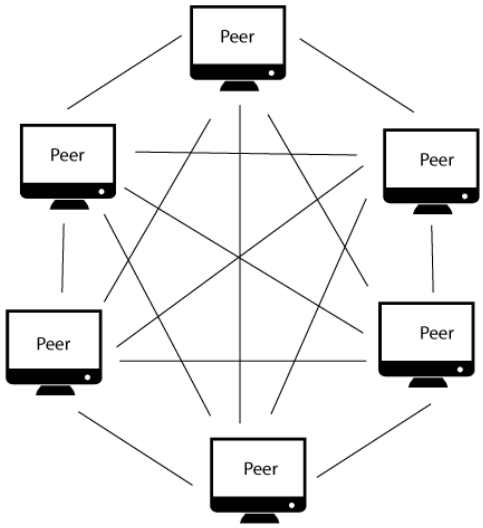
Ως κατανεμημένα συστήματα (distributed systems) ορίζονται τα σύνολα από ανεξάρτητες οντότητες, που συνεργάζονται ώστε να λύσουν κάποιο πρόβλημα που δεν μπορεί να λυθεί από την κάθε οντότητα ανεξάρτητα [41].
Για τα υπολογιστικά συστήματα συγκεκριμένα, ως κατανεμημένο σύστημα θεωρείται ένα σύνολο από υπολογιστές που δεν μοιράζονται κοινή μνήμη ή κοινό φυσικό ρολόι, επικοινωνούν με μηνύματα μέσω ενός δικτύου επικοινωνίας, και κάθε υπολογιστής έχει την δικιά του μνήμη και τρέχει το δικό του λειτουργικό σύστημα. Οι υπολογιστές είναι ημι-αυτόνομοι και χαλαρά συνδεδεμένοι καθώς συνεργάζονται για να αντιμετωπίσουν ένα πρόβλημα συλλογικά [42].

Σε αυτά τα συστήματα, οι χρήστες συμμετέχουν ως κόμβοι (peers) μέσω ενός υπολογιστή. Στην παρούσα διπλωματική, ενδιαφερόμαστε για τα αποκεντρωμένα συστήματα (decentralized systems), που είναι μια υποκατηγορία των κατανεμημένων συστημάτων. Στα αποκεντρωμένα συστήματα δεν υπάρχει κάποιο σταθερό μέρος όπου λαμβάνονται οι αποφάσεις. Κάθε κόμβος παίρνει αποφάσεις ξεχωριστά και η συμπεριφορά του συστήματος χαρακτηρίζεται από το σύνολο αυτών των αποφάσεων. Επιπλέον οι πληροφορίες είναι διανεμημένες στον κόμβους του συστήματος.

Στόχος των αποκεντρωμένων συστημάτων είναι οι κόμβοι που τα αποτελούν, να καταφέρουν να διαμοιράσουν τις πληροφορίες εύκολα, και να πάρουν τις κατάλληλες αποφάσεις συλλογικά. Για να πραγματοποιηθούν τα παραπάνω, θα πρέπει να υπάρχει συναίνεση, παρά τις δυσκολίες μη ύπαρξης έμπιστης κεντρικής αρχής, δυσκολίες επικοινωνίας λόγω ασύγχρονου δικτύου ή διαφόρων σφαλμάτων στους κόμβους.

3.2 Δίκτυα Ομότιμων Κόμβων

Τα αποκεντρωμένα συστήματα βασίζονται στην αρχιτεκτονική των δικτύων ομότιμων κόμβων ή δικτύων γνωστών ως peer-to-peer (P2P). Σύμφωνα με το δίκτυο ομότιμων κόμβων, όλοι οι κόμβοι που το αποτελούν, σχηματίζουν ένα δίκτυο στο οποίο δεν συντονίζονται από κάποια κεντρική αρχή. Κάθε κόμβος μοιράζεται ένα μέρος του υπολογιστικού του εξοπλισμού, ώστε να καλύπτονται οι συνολικές ανάγκες του δικτύου και να πραγματοποιούνται οι κατάλληλες διεργασίες [43]. Κάθε κόμβος συνδέεται με έναν αριθμό άλλων κόμβων ώστε η πληροφορία να διαμοιράζεται εύκολα και γρήγορα.



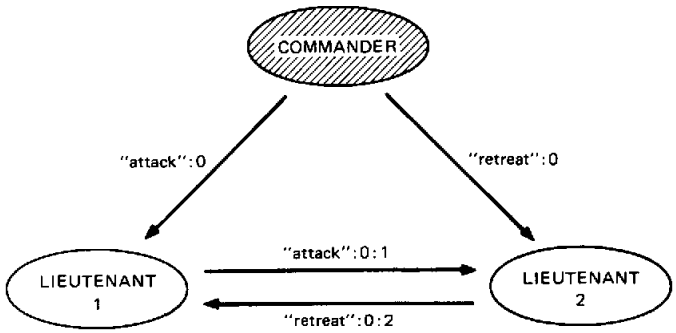
3.3 Βυζαντινά Σφάλματα

Οι γενικότεροι τύποι σφαλμάτων, που μπορεί να προκύψουν σε ένα διανεμημένο σύστημα, είναι γνωστοί ως βυζαντινά σφάλματα. Τα βυζαντινά σφάλματα αποτυπώθηκαν από τους L. Lamport, R. Shostak, και M. Pease στο [7] όπου περιγράφτηκε το Πρόβλημα των Βυζαντινών Στρατηγών. Τέτοια σφάλματα είναι αυτά που μπορεί να προκύψουν από κακόβουλη συμπεριφορά κάποιου κόμβου στο σύστημα, με την έννοια ότι παρεκκλίνει

από τους κανόνες του πρωτοκόλλου. Το σημαντικότερο πρόβλημα που μπορεί να δημιουργηθεί από αυτή την συμπεριφορά είναι να μην μπορούν οι κόμβοι του συστήματος να καταλήξουν σε συμφωνία.

3.3.1 Πρόβλημα Βυζαντινών Στρατηγών

Το πρόβλημα των Βυζαντινών Στρατηγών (BGP) είναι ένα πρόβλημα που περιγράφει την δυσκολία των αποκεντρωμένων συστημάτων να έρθουν σε ομοφωνία χωρίς να εμπιστεύονται κάποια κεντρική αρχή [7]. Έστω ότι υπάρχουν αρκετοί στρατηγοί έξω από μια Βυζαντινή πόλη και την έχουν περικυκλώσει. Θέλουν να αποφασίσουν πότε θα επιτεθούν στην πόλη, αλλά για να γίνει αυτό θα πρέπει να συντονιστούν ώστε να γίνει η επίθεση ταυτόχρονα. Η επικοινωνία τους γίνεται μέσω αγγελιοφόρων. Το πρόβλημα είναι ότι δεν διαθέτουν ασφαλή τρόπο επικοινωνίας, καθώς οποιοσδήποτε μεταφέρει τα μηνύματα των στρατηγών μπορεί να είναι προδότης (βυζαντινός αντίπαλος). Αποδεικνύεται στο [7] πως εάν ένας βυζαντινός αντίπαλος μπορεί να επηρεάσει το $\frac{1}{3}$ των συμμετεχόντων της επικοινωνίας, τότε εμποδίζεται η συμφωνία.



Σχήμα 3.1: Παράδειγμα προσπάθειας ομοφωνίας ανάμεσα σε 3 πρόσωπα εκ των οποίων ένας είναι προδότης [7]

Η παραπάνω εικόνα από το άρθρο [7], δείχνει σχηματικά την προσπάθεια ομοφωνίας μεταξύ ενός στρατηγού και δύο αγγελιοφόρων. Εάν ο στρατηγός είναι μη ειλικρινής, και άρα βυζαντινός αντίπαλος, μπορεί να δώσει διαφορετικές εντολές τους δύο αγγελιοφόρους και άρα να αποτρέψει την ομοφωνία. Πιο συγκεκριμένα, λέει στον ένα εισβολή και στον άλλο υποχώρηση. Όταν λοιπόν αυτοί επικοινωνούν για να έρθουν σε ομοφωνία έχουν διαφορετικές εντολές. Αποδεικνύεται λοιπόν ότι το πρόβλημα των βυζαντινών στρατηγών δεν λύνεται για 3 πρόσωπα στους οποίους οι 2 είναι ειλικρινείς και ο ένας κακόβουλος.

Κεφάλαιο 4

Blockchain

Το blockchain είναι ένα αποδεδειγμένα ασφαλές αποκεντρωμένο σύστημα. Πρόκειται για μια διανεμημένη βάση δεδομένων που υλοποιείται σε ένα δίκτυο ομότιμων κόμβων και βασίζεται σε ένα πρωτόκολλο ομοφωνίας.

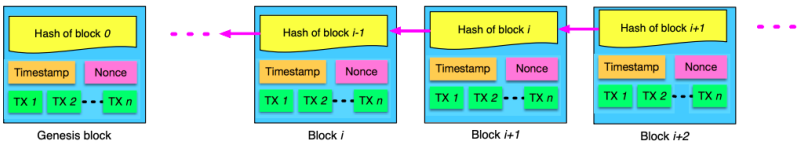
Η ιδέα του blockchain περιγράφτηκε από τον DL. Chaum στην διδακτορική του διατριβή [32] και έπειτα οι S. Haber και WS. Stornetta το 1991 [33], διατύπωσαν μια κρυπτογραφικά ασφαλή σύνδεση δεδομένων. Η ιδέα ήταν ότι έπρεπε να φτιαχτεί ένα σύστημα που οι χρονικές ετικέτες των δεδομένων του συστήματος δεν θα μπορούσαν να παραβιαστούν. Αυτό πραγματοποιήθηκε με την σύνδεση των δεδομένων μέσω των συναρτήσεων κατακερματισμού. Η παραμικρή αλλαγή στα δεδομένα θα έδινε μια διαφορετική τιμή hash. Το 1998, ο N. Szabo πρότεινε την ιδέα ενός αποκεντρωμένου ηλεκτρονικού νομίσματος που θα υλοποιείται με μια βάση δεδομένων σαν αλυσίδα, που τα δεδομένα θα συνδέονται μέσω των τιμών κατακερματισμού hash, τα οποία θα προκύπτουν από την επίλυση ενός κρυπτογραφικού προβλήματος. Έτσι έγινε η πρώτη αναφορά σε ένα blockchain που θα μεγαλώνει με την βοήθεια ενός μηχανισμού που ονομάζεται Proof-of-Work. Ενώ η ιδέα του N. Szabo ήταν πολύ κοντά στην σημερινή υλοποίηση του blockchain, είχε ένα βασικό ελάττωμα: υπήρχε η υπόθεση ότι οι χρήστες που συμμετέχουν στην επίλυση του κρυπτογραφικού προβλήματος είναι έμπιστοι. Ένας χρήστης θα μπορούσε να χρησιμοποιήσει το κρυπτογραφικό πρόβλημα ώστε να φτιάξει μια διακλάδωση της αλυσίδας και άρα να δημιουργήσει σύγχιση στο σύστημα για το ποιό είναι το σωστό κλαδί. Θα πρέπει λοιπόν να υπάρχει ένας τρόπος η αλυσίδα να αυξάνεται γραμμικά ή έστω να μπορεί να διαχωριστεί η ειλικρινής αλυσίδα από τις υπόλοιπες που μπορεί να δημιουργηθούν. Το πρόβλημα επικράτησης μιας ειλικρινώς φτιαχγμένης αλυσίδας ονομάζεται double-spending problem.

Αργότερα, το 2008 εισήχθη από τον/τους Satoshi Nakamoto ένα blockchain

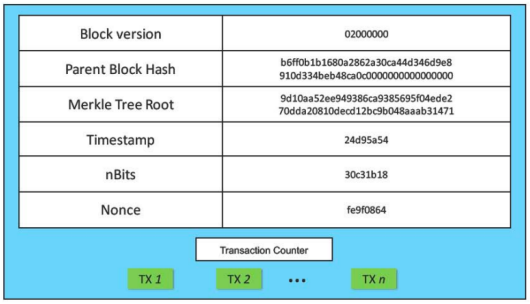
που υλοποιεί ένα αποκεντρωμένο ηλεκτρονικό νόμισμα στο άρθρο "Bitcoin: A Peer-to-Peer Electronic Cash System" [31]. Στο άρθρο αυτό προσδιορίζεται η δομή των block με τις συναλλαγές, η σύνδεση των block με τις συναρτήσεις κατακερματισμού, η χρήση των merkle tree, ο μηχανισμός ομοφωνίας Proof-of-Work και οι αμοιβές που θα δίνονται στους συμμετέχοντες. Επιπλέον αποδεικνύεται η ασφάλεια του πρωτοκόλλου ως προς το πρόβλημα double-spending, με την χρήση των διωνυμικών τυχαίων περιπάτων. Το 2009 έγινε η πρώτη υλοποίηση του από τον S. Nakamoto , και εξήχθη το πρώτο block του Bitcoin που ονομάζεται Genesis Block. Το Bitcoin θεωρείται η απαρχή των blockchain συστημάτων καθώς είναι η πρώτη ολοκληρωμένη και ασφαλής εφαρμογή. Ακόμη και σήμερα, η αρχιτεκτονική των blockchain βασίζεται σε αυτή που προτάθηκε το 2008 από τον S. Nakamoto και αποτελείται από τα εξής βασικά επίπεδα: Επίπεδο Δεδομένων, Επίπεδο Δικτύου, Επίπεδο Ομοφωνίας, Επίπεδο Αμοιβών. Στην συνέχεια με την εξέλιξη των αποκεντρωμένων τεχνολογιών blockchain προστέθηκαν και άλλα επίπεδα, όπως το Επίπεδο Έξυπνων Συμβολαίων.

4.1 Επίπεδο Δεδομένων

Το blockchain αποτελεί μια ακολουθία από block, τα οποία περιέχουν ένα σύνολο δεδομένων. Κάθε block συνδέεται με το ακριβώς προηγούμενο του με έναν δείκτη, που είναι η τιμή κατακερματισμού του προηγούμενου block. Κάθε block έχει ένα δείκτη και άρα μόνο ένα προγούμενο block. Το πρώτο block της αλυσίδας αυτής ονομάζεται Genesis Block. Παρακάτω παρουσιάζεται μια σχηματική αναπαράσταση ενός blockchain (σχήμα 3.1):



Σχήμα 4.1: Παράδειγμα blockchain [34]



Σχήμα 4.2: Παράδειγμα block [34]

Όπως φαίνεται στο σχήμα 3.2, κάθε block έχει συγκεκριμένη δομή. Περιλαμβάνει:

1. Την έκδοση του block
2. Τον δείκτη hash του προηγούμενου block
3. Το merkle root hash των δεδομένων του
4. Τον χρόνο δημιουργίας του
5. Μια ψευδοτυχαία τιμή (nonce)
6. Τα δεδομένα (στην περίπτωση υλοποίησης κρυπτονομίσματος, τις συναλλαγές)

Η συγκεκριμένη δομή του block εξασφαλίζει την ακεραιότητα των δεδομένων του, και όσα περισσότερα block προστίθονται στο blockchain, τόσο πιο ισχυρή δομή δεδομένων γίνεται. Οποιαδήποτε αλλαγή σε κάποιο από τα περιεχόμενα ενός block, σημαίνει αλλαγή και στην τιμή κατακερματισμού του (hash), και άρα οδηγεί σε μια μη έγκυρη αλυσίδα. Τα δεδομένα ενός block μπορούν να επαληθευτούν εύκολα από το δέντρο Merkle Tree, δωσμένης της ρίζας του (merkle tree root hash).

4.2 Επίπεδο Δικτύου

Το δίκτυο που υλοποιεί την blockchain δομή είναι ένα δίκτυο ομότιμων κόμβων, peer-to-peer, όπως περιγράφτηκε στα διανεμημένα συστήματα. Ένας από τους παράγοντες που επηρεάζουν την επικοινωνία των χρηστών μέσω του peer-to-peer δικτύου, είναι ο συγχρονισμός του συστήματος. Όταν φτιάχνεται ένα πρωτόκολλο, βασίζεται σε μια υπόθεση συγχρονισμού, που αφορά τον κυρίως χρόνο διάδοσης των μηνυμάτων στο δίκτυο. Τα συστήματα διαχωρίζονται σε συγχρονισμένα, μερικώς συγχρονισμένα ή ασύγχρονα. Στα ασύγχρονα δίκτυα δεν υπάρχουν όρια στο χρονικό διάστημα διάδοσης των μηνυμάτων, και άρα το σύστημα δεν μπορεί να λειτουργήσει σωστά καθώς υπάρχουν σοβαρές καθυστερήσεις στην επικοιν-

ωνία. Στα συγχρονισμένα συστήματα, υπάρχουν συγκεκριμένα όρια στην παράδοση των μηνυμάτων τα οποία δεν μπορούν να παραβιαστούν. Στα συστήματα αυτά είναι πολύ πιο εύκολη η επίτευξη ομοφωνίας. Στα μερικώς συγχρονισμένα δίκτυα, μπορεί να υπάρχουν καθυστερήσεις αλλά είναι πάντα ελεγχόμενες. Στην συνέχεια, στην ανάλυση των πρωτοκόλλων Algorand [3] και Ouroboros [1], [2] παρουσιάζονται υποθέσεις τόσο συγχρονισμένων, όσο και μερικώς συγχρονισμένων δικτύων.

4.3 Επίπεδο Ομοφωνίας

Τα πρωτόκολλα ομοφωνίας διαχειρίζονται την προσθήκη των δεδομένων στην δομή του blockchain. Αν και το blockchain είναι ένα κατανεμημένο σύστημα με χρήστες από όλο τον κόσμο, τα πρωτόκολλα ομοφωνίας εξασφαλίζουν την ακεραιότητα και την σταθερότητα των δεδομένων [35]. Το πρώτο είδος ομοφωνίας που προτάθηκε για το blockchain είναι ο αλγόριθμος Proof-of-Work (PoW). Ήδη όμως μέσα από την εφαρμογή του στον bitcoin εμφανίστηκαν προβλήματα, και σε συνδυασμό με την έκταση που πήρε η τεχνολογία του blockchain, δημιουργήθηκαν και άλλοι αλγόριθμοι ομοφωνίας με τα δικά τους προτερήματα-ελαττώματα. Στην συνέχεια θα παρουσιαστούν κάποιοι από τους πιο γνωστούς αλγόριθμους ομοφωνίας.

4.3.1 Proof-of-Work

Ο αλγόριθμος ομοφωνίας Proof-of-Work ήταν από τους πρώτους που χρησιμοποιήθηκαν με την εμφάνιση του blockchain στο Bitcoin [31] και στο Ethereum [36]. Σύμφωνα με τον αλγόριθμο αυτό, κάθε κόμβος του συστήματος υπολογίζει μια τιμή κατακερματισμού (hash value), με το block header του προηγούμενου block και με μια τυχαιότητα (nonce). Η hash value που προκύπτει θα πρέπει να είναι μικρότερη ή ίση με τον στόχο, μια τιμή που βρίσκεται στο block και έχει συγκεκριμένη δυσκολία υπολογισμού. Όταν κάποιος χρήστης φτάσει τον στόχο, και οι υπόλοιποι χρήστες το επιβεβαιώσουν, προσθέτει ένα καινούριο block στην αλυσίδα. Οι κόμβοι που υπολογίζουν τις διάφορες hash values ονομάζονται miners και η διαδικασία mining. Σε ένα αποκεντρωμένο σύστημα, μπορεί να παραχθούν έγκυρα block ταυτόχρονα για το ίδιο προηγούμενο block. Το αποτέλεσμα είναι να δημιουργηθούν διακλαδώσεις (forks) στο σύστημα, και το πρόβλημα αυτό λύνεται με το να επιλέγεται η μεγαλύτερη διακλάδωση. Τα δεδομένα θεωρούνται ασφαλή στο blockchain, όταν βρίσκονται σε block που είναι αρκετά βαθιά στην αλυσίδα. Για παράδειγμα στο Bitcoin απαιτούνται 6 block ώστε μια αλυσίδα να προτιμάται σε σχέση με τις άλλες. Για να γίνει το mining απαιτούνται πολλοί υπολογισμοί, οι οποίοι γίνονται με brute force, δηλαδή βρίσκεται το hash μιας τιμής με μια αρ-

χική τυχαιότητα και ελέγχεται εάν αυτή η τιμή είναι μικρότερη του στόχου. Εάν δεν είναι, συνεχίζεται ο υπολογισμός με μια διαφορετική τυχαιότητα. Ο στόχος καθορίζει την δυσκολία εύρεσης, όσο πιο μικρό νούμερο είναι τόσο πιο δύσκολο να βρεθεί. Αυτό είναι ένα από τα βασικότερα προβλήματα του Proof-of-Work: απαιτούνται μεγάλα ποσά ενέργειας για να είναι ασφαλές το blockchain.

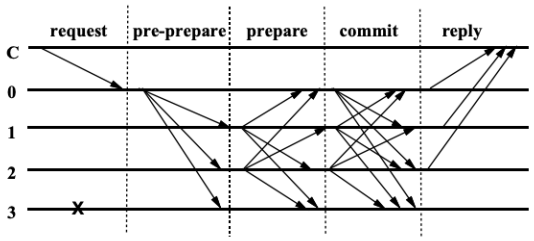
4.3.2 Proof-of-Stake

Ο αλγόριθμος Proof-of-Stake (PoS) είναι μια εναλλακτική πρόταση έναντι του Proof-of-Work. Πολλά πρωτόκολλα blockchain έχουν δημιουργηθεί με βάση τον αλγόριθμο Proof-of-Stake και έχουν αποδειχθεί ασφαλή, με χαρακτηριστικά παραδείγματα τα πρωτόκολλα Ouroboros [1], [2] και το Snow-White [37]. Πρόκειται για έναν οικολογικό αλγόριθμο που δεν απαιτεί οι miners να σπαταλήσουν υπολογιστική ενέργεια για να επικυρώσουν συναλλαγές και να προσθέσουν block, αλλά επιλέγονται βάσει του ποσού των χρημάτων που διαθέτουν στον λογαριασμό τους. Επειδή η επιλογή αυτή είναι άδικη για όσους διαθέτουν έναν μέτριο λογαρισμό, ωφελώντας αυτόν με τα περισσότερα χρήματα διότι θα επιλέγεται μόνιμα, η επιλογή γίνεται τυχαία αλλά πάντα σύμφωνα με την οικονομική του δύναμη στο blockchain. Ο αλγόριθμος PoS είναι μια έξυπνη αντιπρόταση καθώς δίνει κίνητρο καλής συμπεριφοράς, αφού οι χρήστες που προσθέτουν τα νέα block εάν σαμποτάρουν το πρωτόκολλο, τότε κινδυνεύουν να χάσουν τα χρήματα τους. Από την άλλη πλευρά, επειδή οι χρήστες δεν χρειάζεται να λύσουν κάποιο κρυπτογραφικό πρόβλημα όπως στον PoW, είναι πολύ εύκολο κάποιος να επιτεθεί σε ένα PoS πρωτόκολλο καθώς δεν θα του κοστίσει. Αυτό το πρόβλημα ονομάζεται nothing-at-stake. Επιπλέον, υπάρχει κίνδυνος ένα PoS blockchain να μην είναι αποκεντρωμένο εάν η κατανομή των χρημάτων στους χρήστες δεν είναι σχετικά ισότιμη. Γενικότερα όμως η πρόταση Proof-Of-Stake είναι μια καλή λύση για το πρόβλημα της ομοφωνίας και όλο και περισσότερα blockchain μεταβαίνουν από Proof-of-Work σε Proof-of-Stake, με χαρακτηριστικότερο παράδειγμα το Ethereum [38].

4.3.3 PBFT

Το PBFT είναι η συντομογραφία του Practical Byzantine Fault Tolerance, και αποτελεί έναν αλγόριθμο που παρέχει ανοχή σε σφάλματα βυζαντινού τύπου. Το πρόβλημα βυζαντινών στρατηγών ισχύει και για την τεχνολογία blockchain, αφού πρόκειται για ένα αποκεντρωμένο σύστημα στο οποίο θα πρέπει να ληφθεί μια κοινή απόφαση. Ο αλγόριθμος PBFT έχει αποδειχθεί ότι λύνει αποδοτικά το πρόβλημα των βυζαντινών στρατηγών [39], και προτάθηκε το 1999 από τους Miguel Castro και Barbara Liskov.

Πρόκειται για ένα πρωτόκολλο που αποτελείται από τρείς φάσεις pre-prepare, prepare και commit, και υπάρχει ένας βασικός κόμβος που δέχεται αιτήματα και τα προωθεί στους υπόλοιπους. Στην πρώτη φάση, ο βασικός κόμβος δέχεται ένα αίτημα, το επιβεβαιώνει, και στην συνέχεια το προωθεί στους υπόλοιπους. Οι υπόλοιποι κόμβοι αφού δεχτούν ψήφους τουλάχιστον από τα $\frac{2}{3}$ των κόμβων του δικτύου, κάνουν commit, και ο βασικός κόμβος ολοκληρώνει την αλγόριθμο ενημερώνοντας για την ομοφωνία. Σχηματικά η διαδικασία παρουσιάζεται και στο [39], όπου εφαρμόζεται ο PBFT για την ομοφωνία σε ένα δίκτυο 4 κόμβων εκ των οποίων ο ένας έχει επιδείξει βυζαντινή αποτυχία :



Σχήμα 4.3: Αλγόριθμος PBFT με έναν κόμβο σφάλματος βυζαντινού τύπου [39]

Όπως έχει αποδειχθεί και στο BGP, ο αλγόριθμος είναι ασφαλής και φέρνει την ομοφωνία στο σύστημα μόνο εάν τα $\frac{2}{3}$ των χρηστών είναι ειλικρινείς, και άρα μόνο εάν έχουν ληφθεί τουλάχιστον τα $\frac{2}{3}$ των ψήφων σε κάθε βήμα. Ο αλγόριθμος PBFT είναι χρήσιμος για ομοφωνία επιτροπών που αποτελούνται από μικρό αριθμό κόμβων, και παρατηρείται πως διαφοροποιείται από τους άλλους μηχανισμούς συναίνεσης (PoW, PoS) καθώς η διαδικασία ομοφωνίας βασίζεται στην ολοκλήρωση φάσεων ψηφοφορίας. Η πιο γνωστή και άμεση εφαρμογή του PBFT είναι το Hyperledger Fabric της IBM [40].

Κεφάλαιο 5

Verifiable Random Functions

5.1 Εισαγωγή

Η έννοια των Verifiable Random Functions (VRFs) εισήχθη πρώτα από τους Silvio Micali, Michael Rabin και Salil Vadhan στο κοινό τους άρθρο [6] το 1999. Η δημιουργία τους προέκυψε ως ανάγκη για ψευδοτυχαίες συναρτήσεις που θα έχουν επαληθευόμενο αποτέλεσμα. Πρόκειται για συναρτήσεις που δεδομένου ενός τυχαίου string που ονομάζεται seed, θα παράγεται ψευδοτυχαίο αποτέλεσμα που θα μπορεί να αποδειχθεί σωστό χωρίς να επηρεάζεται η προβλεψιμότητα του. Η απόδειξη του ψευδοτυχαίου αποτελέσματος θα πρέπει να είναι μοναδική. Ένα τέτοιο μαθηματικό μοντέλο ονομάζεται Verifiable Random Function ή αλλιώς VRF.

5.2 Περιγραφή

Οι ψευδοτυχαίες συναρτήσεις επαληθευόμενου αποτελέσματος (VRFs) προέρχονται από ένα ζευγάρι δημοσίου και ιδιωτικού κλειδιού (pk, sk), με την ιδιότητα το ιδιωτικό κλειδί να είναι δύσκολο να παραχθεί από το δημόσιο κλειδί. Οι VRFs μπορούν να παρομοιασθούν με συναρτήσεις κατακερματισμού δημοσίου κλειδιού (*public keyed hash functions*), αφού παράγεται ένα ψευδοτυχαίο hash αποτέλεσμα με είσοδο το ιδιωτικό κλειδί $(f_{sk}(x))$, αλλά οποιοσδήποτε ξέρει το αντίστοιχο δημόσιο κλειδί (pk) μπορεί να το επαληθεύσει. Προκειμένου να οριστεί σωστά η συνάρτηση, υπάρχει ένας αλγόριθμος που παράγει το ζευγάρι των κλειδιών (*generation algorithm*).

Προκειμένου να γίνεται η επαλήθευση της hash τιμής του αλγορίθμου $f_{sk}(x)$, χρειάζεται να παραχθεί μια απόδειξη $proof_x$ μαζί με το ζητούμενο αποτέλεσμα της συνάρτησης. Ο αλγόριθμος που πραγματοποιεί αυτή

την διαδικασία ονομάζεται *evaluation algorithm.* Επιπλέον, θα πρέπει να υπάρχει και ένας αλγόριθμος που θα επιτρέπει την απόδειξη ορθότητας σε όποιον έχει το x, το pk_f, το $f_{sk}(x)$ και το $proof_x$. Αυτός ονομάζεται αλγόριθμος επαλήθευσης (*verification algorithm*).

5.3 Ιδιότητες

Οι Verifiable Random Functions είναι σχεδιασμένες έτσι ώστε να ικανοποιούν τις παρακάτω ιδιότητες :

1) ***Uniqueness*** :
Σύμφωνα με αυτή την ιδιότητα, ο αλγόριθμος υπολογισμού, για συγκεκριμένο ζευγάρι κλειδιών και συγκεκριμένη είσοδο x, θα επιστρέφει πάντα το ίδιο επαληθεύσιμο αποτέλεσμα, ακόμα και αν ο χρήστης που θα το επαληθεύσει γνωρίζει το ιδιωτικό κλειδί.

2) ***Collision resistance*** :
Είναι απαραίτητο για τις ψευδοτυχαίες συναρτήσεις επαληθευόμενου αποτελέσματος, όπως και για κάθε ψευδοτυχαία συνάρτηση, να είναι δύσκολο για δύο διαφορετικές εισόδους x_1 και x_2 να επιστραφεί το ίδιο ψευδοτυχαίο αποτέλεσμα. Αυτό θέλουμε να ισχύει ακόμα και στην περίπτωση που έχει διαρρεύσει το ιδιωτικό κλειδί (sk).

3) ***Unique provability*** :
Η ιδιότητα αυτή αφορά τον αλγόριθμο επαλήθευσης, και δηλώνει πως για δύο οποιαδήποτε διαφορετικά αποτελέσματα $f_{sk}(x_1)$ και $f_{sk}(x_2)$, θα πρέπει να υπάρχουν δύο διαφορετικές αποδείξεις επαλήθευσης $proof_{x_1}$ και $proof_{x_2}$. Η πιθανότητα, δηλαδή, να επαληθευτούν με την ίδια απόδειξη $proof$ πρέπει να είναι αμελητέα.

4) ***Pseudorandomness*** :
Η ιδιότητα αυτή δηλώνει πως για οποιονδήποτε χρήστη δεν γνωρίζει την απόδειξη αποτελέσματος των Verifiable Random Functions ($proof$), τότε δεν μπορεί να διακρίνει το αποτέλεσμα από μια εντελώς τυχαία τιμή. Είναι σημαντικό η τιμή αυτή να φαίνεται τυχαία μόνο για τους χρήστες που δεν γνωρίζουν το ιδιωτικό κλειδί ή την απόδειξη $proof$ των VRFs. Μια σημαντική προϋπόθεση για όλα αυτά είναι η παραγωγή του ζεύγους των κλειδιών του χρήστη να έχει γίνει με αξιόπιστο τρόπο.

   - ***Unpredictability under malicious key generation*** :
   Στην περίπτωση που η παραγωγή των κλειδιών δεν έχει γίνει με αξιόπιστο τρόπο, υπάρχει και η περίπτωση να εξασφαλίζεται ψευδοτυχαιότητα. Μια τέτοια ιδιότητα είναι σημαντική στις περιπτώσεις που οι ψευδοτυχαίες συναρτήσεις επαληθευόμενου αποτελέσματος χρησιμοποιούνται σε πρωτόκολλα ομοφωνίας, στα οποία η πιθανότητα επιλογής συγκεκριμένων κλειδιών με σκοπό την οποιαδήποτε επιρροή στο αποτέλεσμα είναι μεγάλη.

5.4 Ορισμός

Ορισμός 5.4.1. Έστω G,F και V αλγόριθμοι πολυωνυμικού χρόνου ώστε:

- G (γεννήτρια συνάρτηση) πρόκειται για έναν πιθανολογικό αλγόριθμο που δέχεται σαν είσοδο ένα μοναδιαίο string και επιστρέφει δύο δυαδικά strings. Η είσοδος είναι μια παραμετρος ασφάλειας k, ενώ η έξοδος είναι το ζευγάρι δημοσίου - ιδιωτικού κλειδιού (pk,sk)
- $F = (F_1, F_2)$ (εκτιμήτρια συνάρτηση) πρόκειται για έναν ντετερμινιστικό αλγόριθμο με είσοδο δύο δυαδικά strings και έξοδο δύο δυαδικά strings. Η είσοδος είναι το ιδιωτικό κλειδί (sk) και ένα αλφαριθμητικό x, ενώ η έξοδος είναι η τιμή $F_1(sk, x)$ (το ψευδοτυχαίο αποτέλεσμα των VRFs) και η τιμή $F_2(sk, x) = proof$ (η απόδειξη για την επαλήθευση της $F_1(sk, x)$)
- V (συναρτηση επαλήθευσης) πρόκειται για έναν πιθανολογικό αλγόριθμο που δέχεται σαν είσοδο τέσσερα δυαδικά strings και έχει δύο πιθανές εξόδους YES/NO. Η είσοδος στον αλγόριθμο είναι το δημόσιο κλειδί (pk), το αλφαριθμητικό x, μια τιμή u και η απόδειξη proof.

'Εστω a: $\mathbb{N} \to \mathbb{N} \cup \{*\}$ και b,s: $\mathbb{N} \to \mathbb{N}$, τρείς συναρτήσεις πολυωνυμικού χρόνου $poly(k)$,όπου k είναι η παράμετρος ασφάλειας. Τότε λέμε ότι η τριπλέτα συναρτήσεων (G,F,V) ονομάζονται ***Verifiable Pseudorandom Functions , VRFs*** με είσοδο μήκους a(k), έξοδο μήκους b(k) και ασφάλεια s(k), εαν ισχύουν οι παρακάτω ιδιότητες :

a) Για κάθε $x \epsilon \{0,1\}^{a(k)}$ ισχύει:

$$Pr[F_1(sk, x)\ \epsilon \{0,1\}^{b(k)}] = 1 - 2^{-\Omega(k)}$$

b) Για κάθε $x \epsilon \{0,1\}^{a(k)}$, εάν $(u, proof) = F(sk, x)$ τότε:

$$Pr[V(pk, x, u, proof) = YES] > 1 - 2^{-\Omega(k)}$$

c) Για κάθε $pk, x, u_1, u_2, proof_1, proof_2$ με $u_1 \neq u_2$ τότε για $i = 1$ ή $i = 2$ ισχύει :

$$Pr[V(pk, x, u_i, proof_i) = YES] < 2^{-\Omega(k)}$$

d) Έστω ένας αλγόριθμος T με πρόσβαση στην εκτιμήτρια συνάρτηση $F = (F_1, F_2)$.
Ο T αποτελείται από δύο μέρη, το T_1, ένας αλγόριθμος που λαμβάνει τα δεδομένα από την F μέσω του O_f, και το T_2, αλγόριθμος που κρίνει εάν τα δεδομένα προήλθαν από την f ή είναι τυχαίες τιμές. Οι αλγόριθμοι $T = (T_1, T_2)$ τρέχουν με το πολύ $s(k)$ βήματα με αρχική είσοδο το 1^k.
Εκτελούμε το παρακάτω πείραμα, με πιθανότητα επιτυχίας του T το πολύ $\frac{1}{2} + \frac{1}{s(k)}$:

Algorithm 1: Efficient statistical test for VRFs

$(pk, sk) \leftarrow G(1^k)$

$(x, st) \leftarrow T_1^{F(sk,.)}(pk, 1^k)$

choose $r \xleftarrow{R} \{0,1\}$

if $r = 0$ let $u = F_1(sk, x)$

if $r = 1$ **choose** $u \xleftarrow{R} \{0,1\}^{b(k)}$

$guess \leftarrow T_2^{F(sk,.)}(u, 1^k, st)$

if $x \epsilon \{0,1\}^{a(k)}, guess == r$ and x was not queried before, T *succeeds.*

Παρατήρηση 5.4.1. Το πείραμα που πραγματοποιείται μπορεί να περιγραφτεί ως εξής : Ο αλγόριθμος T_1 χρησιμοποιώντας το δημόσιο κλειδί pk λαμβάνει πληροφορίες για την συνάρτηση f, δοκιμάζοντας διάφορες εισόδους λαμβάνει την έξοδο των VRFs και την αντίστοιχη απόδειξη. Μετά την συλλογή των πληροφοριών δίνοντας σαν είσοδο μια τιμή x, που δεν έχει ξαναχρησιμοποιήσει, αποκτά την τιμή u. Στρίβοντας ένα νόμισμα, δίνουμε στην τιμή u το αποτέλεσμα των VRFs, εάν το νόμισμα φέρει την τιμή "0", αλλιώς μια τυχαία τιμή από το σύνολο $\{0,1\}^{b(k)}$. Ο δεύτερος αλγόριθμος T_2 θα κρίνει εάν η τιμή u προήλθε από τις VRFs ή είναι μια τυχαία τιμή.

Ορισμός 5.4.2. Ως ***Verifiable Unpredictable Functions, VUF's*** ονομάζουμε μια τριπλέτα συναρτήσεων (G,F,V), εισόδου $a(k)$, εξόδου $b(k)$ και ασφάλειας $s(k)$, που ορίζονται ακριβώς όπως οι VRF's, αλλά με την ακόλουθη ιδιότητα μη προβλεψιμότητας αντί για την ψευδοτυχαιότητα:

c) Έστω ένας αλγόριθμος T με πρόσβαση στην εκτιμήτρια συνάρτηση $F = (F_1, F_2)$, πολυωνυμικού χρόνου $s(k)$ όταν δέχεται σαν πρώτη είσοδο το 1^k. Εκτελούμε το παρακάτω πείραμα, με πιθανότητα επιτυχίας του T το πολύ $\dfrac{1}{s(k)}$:

Algorithm 2: Efficient statistical test for VUFs

$(pk, sk) \leftarrow G(1^k)$

$(x, guess) \leftarrow T^{F(sk,.)}(1^k, pk)$

if $x \epsilon \{0,1\}^{a(k)}, guess == F_1(sk, x)$ and x was not queried before, T *succeeds.*

Παρατήρηση 5.4.2. Η μη προβλεψιμότητα (unpredictability), είναι μια ασθενέστερη ιδιότητα από την ψευδοτυχαιότητα (pseudorandomness) που έχουν οι VRF's. Πιο συγκεκριμένα, στο πείραμα των VUF's ζητείται απο τον αλγόριθμο T να υπολογιστεί η σωστή απάντηση $(guess == F_1(sk, x))$, ενώ στο πείραμα των VRF's ζητείται να διαχωριστεί η απάντηση από μια εντελώς τυχαία τιμή.

5.5 Θεωρία

Οι ψευδοτυχαίες συναρτήσεις είναι ένα πολύ βασικό κρυπτογραφικό εργαλείο. Σύμφωνα με το άρθρο [11] των Goldreich, Goldwasser και Micali το 1986, μπορούμε να κατασκευάσουμε ψευδοτυχαίες συναρτήσεις χρησιμοποιώντας ψευδοτυχαίες γεννήτριες. Για να μπορέσουμε να κατασκευάσουμε ψευδοτυχαίες γεννήτριες και άρα ψευδοτυχαίες συναρτήσεις χρησιμοποιούμε τις έννοιες των μονόδρομων συναρτήσεων, ή αλλιώς one-way functions, και των hardcore bit predicates. Σε αυτή την ενότητα λοιπόν, θα παρουσιαστεί η απαραίτητη θεωρία προκειμένου να γίνει κατανοητός ο τρόπος κατασκευής των VRFs.

Ορισμός 5.5.1 ([29])**.** Μια συνάρτηση $f : \{0,1\}^* \to \{0,1\}^*$ ονομάζεται ***μονόδρομη*** ή ***one-way function*** εάν ισχύουν οι εξής ιδιότητες :

- Είναι εύκολο να υπολογιστεί. Δηλαδή υπάρχει πολυωνυμικού χρόνου αλγόριθμος Π_f , ώστε $\Pi_f(x) = f(x)$, για κάθε x.
- Είναι δύσκολο να αντιστραφεί. Δηλαδή για κάθε PPT αλγόριθμο A:
$$Pr[Invert_{A,f}(n) = 1] \leq negl(n)$$

Ορισμός 5.5.2. Μια συνάρτηση $f : \{0,1\}^* \to f : \{0,1\}^*$ λέμε ότι *διατηρεί το μήκος* εάν $|f(x)| = |x|$, για κάθε $x \epsilon \{0,1\}^*$

Ορισμός 5.5.3. Μια συνάρτηση $f : \{0,1\}^* \to f : \{0,1\}^*$ λέμε ότι είναι *"ένα προς ένα"* ή *1-1* εάν για κάθε $x_1, x_2 \epsilon \{0,1\}^*$ με $f(x_1) = f(x_2) \Rightarrow x_1 = x_2$

Ορισμός 5.5.4 ([29])**.** Μια μονόδρομη συνάρτηση $f : \{0,1\}^* \to \{0,1\}^*$ ονομάζεται ***μονόδρομη μετάθεση*** ή ***one-way permutation*** εάν:

- Είναι συνάρτηση που διατηρεί το μήκος
- Είναι 1-1

Η χρησιμότητα των μονόδρομων συναρτήσεων είναι μεγάλη, καθώς έχουν την ιδιότητα να υπολογίζονται εύκολα αλλά να αντιστρέφονται πολύ δύσκολα (υπολογιστικά αδύνατη αντιστροφή). Για αυτούς τους λόγους χρησιμοποιούνται ως βάση για την κατασκευή κρυπτογραφικών εργαλείων όπως οι *μονόδρομες συναρτήσεις συμπίεσης* (one-way hash functions) και οι *ψευδοτυχαίες συναρτήσεις* (pseudorandom functions). Θα εστιάσουμε στο θεωρητικό υπόβαθρο που χρειάζεται για την κατασκευή ψευδοτυχαίων συναρτήσεων.

Όπως προείπαμε, δωσμένου $y = f(x)$ μιας μονόδρομης συνάρτησης f, είναι υπολογιστικά δύσκολο να βρεθεί η τιμή x, διότι απαιτείται εκθετικός

χρόνος για τον υπολογισμό αυτό. Παρ' όλα αυτά, είναι δυνατόν με αλγόριθμο πολυωνυμικού χρόνου να υπολογιστεί ένα μέρος της τιμής x. Για παράδειγμα, για μια μονόδρομη συνάρτηση f με $g(x_1, x_2) = (x_1, f(x_2))$ και $|x_1| = |x_2|$, είναι εύκολο να αποδειχθεί ότι η g είναι επίσης μια μονόδρομη συνάρτηση παρότι γνωρίζουμε μέρος της εισόδου (ένας αλγόριθμος πολυωνυμικού χρόνου για να μπορέσει να αντιστρέψει την g θα πρέπει να μπορεί να αντιστρέψει πρώτα την f). Θα πρέπει, άρα, να υπάρχει και ένα μέρος της εισόδου που δεν θα μπορεί να υπολογιστεί πολυωνυμικά. Η πληροφορία αυτή της εισόδου x μιας μονόδρομης συνάρτησης f μπορεί να περιοριστεί σε ένα μόνο bit, και η δυαδική συνάρτηση που δίνει αυτό το bit ονομάζεται *hardcore predicate*. Πιο συγκεκριμένα, για ένα hardcore predicate δωσμένης της αντίστοιχης μονόδρομης συνάρτησης f, το να υπολογιστεί το κρυμμένο bit είναι όσο δύσκολο να υπολογιστεί η είσοδος x της $f(x)$. Οι hardcore predicates είναι πολύ σημαντικές συναρτήσεις καθώς έχουν τελείως απρόβλεπτο αποτέλεσμα, που μοιάζει ομοιόμορφα επιλεγμένο.

Ορισμός 5.5.5 ([29])**.** Μια δυαδική συνάρτηση $P : \{0,1\}^* \to \{0,1\}$ λέμε ότι είναι ***hardcore predicate*** μιας μονόδρομης συνάρτησης $f : \{0,1\}^* \to \{0,1\}^*$ εάν ισχύουν τα παρακάτω:

- Δωσμένης της εισόδου x, η συνάρτηση $P(x)$ μπορεί να υπολογιστεί σε πολυωνυμικό χρόνο $poly(n)$.
- Για κάθε PPT αλγόριθμο εισόδου $f(x), x\epsilon\{0,1\}^n$ και εξόδου $\epsilon\{0,1\}$ υπάρχει αμελητέα συνάρτηση $negl$ ώστε :

$$Pr_{x\leftarrow\{0,1\}^n}[A(f(x), 1^n) = P(x)] \leq \frac{1}{2} + negl(n)$$

Η ύπαρξη hardcore predicate *για κάθε* μονόδρομη συνάρτηση δεν έχει αποδειχθεί ακόμα και είναι ακόμα ένα ανοιχτό ερώτημα, αλλά για κάθε μονόδρομη συνάρτηση υπάρχει μια άλλη μονόδρομη συνάρτηση με hardcore predicate. Πιο αναλυτικά, με την προϋπόθεση ότι υπάρχουν οι μονόδρομες συναρτήσεις τότε αποδεικνύεται πως δωσμένης μιας μονόδρομης συνάρτησης f μπορεί να φτιαχτεί μια διαφορετική μονόδρομη συνάρτηση g με αντίστοιχο hardcore predicate. Αυτό αποδείχθη από τους Goldreich και Levin στο κοινό τους άρθρο [22] το 1989.

Θεώρημα 5.5.1 (**Goldreich-Levin**)**.** *Έστω ότι υπάρχει μια μονόδρομη συνάρτηση f, τότε μπορούμε να φτιάξουμε μια άλλη μονόδρομη συνάρτηση g και μια hardcore predicate για την g.*
Έστω $f : \{0,1\}^* \to \{0,1\}^*$ μια μονόδρομη συνάρτηση. Τότε ορίζουμε $g : \{0,1\}^{2n} \to \{0,1\}^{2n}$ με $g(x||r) = (f(x)||r)$ $(|x| = |r| = n)$ και $hp : \{0,1\}^{2n} \to \{0,1\}$ με $hp(x||r) =< x, r >$.
Τότε η g είναι μονόδρομη συνάρτηση και η hp είναι μια hardcore predicate για την g.

Έστω I_k το σύνολο όλων των k-bit strings, και H_k το σύνολο όλων των

συναρτήσεων τέτοιων ώστε $H_k : I_k \to I_k$.

Ορισμός 5.5.6. ***Poly-random collection*** ονομάζεται μια συλλογή συναρτήσεων $F = \{F_k\}$ με τις εξής ιδιότητες :

- *Indexing*: Κάθε συνάρτηση στο F_k σχετίζεται με έναν μοναδικό k-bit δείκτη $F_k = \{f_i | i \epsilon I_k\}$.
- *Poly-time Evaluation*: Υπάρχει ένας αλγόριθμος πολυωνυμικού χρόνου ώστε για κάθε $k \geq 1$ με είσοδο $i \epsilon I_k$ και όρισμα $x \epsilon I_k$, υπολογίζει το $f_i(x)$.
- *Pseudorandomness*: Δεν υπάρχει PPT αλγόριθμος, πολυωνυμικού χρόνου k, που να μπορεί να διακρίνει τις συναρτήσεις της F_k από τις συναρτήσεις H_k.

Ορισμός 5.5.7. Ένας ***πολυωνυμικού χρόνου στατιστικός έλεγχος συναρτήσεων***, είναι ένας πολυωνυμικού χρόνου αλγόριθμος T ώστε, για είσοδο k και με πρόσβαση στο oracle O_f συνάρτησης $f : I_k \to I_k$, επιστρέφει 0 ή 1.

Ορισμός 5.5.8. Έστω μια συλλογή συναρτήσεων $F = \{F_k\}$ και ένας πολυωνυμικού χρόνου στατιστικός έλεγχος συναρτήσεων T. Λέμε ότι ***n*** F ***περνάει τον έλεγχο*** T εάν για οποιοδήποτε πολυώνυμο Q, για κάθε αρκετά μεγάλο k ισχύει :

$$|p_k^F - p_k^H| < \frac{1}{Q(k)},$$

όπου p_k^F είναι η πιθανότητα ο έλεγχος T να επιστρέψει 1 για είσοδο k και πρόσβαση σε oracle O_f συνάρτησης $f \epsilon F_k$, και p_k^H είναι η πιθανότητα ο έλεγχος T να επιστρέψει 1 για είσοδο k και πρόσβαση σε oracle O_f συνάρτησης $f \epsilon H_k$.

Επι της ουσίας, συγκρίνεται μια συνάρτηση f με μια εντελώς τυχαία συνάρτηση, μέσω του αλγορίθμου T. Ο αλγόριθμος αυτός λαμβάνει δεδομένα για την f μέσω του oracle O_f με το δικαίωμα να επιλέξει όποια ορίσματα για την f θέλει. Ο T δεν γνωρίζει εαν η συνάρτηση ανήκει στην συλλογή των F ή στις τυχαίες συναρτήσεις H_k. Αφού συγκεντρωθούν τα δεδομένα, επιστρέφει 0 εάν θεωρεί πως $f \epsilon F_k$, και 1 εάν θεωρεί πως $f \epsilon H_k$.

Ορισμός 5.5.9 (**Κρυπτογραφικά ασφαλής ψευδοτυχαία γεννήτρια αριθμών**)**.** Ένας πολυωνυμικού χρόνου ντετερμινιστικός αλγόριθμος G, με $G : \{0,1\}^n \to \{0,1\}^m$ είναι κρυπτογραφικά ασφαλής ψευδοτυχαία γεννήτρια αριθμών (PRG) εάν :

1. $m > n$
2. Για κάθε PPT αλγόριθμο D,

$$|Pr[x \leftarrow I_n : D(G(x)) = 1] - Pr[y \leftarrow I_m : D(y) = 1]| = negl(n)$$

Στην ουσία οι ψευδοτυχαίες γεννήτριες είναι αποδοτικοί αλγόριθμοι που επεκτείνουν (stretch) μικρού μήκους string σε ψευδοτυχαίες ακολουθίες. Ουσιαστικά λοιπόν δεν παράγουν τυχαιότητα αλλά χρησιμοποιούν την τυχαιότητα ενός string και την επεκτείνουν. Η ιδιότητα των PRG να είναι κρυπτογραφικά ασφαλείς είναι ισοδύναμη με την ιδιότητα το αποτέλεσμα τους να είναι μη-προβλέψιμο.

Με την βοήθεια των hardcore predicates hp και των μονόδρομων μεταθέσεων f (one-way permutations), μπορεί να φτιαχτεί μια ψευδοτυχαία γεννήτρια $G(s) = f(s)||hp(s)$. Η λογική πίσω από αυτό είναι ότι αφού n f είναι μια μονόδρομη μετάθεση, τότε το αποτέλεσμά της είναι κατανεμημένο ομοιόμορφα, εάν ο σπόρος s είναι επιλεγμένος ομοιόμορφα. Επίσης το hardcore predicate είναι ένα bit που δεν μπορεί να διακριθεί από ένα τυχαίο με μεγαλύτερη πιθανότητα από $\frac{1}{2}$. Συνεπώς το αποτέλεσμα $f(s)||hp(s)$ είναι ψευδοτυχαίο όταν ο s είναι επιλεγμένος ομοιόμορφα. Μπορεί λοιπόν να αποδειχθεί το παρακάτω θεώρημα:

Θεώρημα 5.5.2 ([47])**.** *Εάν υπάρχουν οι μονόδρομες συναρτήσεις , τότε και μόνο τότε υπάρχουν και οι ψευδοτυχαίες γεννήτριες.*

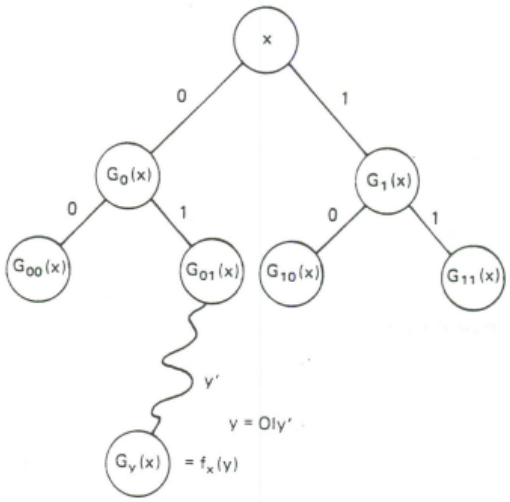
Χρησιμοποιώντας τις ψευδοτυχαίες γεννήτριες, μπορούν να κατασκευαστούν οι ψευδοτυχαίες συναρτήσεις. Ουσιαστικά, πρόκειται για συναρτήσεις ($\in F_n$) που δεν μπορούν να διακριθούν από τυχαίες συναρτήσεις ($\in H_n$) από κανέναν πολυωνυμικού χρόνου αλγόριθμο A.

Ορισμός 5.5.10 (**Ψευδοτυχαία οικογένεια συναρτήσεων**)**.** Μια οικογένεια συναρτήσεων $\mathcal{F} = \{F_n\}_{n\in\mathbb{N}}$, είναι ψευδοτυχαία εάν :

1. Κάθε συνάρτηση στο σύνολο F_n είναι εύκολο να υπολογιστεί.
2. Υπάρχει ένας αποδοτικός αλγόριθμος E ώστε : $E(s,x) = f_s(x)$ για κάθε $x, s \in \{0,1\}^n$.
3. Για κάθε PPT αλγόριθμο A,

$$\left|Pr[f \leftarrow F_n : A(1^n)^f = 1] - Pr[f \leftarrow H_n : A(1^n)^f = 1]\right| = negl(n)$$

Η κατασκευή των ψευδοτυχαίων συναρτήσεων μέσω των ψευδοτυχαίων γεννητριών, εισήχθη από τους Goldreich, Goldwasser, και Micali στην δημοσίευση [11]. Η κατασκευή αυτή χρησιμοποιεί την δομή των δυαδικών δέντρων, μια ψευδοτυχαία γεννήτρια $G : \{0,1\}^k \leftarrow \{0,1\}^{2k}$ και ένα string x μήκους k. Τότε θεωρούμε την ρίζα του δέντρου ως το string x. Στην συνέχεια εφαρμόζουμε την G στο x και παίρνουμε τα k πρώτα bit του $G(x)$ και τα ορίζουμε ως το πρώτο παιδί του x, $G_0(x)$. Στην συνέχεια παίρνουμε τα k επόμενα bit του $G(x)$ και τα ορίζουμε ως το δεύτερο παιδί του x, $G_1(x)$. Εάν αυτό γίνει επαγωγικά για κάθε κόμβο, θα προκύψει ένα δέντρο όπως φαίνεται στην παρακάτω εικόνα :



Σχήμα 5.1: Κατασκευή ψευδοτυχαίων συναρτήσεων, [11]

Εν τέλει για string $x, y \in \{0,1\}^k$ με $y = y_1 y_2 ... y_k$ προκύπτει η ψευδοτυχαία συνάρτηση $f_x(y) = G_{y_k}(...(G_{y_3}(G_{y_2}(G_{y_1}(x)))))$ που αποτελεί το φύλλο του μονοπατιού που ξεκινάει από την ρίζα ώστε x. Συνεπώς μπορεί να αποδειχθεί το παρακάτω θεώρημα:

Θεώρημα 5.5.3. *Εάν υπάρχουν οι ψευδοτυχαίες γεννήτριες , τότε και μόνο τότε υπάρχουν και οι ψευδοτυχαίες συναρτήσεις.*

5.6 Κατασκευή

Για την κατασκευή των VRFs αρκεί να γίνει κατανοητή η σύνδεση τους με τις VUFs. Οι VUFs είναι κάτι σαν σχήματα υπογραφών, που το αποτέλεσμα τους (η υπογραφή) είναι μια τιμή μη-προβλέψιμη αλλά όχι τυχαία. Τα σχήματα υπογραφών γενικά έχουν καλές ιδιότητες που χρειαζόμαστε στις VRFs, όπως η επαλήθευση και το μοναδικό αποτέλεσμα. Όμως το αποτέλεσμα αυτό δεν είναι επαληθεύσιμο από μια μοναδική υπογραφή. Δηλαδή μπορεί να γίνει επαλήθευση υπογραφής για το ίδιο μήνυμα απλά παραγόμενη από άλλα κλειδιά. Τα σχήματα υπογραφών με αποτέλεσμα επαληθεύσιμο από μια μοναδική υπογραφή ονομάζονται VUFs. Η μη προβλεψιμότητα μπορεί να μετατραπεί σε ψευδοτυχαιότητα, με την βοήθεια των hardcore predicates του Θεωρήματος 5.5.1. Το μόνο που μένει λοιπόν είναι να κατασκευαστούν οι VUFs, οι οποίες μετατρέπονται σε VRFs.

Σύμφωνα με το Θεώρημα 5.5.1, οι Goldreich και Levin απέδειξαν πως δωσμένης μιας μονόδρομης συνάρτησης f, με τυχαία είσοδο x, και ενός τυχαίου διανύσματος r, είναι αδύνατο να υπολογιστεί το εσωτερικό γινόμενο $< f(x), r >$ με πιθανότητα μεγαλύτερη του $\frac{1}{2}$ [22]. Ουσιαστικά, μετατρέπεται η δυσκολία υπολογισμού, σε ψευδοτυχαιότητα. Δηλαδή μια συνάρτηση ψευδοτυχαίου αποτελέσματος $f(x)'$ μπορεί να φτιαχτεί από μια συνάρτηση μη-προβλέψιμου αποτελέσματος $f(x)$, ως το εσωτερικό γινόμενο της με αριθμητική $\mathrm{mod}2$, με ένα ομοιόμορφα επιλεγμένο string r, ίδιου μήκους με την f.

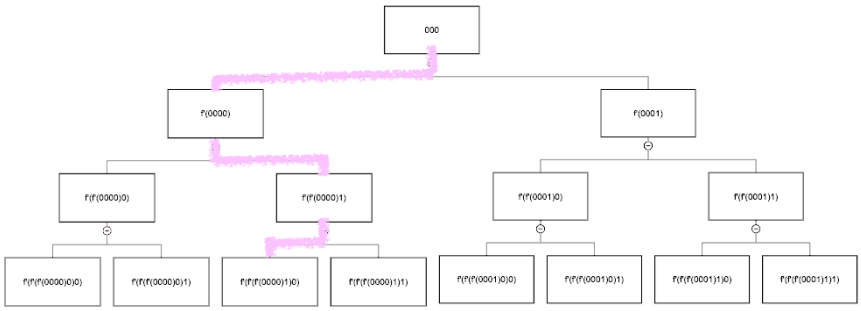
Στο [6] αποδεικνύεται μέσω του θεωρήματος 5.5.1 και των ορισμών των VRFs και VUFs πως :

Πρόταση 5.6.1. *Εάν υπάρχει μια VUF με μήκος εισόδου $a(k)$, μήκος εξόδου $b(k)$ και ασφάλειας $s(k)$, τότε για κάθε $a'(k) \leq a(k)$, υπάρχει μια VRF με μήκος εισόδου $a'(k)$, μήκος εξόδου $b(k) = 1$ και ασφάλειας $s'(k) = \frac{s(k)^{1/3}}{(poly(k) \cdot 2^{a'(k)})}$*

Σύμφωνα με το [48], εάν με την παραπάνω κατασκευή χρησιμοποιηθεί δημόσιο $r \in \{0,1\}^{b(k)}$ τότε μπορεί το αποτέλεσμα των επαγώμενων VRFs της πρότασης 5.6.1 να μην είναι πλέον ψευδοτυχαίο. Πιο συγκεκριμένα, για ένα δημόσιο r, αποδεικνύεται πως μπορεί να επιλεχθεί κατάλληλη είσοδος x εξαρτώμενη από το δημόσιο r για την VUF, και άρα πλέον θα μπορεί να μαντευτεί η τιμή $< f(x), r >$. Αυτό το εμπόδιο μπορεί να αντιμετωπιστεί εάν το μήκος $a'(k)$ της εισόδου της VRFs είναι λογαριθμικά συσχετισμένο με την ασφάλεια $s(k)$. Αποδεικνύεται πως από μια VRF f με είσοδο μήκους a, έξοδο μήκους 1 και ασφάλειας s, μπορεί να φτιαχτεί μια VRF f' με είσοδο μήκους $a' = a - O(loga)$, έξοδο μήκους $b' = a' - 1$ και ασφάλειας $s' = \frac{s}{b'} = \frac{s}{poly(k)}$.

Η συνάρτηση f' μπορεί να μετατραπεί σε VRF f'' που δέχεται σαν είσοδο strings αυθαίρετου μήκους $x \in \{0,1\}^*$. Η μετατροπή αυτή γίνεται με την δημιουργία ενός δυαδικού δέντρου που αποτελείται από κόμβους με τιμές της συνάρτησης f'. Το δέντρο αυτό έχει άπειρο μήκος. Οι τιμές σε κάθε κόμβο, ως τιμές της συνάρτησης f', έχουν μήκος $b' = a' - 1$. Έστω ο αριθμός $x = b_1 b_2 ... b_t$. Τότε θα λέμε ότι η τιμή της $f''(x)$ είναι ο τελευταίος κόμβος του μονοπατιού μήκους t. Η ρίζα του δέντρου είναι ο αριθμός $y_0 = 0^{a'-1}$, ενώ οι υπόλοιποι κόμβοι ορίζονται ως $y_i = f'(y_{i-1} b_i)$.

Για παράδειγμα, το δέντρο μιας f'' που δημιουργείται από την f' με είσοδο μήκους $a' = 4$ και έξοδο μήκους $b' = 3$, για την είσοδο $x = 010$ ακολουθείται το εξής μονοπάτι:



Και άρα η ζητούμενη τιμή είναι $f''(010) = f'(f'(f'(0000)1)0)$. Έχουμε καταλήξει λοιπόν σε VRFs με είσοδο αυθαίρετου μήκους.

Πρόταση 5.6.2. *Εάν υπάρχει μια VRF με μήκος εισόδου $a(k)$, μήκος εξόδου 1 και ασφάλειας $s(k)$, τότε υπάρχει μια VRF με αυθαίρετο μήκος εισόδου, μήκος εξόδου $b(k) = 1$ και ασφάλειας τουλάχιστον $\frac{min\{s(k)^{1/5}, 2^{a(k)/5}\}}{poly(k)}$.*

Το μόνο που μένει για την κατασκευή των VRFs, είναι η κατασκευή των VUFs. Η κατασκευή αυτή γίνεται με την γεννήτρια μη-προβλέψιμων αριθμών του Shamir το 1983 [10]. Ο Adi Shamir βασίζει την κατασκευή της γεννήτριας αυτής στο κρυπτοσύστημα RSA. Η συνάρτηση κρυπτογράφησης RSA αντιστοιχίζει ένα κρυφό μήνυμα M, υπό την γνώση ενός δημοσίου κλειδιού K σε ένα κρυπτογραφημένο μήνυμα M^K με αριθμητική modulo N. Δηλαδή η συνάρτηση κρυπτογράφησης RSA περιγράφεται ως :

$$E_K(M) = M^K(modN)$$

Γνωρίζουμε πως για μεγάλο και σύνθετο N άγνωστης παραγοντοποίησης, το να βρεθεί το μήνυμα M παίρνοντας την K-ιοστή ρίζα του $M^K(modN)$ είναι ένα δύσκολο πρόβλημα. Σε αυτή την υπόθεση δυσκολίας υπολογισμού ριζών $modN$ βασίζεται η ασφάλεια του κρυπτοσυστήματος. Παρ' όλα αυτά εάν χρησιμοποιηθεί ένας N με γνωστή συνάρτηση του Euler $\varphi(N)$, και ο αριθμός K είναι σχετικά πρώτος με την $\varphi(N)$, τότε το πρόβλημα μπορεί να επιλυθεί γρήγορα. Εάν λοιπόν βρεθεί μια ακολουθία κλειδιών $\{K_1, K_2, ...\}$ ώστε $gcd(K_i, \varphi(N)) = 1$ και οι όροι K_i να είναι σχετικά πρώτοι μεταξύ τους για κάθε i, με κατάλληλο N μπορεί να φτιαχτεί ψευδοτυχαία ακολουθία :

$$R_1 = S^{1/K_1}(modN), R_2 = S^{1/K_2}(modN), ...$$

Δηλαδή η γνώση της ακολουθίας των R για j συνεχόμενα K_i, δεν βοηθάει στο να μπορέσει να βρει κάποιος τον επόμενο όρο της ακολουθίας. Με αυτές τις πληροφορίες μπορεί να κατασκευαστεί VUF ως:

Έστω η ομάδα $\mathbb{Z}_m^*$ που είναι το σύνολο των θετικών ακεραίων $< m$ που είναι σχετικά πρώτοι με τον m. Η τάξη της ομάδας, δηλαδή ο αριθμός

των στοιχείων της, είναι ίσος με $\varphi(m)$. Γνωρίζουμε πως για $x \in \mathbb{Z}_m^*$ η αντιστοίχιση $x \to x^e$ με $gcd(e, \varphi(m)) = 1$ είναι μια μετάθεση στην ομάδα $\mathbb{Z}_m^*$. Δηλαδή για κάθε ακέραιο r, υπάρχει το πολύ ένα $x \in \mathbb{Z}_m^*$ ώστε $x^e = r(modm)$. Μπορούμε να θεωρήσουμε λοιπόν $x = r^{1/e}(modm)$ και την συνάρτηση VUF ως την $f(x) = r^{1/p_x}(modm)$ όπου $r \in \mathbb{Z}_m^*$, με m, r με $r \in \mathbb{Z}_m^*$ γνωστά. Αποδεικνύεται στο [10] ότι η επαγώμενη ακολουθία $r^{1/p_{x_1}}(modm), r^{1/p_{x_2}}(modm), ...$ είναι ψευδοτυχαία.

s(k) - Υπόθεση δυσκολίας της RSA : *Έστω A ένας πιθανολογικός αλγόριθμος που εκτελείται σε χρόνο $s(k)$ όταν έχει πρώτη είσοδο το 1^k. Τότε η πιθανότητα ο A να επιτύχει στο παρακάτω πείραμα είναι το πολύ $\frac{1}{s(k)}$:*

1. *Επέλεξε $m \leftarrow RSA_k$; $x \leftarrow \mathbb{Z}_m^*$; $p \leftarrow PRIMES_{k+1}$*
2. *Έστω $y \leftarrow A(1^k, m, x, p)$*
3. *Ο A εάν $y^p = x(modm)$*

Πρόταση 5.6.3. *Έστω $a(k) \leq poly(k)$ και $s(k)$ δύο οποιεσδήποτε συναρτήσεις υπολογισμένες σε πολυωνυμικό χρόνο με το k. Τότε σύμφωνα με την υπόθεση δυσκολίας της RSA, υπάρχει μια VUF με είσοδο μήκους $a(k)$, έξοδο μήκους $b(k) = 1$ και ασφάλειας $s'(k) = \frac{s(k)}{(2^{a(k)} \cdot poly(k))}$.*

Με την χρήση των προτάσεων 5.6.1, 5.6.2 και 5.6.3 με παράμετρο $a(k) = a'(k) = \frac{logs(k)}{7}$ αποδεικνύεται το παρακάτω θεώρημα για την κατασκευή των VRFs:

Θεώρημα 5.6.1. *Υπό την $s(k)$-υπόθεση δυσκολίας της RSA, υπάρχει μια VRF με είσοδο αυθαίρετου μήκους, έξοδο μήκους $b(k) = 1$ και ασφάλειας $\frac{s(k)^{1/35}}{poly(k)}$. Πιο συγκεκριμένα, ισχύει ότι εάν $s(k) = k^{\omega(1)}$ τότε και η VRF έχει επίσης ασφάλεια $k^{\omega(1)}$.*

Το τελευταίο βήμα για την ολοκλήρωση της κατασκευής των VUF είναι να βρεθούν οι κατάλληλοι αριθμοί p_x που θα χρησιμοποιηθούν για την παραγωγή της ακολουθίας μη-προβλέψιμων αριθμών. Στο άρθρο [10] του Adi Shamir αναφέρεται πως αρκεί οι αριθμοί p_x να είναι πρώτοι μεταξύ τους και με το $\varphi(m)$. Χρησιμοποιείται λοιπόν η γεννήτρια ακολουθίας πρώτων αριθμών που έχει προταθεί από τους Cachin, Micali και Stadler το 1999 [23]. Πρόκειται για μια αντιστοίχιση 1-1 ενός string μήκους α bit, σε έναν πρώτο αριθμό μήκους $k+1$ bit. Η συνάρτηση που πραγματοποιεί αυτή την αντιστοίχιση $x \to p_x$ ορίζεται ως $Q : \{0,1\}^\alpha x\{1, ..., 2k^2\} \to$ {i | i είναι ένας k+1 bit ακέραιος}. Τότε δημιουργούνται οι αριθμοί $Q(x,1), Q(x,2), ..., Q(x,2k^2)$ και ο ζητούμενος αριθμός p_x είναι ο πρώτος αριθμός που εμφανίζεται πρώτος ανάμεσά τους. Για την υλοποίηση αυτής της ιδέας χρησιμοποιείται

ένας αλγόριθμος που ονομάζεται *PrimalityTest* και επιστέφει 1 εάν η είσοδος p_x είναι πρώτος αριθμός ή 0 εάν είναι σύνθετος ([25] ή [24]). Η τελική γεννήτρια ακολουθίας πρώτων αριθμών ονομάζεται *PrimeSeq(x,Q,coins)*.

Πρόταση 5.6.4. *Έστω* $a(k) \leq \frac{k}{2}$. *Τότε με πιθανότητα τουλάχιστον* $1 - 2^{-\Omega(k)}$ *για την συνάρτηση* Q, *και για sting coins επιλεγμένα ομοιόμορφα, τότε το* $\{PrimeSeq(x,Q,coins) : x \in \{0,1\}^a\}$ *είναι ένα σύνολο από* 2^a *διακριτούς* $(k+1) - bit$ *πρώτους.*

Περιγραφή της VUF

Γεννήτρια συνάρτηση G

Είσοδος : παράμετρος ασφάλειας 1^k
Έξοδος : δημόσιο κλειδί $pk = (m, r, Q, coins)$, ιδιωτικό κλειδί $sk = (pk, \varphi(m))$ με $m \in RSA_k; r \in \mathbb{Z}_m^*; coins \in \{0,1\}^l$ και Q είναι ένα πολυώνυμο βαθμού το πολύ $2k^2 - 1$.
Περιγραφή :

1. Με το *PrimalityTest* υπολογίζονται δύο πρώτοι q_1, q_2 ώστε $m = q_1 q_2 \in RSA_k$ και $\varphi(m) = (q_1 - 1)(q_2 - 1)$.
2. $r \leftarrow \mathbb{Z}_m^*,\ coins \leftarrow \{0,1\}^l$.
3. Επιλογή του πολυωνύμου Q βαθμού το πολύ $2k^2 - 1$.
4. Επιστροφή $(m, r, Q, coins)$ και $\varphi(m)$.

Εκτιμήτρια συνάρτηση F

Είσοδος : ιδιωτικό κλειδί $sk = (pk, \varphi(m))$ με $pk = (m, r, Q, coins)$, και $x \in \{0,1\}^a$
Έξοδος : $u \in \mathbb{Z}_m^*$
Περιγραφή :

1. Με το *PrimeSeq* υπολογίζεται ο εκθέτης $p_x = PrimeSeq(x, Q, coins)$, όπου p_x ένας $(k+1)$-bit πρώτος.
2. Υπολογισμός $u = r^{1/p_x}(mod m)$

Συνάρτηση επαλήθευσης V

Είσοδος : δημόσιο κλειδί $pk = (m, r, Q, coins)$, x και u
Έξοδος : YES / NO

Περιγραφή :

1. Υπολίγισε $p_x = PrimeSeq(x, Q, coins)$
2. Έλεγξε ότι ο αριθμός p_x είναι μεγαλύτερος από τον m και ότι είναι πρώτος με το $PrimalityTest$.
3. Έλεγξε ότι $u \in \mathbb{Z}_m^*$ και $u^{p_x} = r(mod m)$.
4. Εάν οι 3 παραπάνω συνθήκες ισχύουν επέστρεψε YES, αλλιώς επέστρεψε NO.

Κεφάλαιο 6

Algorand

6.1 Εισαγωγή

Το Algorand ([3],[4]) είναι ένα πρωτόκολλο κρυπτονομίσματος που αξιοποιεί την τεχνολογία blockchain για επιτυχή επικύρωση συναλλαγών. Για την ομοφωνία στο blockchain του Algorand αξιοποιείται ο αλγόριθμος Proof of Stake (PoS) ώστε χρήστες να επιλέγονται τυχαία ανάλογα με το stake που διαθέτουν ώστε να προτείνουν και να συμφωνήσουν σε ένα καινούριο block συναλλαγών. Η τυχαία επιλογή χρηστών γίνεται με την τεχνολογία του κρυπτογραφικού εργαλείου Verifiable Random Functions (VRFs), ενώ η ομοφωνία επιτυγχάνεται με την χρήση ενός πρωτοκόλλου Βυζαντινής Συμφωνίας (BA★).

6.2 Η λύση για το Blockchain Trilemma

Τα συστήματα που χρησιμοποιούν την τεχνολογία blockchain περιστρέφονται γύρω από τρία σημαντικά εργαλεία: την αποκέντρωση, την ασφάλεια και την απόδοση. Είναι σημαντικό να διευκρινιστεί ο ρόλος και η σημασία της κάθε έννοιας ξεχωριστά.

Η αποκέντρωση (decentralization) αποτελεί την θεμελιώδη λίθο δημιουργίας των τεχνολογιών blockchain. Ως αποκέντωση ορίζουμε την μεταφορά της λειτουργίας και της οργάνωσης ενός συστήματος από μια κεντρική εξουσία, σε ένα διαμοιρασένο σύστημα χρηστών. Οι αποφάσεις λαμβάνονται συλλογικά από μια ομάδα χρηστών με ομοφωνία (consensus). Με άλλα λόγια, οι ίδιοι χρήστες πραγματοποιούν την εισαγωγή των δεδομένων στο αποκεντρωμένο σύστημα, και άρα αυτά δεν ελέγχονται από καμία συγκεκριμένη αρχή.

Η ασφάλεια (security) είναι επίσης ένας βασικός πυλώνας της τεχνολογίας αυτής. Δεδομένου ότι το blockchain αποτελεί μια βάση δεδομένων για αποθήκευση πληροφοριών, είναι σημαντικό να εξασφαλίζεται η ακεραιότητα τους. Επιπλέον, οι περισσότερες εφαρμογές της τεχνολογίας αυτής εί-

ναι οικονομικής φύσης (κρυπτονομίσματα), όπου τα δεδομένα είναι επιρρεπή σε προσπάθειες υποκλοπής. Οι κατάλληλοι αλγόριθμοι ομοφωνίας σε συνδυασμό με την κρυπτογραφία μπορούν να υποσχεθούν ασφαλείς πληροφορίες.

Μια τελευταία αλλά εξαιρετικά σημαντική έννοια είναι η απόδοση (scalability) του συστήματος. Διανύοντας μια εποχή που τα δεδομένα υπάρχουν παντού γύρω μας, η τεχνολογία blockchain θα πρέπει να μπορεί να υποστηρίξει μεγάλους όγκους πληροφοριών αλλά και να παρέχει γρήγορη αφομοίωσή τους στην αλυσίδα. Επίσης οι χρήστες γίνονται όλο και περισσότεροι με αποτέλεσμα να υπάρχει ανάγκη για blockchain αποδοτικά σε μεγάλο αριθμο χρηστών (γρήγορη ομοφωνία ανεξάρτητη από τον αριθμό αυτό).

Ορισμός 6.2.1. Το blockchain trilemma δηλώνει πως οι τρέχουσες τεχνολογίες blockchain μπορούν να διατηρούν μόνο δύο από τις παρακάτω ιδιότητες :

- Αποκέντρωση
- Ασφάλεια
- Απόδοση

Τα περισσότερα blockchain δυσκολεύονται να λύσουν το παραπάνω τρίλημμα, θυσιάζοντας κάποια από τις τρεις ιδιότητες για την λειτουργία τους. Το Algorand φαίνεται να είναι μια καλή λύση αποκεντρωμένου, ασφαλούς και αποδοτικού blockchain. Αποτελεί ένα proof-of-stake blockchain που αξιοποιεί τα πρωτόκολλα ανοχής σε Βυζαντινά Σφάλματα (PBFT) για την δημιουργία μιας αποκεντρωμένης αλυσίδας συναλλαγών. Ο αλγόριθμος ομοφωνίας αυξάνει την αλυσίδα του Algorand επιλέγοντας τυχαία κάποιους χρήστες του δικτύου (αντιπρόσωποι) , οι οποίοι προτείνουν καινούρια block και συμφωνούν για το ποιό θα επισυναπτεί στην αλυσίδα. Η ψηφοφορία γίνεται σε γύρους (rounds) και οι αντιπρόσωποι ψηφίζουν μέχρι να έρθουν σε ομορφωνία. Για να συμβεί αυτό, υπάρχει κάτω όριο στους ψήφους για να καταλήξουν σε συμφωνία, το οποίο είναι κατάλληλο ώστε η απόφαση να γίνεται με πλειοψηφία ειλικρινών χρηστών.

Πιο αναλυτικά, σε κάθε γύρο οι χρήστες τρέχουν έναν αλγόριθμο που ονομάζεται cryptographic sortition με κατάλληλες εισόδους για να δουν εαν έχουν επιλεχθεί για να προτείνουν καινούριο block. Στην περίπτωση που έχουν επιλεχθεί, δημιουργούν blocks με συναλλαγές που δεν έχουν προστεθεί ακόμα στην αλυσίδα, και τα διαμοιράζονται στο δίκτυο του Algorand. Στην συνέχεια, μέσω ενός πρωτοκόλλου Βυζαντινής Συμφωνίας (BA★), πραγματοποιούνται γύροι ψηφοφορίας από ομάδες αντιπροσώπων μέχρι να καταλήξουν στο block με τον κατάλληλο αριθμό ψήφων.

Ο λόγος που θεωρείται το Algorand λύση του blockchain trilemma, είναι η γρήγορη επικύρωση και προσθήκη συναλλαγών στην αλυσίδα (1 λεπτό ανα block), αλλά και η αποδοτικότητα που έχει ενώ συμμετέχουν χιλιάδες χρήστες. Επιπλέον είναι ιδιαίτερα ασφαλές, μιας και ο αλγόριθμος ομοφωνίας που έχει δίνει αμελητέα πιθανότητα για fork. Είναι σημαντικό λοιπόν να εξετάσουμε εις βάθος την λειτουργία του πρωτοκόλλου,

ως μια καινοτόμα προσέγγιση με σημαντικά αποτελέσματα.

6.3 Κρυπτογραφικά Εργαλεία

Συναρτήσεις Σύνοψης - Hash functions

Η ασφάλεια του πρωτοκόλλου βασίζεται στην χρήση επίσης ασφαλών συναρτήσεων σύνοψης, οι οποίες θα αντιστοιχίζουν αλφαριθμητικά σε δυαδικό διάνυσμα συγκεκριμένου μήκους. Ικανοποιούν όλες τις ιδιότητες (preimage resistance, second preimage resistance, collision resistance). Το μήκος εξόδου που χρησιμοποιείται είναι τα 256bits ώστε το πρωτόκολλο να είναι λειτουργικό αλλά και ασφαλές. Οι συναρτήσεις σύνοψης μοντελοποιούνται ως random oracle, δηλαδή το αποτέλεσμα θα πρέπει να φαίνεται τυχαία και ανεξάρτητα επιλεγμένο.

Ψηφιακές Υπογραφές

Οι ψηφιακές υπογραφές είναι ένα πολύ βασικό εργαλείο για πρωτόκολλα κρυπτονομισμάτων και αλγόριθμους ομοφωνίας, καθώς έτσι επιβεβαιώνονται συναλλαγές και γίνονται ψηφοφορίες. Αποτελούνται από τρεις αλγόριθμους, τον αλγόριθμο παραγωγής κλειδιών G, τον αλγόριθμο υπογραφής S και τον αλγόριθμο επαλήθευσης V. Κάθε χρήστης διαθέτει ένα δημόσιο και ένα αντίστοιχο ιδιωτικό κλειδί (pk,sk) με τα οποία μπορούν να υπογράψουν ηλεκτρονικά. Επίσης με την γνώση του δημοσίου κλειδιού ενός άλλου χρήστη μπορούν να επαληθεύσουν το γνήσιο μιας υπογραφής.

Η ψηφιακή υπογραφή sig_{pk_i} ενός χρήστη με δημόσιο κλειδί pk_i δημιουργείται από τον αλγόριθμο S με το ιδιωτικό κλειδί του χρήστη sk_i και με την χρήση μιας συνάρτησης σύνοψης $H(.)$ στο μήνυμα m που προσυπογράφεται. Πιο συγκεκριμένα : $sig_{pk_i}(m) = S(H(m), sk_i)$.

Η ψηφιακή υπογραφή sig_{pk_i} επαληθεύεται μέσω του αλγορίθμου V με είσοδο το δημόσιο κλειδί pk_i, την υπογραφή sig_{pk_i} και το μήνυμα m. Στην περίπτωση επαλήθευσης, ο αλγόριθμος V επιστρέφει YES. Επίσης, είναι αδύνατη η πλαστογράφηση κάποιας ηλεκτρονικής υπογραφής, καθώς δεν γίνεται να βρεθεί υπογραφή $sig(m)$ ώστε ο V να επιστρέψει YES χωρίς την γνώση του ιδιωτικού κλειδιού.

Οι ψηφιακές υπογραφές που χρησιμοποιούνται στο Algorand έχουν την ιδιότητα της ανάκτησης μηνύματος, ενώ τυπικά οι ψηφιακές υπογραφές δεν δίνουν αυτή την δυνατότητα. Για τον λόγο αυτό ορίζεται : $SIG_i(m) = (i, m, sig_{pk_i}(m))$.

Verifiable Random Functions

Η ιδιότητα της φαινομενικής τυχαιότητας του αποτελέσματος των συναρτήσεων

σύνοψης σε συνδυασμό με την ιδιότητα της επαλήθευσης των ηλεκτρονικών υπογραφών, έδωσαν την ιδέα των ψευδοτυχαίων συναρτήσεων επαληθευόμενου αποτελέσματος (VRFs). Έχουν πολύ σημαντική χρησιμότητα σε πρωτόκολλα κρυπτονομισμάτων όπου θέλουμε τυχαία επιλογή χρηστών αλλά υπάρχει η ανάγκη επαλήθευσης της τυχαίας επιλογής. Ο ρόλος των συναρτήσεων αυτών στο Algorand είναι να βοηθούν στην αποτελεσματική δημιουργία μιας ομάδας χρηστών που θα ψηφίζουν για την προσθήκη συναλλαγών στην αλυσίδα.

6.4 Οργάνωση του πρωτοκόλλου

Το Algorand χωρίζεται σε γύρους (rounds, r). Σε κάθε γύρο διατηρείται μια λίστα με τους λογαριασμούς που υπάρχουν στο σύστημα και με τα αντίστοιχα κρυπτονομίσματα που διαθέτει ο κάθε λογαριασμός. Κάθε χρήστης-λογαριασμός προσδιορίζεται από ένα δημόσιο κλειδί (public key, pk) με το οποίο μπορεί να πραγματοποιεί πληρωμές προς άλλους χρήστες. Οι πληρωμές αυτές διατηρούνται στο σύστημα, με στόχο να οργανωθούν σε διαδοχικά blocks (B^r). Θα πρέπει πάντα να ικανοποιούνται οι παρακάτω τρεις ιδιότητες:

- Τα blocks πρέπει να είναι δημόσια στο σύστημα.
- Οι πληρωμές που μπαίνουν σε κάποιο block πρέπει να είναι έγκυρες (σύμφωνα με την λίστα λογαριασμών).
- Οι έγκυρες πληρωμές να εμφανίζονται γρήγορα σε blocks.

Οι πληρωμές πραγματοποιούνται με την βοήθεια των ψηφιακών υπογραφών (digital signatures) και περιγράφονται ως εξής :

$$\mathcal{T} = SIG_{pk}(pk, pk', a', I, H(\mathcal{I}))$$

Όπου pk και pk' είναι τα δημόσια κλειδιά του αποστολέα και του παραλήπτη αντίστοιχα, a' είναι τα απεσταλμένα κρυπτονομίσματα, I είναι οι δημόσιες πληροφορίες για την πληρωμή ενώ $\mathcal{I}$ οι ευαίσθητες πληροφορίες.
Οι πληρωμές και οι ευαίσθητες πληροφορίες είναι ασφαλείς, όσο είναι ασφαλείς οι ψηφιακές υπογραφές και οι συναρτήσεις σύνοψης.

Περιγραφή γύρου

Ο στόχος σε κάθε γύρο είναι να προστεθεί ένα καινούριο block με νέες συναλλαγές στην αλυσίδα του Algorand. Για τον λόγο αυτό επιλέγονται τυχαία χρήστες του Algorand οι οποίοι θα προτείνουν τις συναλλαγές που θα προστεθούν στο καινούριο block. Από τα προτεινόμενα block επιλέγεται ένα και οργανώνεται μια καινούρια ομάδα χρηστών που θα ψηφίσει για αυτό. Μετά από κάποια βήματα ψηφοφορίας, αφού επιτευχθεί βυζαντινή συμφωνία στο block, αυτό μοιράζεται στο δίκτυο του Algorand και προστίθεται στην αλυσίδα blockchain.

Σε κάθε γύρο r επαναλαμβάνεται η εξής ακολουθία :

1. Τυχαία επιλογή ενός χρήστη ως block proposer
2. Δημιουργία, υπογραφή και διαμοιρασμός block B_r στο δίκτυο
3. Τυχαία επιλογή επιτροπής ψηφοφορίας $SV^{r,s}$ στο προτεινόμενο block B_r
4. Βυζαντινή συμφωνία στο block B_r
5. Διαμοιρασμός του block B_r και άρα προσθήκη στην αλυσίδα

6.5 Υποθέσεις

Το Algorand λειτουργεί υπό δύο στόχους, ασφάλειας και αποδοτικότητας. Όσον αφορά την ασφάλεια, θέλουμε όλοι οι χρήστες να συμφωνούν στις ίδιες συναλλαγές. Εάν μια συναλλαγή προταθεί από ειλικρινή χρήστη και μπεί στην αλυσίδα, τότε και οι υπόλοιπες συναλλαγές που προτείνουν ειλικρινείς χρήστες θα μπούν στην ίδια αλυσίδα. Όσον αφορά την αποδοτικότητα, θέλουμε να επιτυγχάνεται ομοφωνία σε περίπου ένα λεπτό, δηλαδή κάθε λεπτό να προστίθεται και ένα καινούριο block σην αλυσίδα. Προφανώς όλα αυτά θέλουμε να τηρούνται με βάση κάποιες υποθέσεις .

Η βασικότερη υπόθεση για το πρωτόκολλο είναι πως το μεγαλύτερο ποσοστό των χρημάτων ανήκει σε ειλικρινείς χρήστες. Πιο συγκεκριμένα, επειδή το Algorand υλοποιεί έναν μηχανισμό ομοφωνίας που βασίζεται στην Βυζαντινή Συμφωνία, θέλουμε πάνω από τα $\frac{2}{3}$ των χρημάτων να αντιστοιχούν σε λογαριασμούς έμπιστων χρηστών. Αυτό είναι μια λογική υπόθεση δεδομένου ότι εάν κάποιος θέλει να βλάψει ένα proof-of-stake σύστημα με μια επίθεση του τύπου "majority attack", θα πρέπει να ελέγχει πάνω από το $\frac{1}{3}$ των κρυπτνομισμάτων του συστήματος. Δηλαδή να ρισκάρει να χάσει μεγάλο ποσοστό των χρημάτων, καθώς με μια τέτοια επίθεση θα προκαλέσει πρόβλημα στην επιβεβαίωση νέων συναλλαγών.

Επιπροσθέτως, χρειάζεται να οριστεί ένα μοντέλο αντιπάλου, δεδομένης της υπόθεσης ότι δεν γίνεται αποκλειστικά ειλικρινής χρήση του πρωτοκόλλου. Στο Algorand, ο αντίπαλος (Adversary) θεωρείται ότι έχει την απόλυτη ελευθερία να πράξει όπως θέλει. Μπορεί να διαφθείρει άμεσα όποιον θέλει, να ελέγχξει και να συντονίσει όλους τους κακόβουλους χρήστες, να επιτεθεί στο δίκτυο και στο πρωτόκολλο. Υπάρχουν ωστόσο ένα βασικό εμπόδιο, οι κρυπτογραφικές υποθέσεις. Υποθέτουμε πως ο Adversary δεν έχει απεριόριστη υπολογιστική δύναμη και άρα, εφόσον οι συναρτήσεις κατακερματισμού και οι ψηφιακές υπογραφές λειτουργούν ορθά, το Algorand είναι ασφαλές. Επιπρόσθετα, δεν μπορεί να επέμβει στην επικοινωνία των ειλικρινών χρηστών, και άρα είναι αδύνατο να διαφθείρει μεγάλο ποσοστό του δικτύου. Μπορεί όμως να αλλάξει την σειρά

των ειλικρινών μηνυμάτων ή να στείλει πιο γρήγορα κάποια μηνύματα.

Πιο συγκεκριμένα για την επικοινωνία των ειλικρινών χρηστών, υποθέτουμε ότι έχουν αυστηρό συγχρονισμό. Δηλαδή ότι μπορούν να επικοινωνούν μεταξύ τους με μεγάλο ποσοστό επιτυχίας. Πιο αναλυτικά, το 95% των έμπιστων χρηστών μπορούν να στείλουν μήνυμα το οποίο θα παραληφθεί πάλι κατά μεγάλο ποσοστό από τους υπόλοιπους ειλικρινείς χρήστες (95%) σε συγκεκριμένο χρονικό περιθώριο. Άρα είναι πολύ δύσκολο για τον αντίπαλο να έχει τον έλεγχο του μεγαλύτερου ποσοστού του δικτύου, αφού δεν θα μπορεί να εμποδίσει την ειλικρινή επικοινωνία, και διασφαλίζεται ότι n αλυσίδα θα συνεχίσει να αυξάνεται.

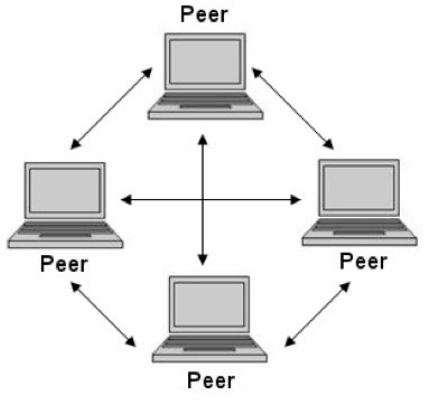
Ενώ οι ειλικρινείς χρήστες έχουν αυστηρό συγχρονισμό, το δίκτυο συνολικά είναι μερικώς συγχρονισμένο. Διακρίνεται δηλαδή από δύο φάσεις συγχρονισμού, μπορεί να είναι τελείως ασύγχρονο για μια μεγάλη αλλά περιορισμένη χρονική περίοδο b (από μια μέρα έως και μια βδομάδα), θα πρέπει όμως μετά από την b περίοδο να έχουμε μια περίοδο s αυστηρού συγχρονισμού, ώστε να επανέλθει το δίκτυο. Το διάστημα s είναι μικρότερο το b (μερικές ώρες ή και μια μέρα).

Λόγω της υπόθεσης μερικού συγχρονισμού, προκειμένου το πρωτόκολλο να μπορεί να επανέλθει σε αυστηρό συγχρονισμό ύστερα από την περίοδο b, οι χρήστες υποθέτουμε ότι διαθέτουν loosely synchronized clocks. Αυτό μπορεί να πραγματοποιηθεί με την χρήστη του πρωτοκόλλου Network Time Protocol (NTP). Ουσιαστικά πρόκειται για ένα πρωτόκολλο που χρησιμοποιείται σε διαμοιρασμένα ασύγχρονα δίκτυα ώστε να συγχρονιστούν όλοι οι υπολογιστές του συστήματος με μικρή χρονική διαφορά (ίσως κάποια χιλιοστά του δευτερολέπτου).

6.6 Μοντέλο επικοινωνίας χρηστών

Οι χρήστες του Algorand διαθέτουν από έναν "κόμβο" (node) σε ένα αποκεντρωμένο ομότιμο δίκτυο (peer-to-peer), όπου συνδέονται τυχαία με άλλους ενεργούς κόμβους χρηστών. Τα μηνύματα μεταξύ δύο κόμβων αποστέλνονται μέσω ενός διαύλου επικοινωνίας, χωρίς να απαιτείται διαμεσολαβητής. Είτε κάποιος χρήστης θα αποστέλνει μηνύματα σε άλλους κόμβους, είτε θα λαμβάνει. Το νόημα είναι ότι τα μηνύματα θα διαμοιάζονται γρήγορα στο δίκτυο μεταξύ των κόμβων, και εν τέλει όλοι οι κόμβοι θα έχουν λάβει μετά από λίγη ώρα το ίδιο μήνυμα. Ο τρόπος με τον οποίο διαμοιράζονται τα μηνύματα βασίζεται σε ένα πρωτόκολλο επικοινωνίας που ονομάζεται gossip protocol. Σύμφωνα με αυτό, κάθε κόμβος συνδέεται με έναν αριθμό χρηστών που ονομάζονται γειτονικοί κόμβοι (neighbors), και κάθε φορά που λαμβάνει ένα μήνυμα το στέλνει στους γείτονες του. Αυτό το μοτίβο το ακολουθούν όλοι οι κόμβοι για τους γείτονές

τους και εν τέλει τα μηνύματα διαμοιάζονται σε όλο το δίκτυο. Στο Algorand, προκειμένου να μην υπερφορτωθεί κάποιος κόμβος με μηνύματα, κάθε μήνυμα είναι υπογεγραμμένο και ελέγχεται από κάθε κόμβο πριν διαμοιραστεί. Πιο συγκεκριμένα, επιτρέπεται ένα μήνυμα από κάθε χρήστη την φορά, οπότε εάν κάποιος κόμβος λάβει το ίδιο μήνυμα απλά το αγνοεί και δεν το διαδίδει στους γειτόνους του. Επίσης, σε κάθε γύρο r οι κόμβοι αλλάζουν γείτονες, προκειμένου να μην υπάρχουν απομονωμένοι κόμβοι για πολλή ώρα στο δίκτυο.



Σχήμα 6.1: Ομότιμο δίκτυο, peer-to-peer

Στα πρωτόκολλα κρυπτονομισμάτων, τα peer-to-peer δίκτυα θα πρέπει να δομούνται ανάλογα με τον μηχανισμό ομοφωνίας που διαθέτουν. Για παράδειγμα, στο δίκτυο που Bitcoin όλοι οι κόμβοι είναι ισότιμοι και δεν υπάρχει κάποιο κεντρικό σύστημα που ελέγχει την επικοινωνία. Και αυτό είναι άμεσα εξαγόμενο από την φιλοσοφία των κρυπτονομισμάτων ως αποκεντρωμένα νομισματικά συστήματα. Παρ' όλα αυτά, το Algorand σε σχέση με το Bitcoin, λειτουργεί με διαφορετικό μηχανισμό ομοφωνίας και η υιοθέτηση παρόμοιου μοντέλου επικοινωνίας με το Bitcoin, το θέτει σε κίνδυνο επιθέσεων. Στο άρθρο αναφέρεται πως η σύνδεση μεταξύ των χρηστών στο δίκτυο γίνεται με τυχαίο τρόπο.

Τα μηνύματα που αποστέλλονται στο δίκτυο του Algorand είναι τεσσάρων ειδών :

- Μηνύματα συναλλαγών $\mathcal{T}$
- Μηνύματα ψήφου $\mathcal{V}_i^{r,s}$
- Μηνύματα πρότασης block $\mathcal{P}_i^r$
- Μηνύματα προτεραιότητας $\mathcal{C}_i^r$

Τα μηνύματα συναλλαγών έχουν την μορφή :

$$\mathcal{T} = SIG_{pk}(pk, pk', a', I, H(\mathcal{I}))$$

Όπου $\mathcal{T}$ είναι η συναλλαγή από τον χρήστη με δημόσιο κλειδί pk στον χρήστη pk', ποσού a' με περιγραφή I και ευαίσθητες πληροφορίες $\mathcal{I}$ προστατευόμενες από μια συνάρτηση σύνοψης $H(.)$ με 256-bit.

Τα μηνύματα ψήφου $\mathcal{V}_i^{r,s}$ προσδιορίζονται από την ψήφο του χρήστη i $v_i^{r,s}$ στον γύρο r με βήμα s, υπογεγραμμένη, και από την απόδειξη ότι ο χρήστης i είναι εκλεγμένος ως ψηφοφόρος για τον γύρο r και το βήμα s, $(\sigma_i^{r,s})$. Πιο αναλυτικά :

$$\mathcal{V}_i^{r,s} = (v_i^{r,s}, \sigma_i^{r,s})$$

Τα μηνύματα πρότασης block $\mathcal{P}_i^r$ περιλαμβάνουν το block που προτείνεται από τον χρήστη i για τον γύρο r, (B_i^r), το hash του B_i^r υπογεγραμμένο από τον χρήστη i, $sig_i(H(B_i^r))$, και την απόδειξη ότι ο χρήστης i είναι εκλεγμένος για να προτείνει block για τον γύρο r, $(\sigma_i^{r,1})$. Το βήμα s που συμβαίνει το block poposal είναι το πρώτο βήμα του κάθε γύρου και γι'αυτό $s = 1$. Άρα :

$$\mathcal{P}_i^r = (B_i^r, sig_i(H(B_i^r)), \sigma_i^{r,1})$$

Τέλος, το τέταρτο είδος μηνυμάτων πραγματοποιείται στο ίδιο βήμα $s = 1$ με το μήνυμα πρότασης block $\mathcal{P}_i^r$, και περιλαμβάνεται σε αυτό. Υπάρχει, όμως, σημαντικός λόγος που διαμοιράζεται ξεχωριστά. Τα μηνύματα πρότασης block περιλαμβάνουν τα block με τις προτεινόμενες συναλλαγές, τα οποία είναι αρκετά μεγάλου μεγέθους. Προκειμένου να μην υπάρχουν καθυστερήσεις, στέλνονται πρώτα τα μηνύματα προτεραιότητας C_i^r που είναι αρκετά μικρά (περίπου 200bytes) και διαδίδονται γρήγορα στο δίκτυο. Έτσι οι χρήστες γνωρίζουν ποιά είναι η πρόταση με την μεγαλύτερη προτεραιότητα και στην συνέχεια διαδίδουν μόνο τα μηνύματα πρότασης block που σχετίζονται με αυτή.

Τα μηνύματα προτεραιότητας λοιπόν περιγράφονται ως :

$$C_i^r = (\sigma_i^{r,1})$$

Όπου $\sigma_i^{r,1}$ είναι η απόδειξη ότι ο χρήστης i έχει επιλεχθεί ως block proposer για τον γύρο r.

6.7 Cryptographic Sortition

Χρήση στο πρωτόκολλο

Το Algorand είναι ένα δημόσιο permissionless proof-of-stake blockchain. Αυτό σημαίνει πως οι χρήστες που θα προσθέτουν τις νέες συναλλαγές στην αλυσίδα και θα ψηφίζουν, θα πρέπει να επιλέγονται κατάλληλα βάσει των κρυπτονομισμάτων που διαθέτουν στο σύστημα (δηλαδή βάσει του stake τους). Παράλληλα, επειδή το σύστημα είναι αποκεντρωμένο, θέλουμε οι χρήστες να επιλέγονται με τυχαίο και ομοιόμορφο τρόπο. Όλα

αυτά τα προβλήματα επιλογής χρηστών λύνονται από τον κρυπτογραφικό αλγόριθμο που ονομάζεται *Cryptographic Sortition*.

Λόγω του ότι οι χρήστες που υπάρχουν στο σύστημα είναι πολλοί, για να παρθούν αποφάσεις δημιουργούνται μικρές επιτροπές. Οι χρήστες τρέχουν ιδιωτικά τον αλγόριθμο σε κάθε βήμα για να δουν εάν έχουν επιλεχθεί να ψηφίσουν. Υπάρχουν δύο είδη επιτροπών που δημιουργούνται μέσω του Cryptographic Sortition: οι χρήστες που θα προτείνουν το block που θα μπει στη αλυσίδα στο τέλος του γύρου, και οι χρήστες που θα έρχονται σε ομοφωνία για ένα block συναλλαγών. Οι πρώτοι αναφέρονται ως block proposers ενώ οι δεύτεροι ως committee members.

Οι block proposers προέρχονται από την εκτέλεση του αλγορίθμου στο πρώτο βήμα του πρωτοκόλλου. Αφού διαμοιραστούν οι προτάσεις των block, έχουμε αλλεπάλληλες ψηφοφορίες μέχρι την ομοφωνία, με διαφορετική επιτροπή σε κάθε ψηφοφορία. Συνεπώς ο αλγόριθμος Cryptographic Sortition χρησιμοποιείται σχεδόν σε κάθε βήμα του πρωτοκόλλου. Το μέγεθος του συνόλου των block proposers είναι διαφορετικό από αυτό των committee members. Οι block proposers, επειδή πρόκειται να δημιουργήσουν και να διαδώσουν από ένα block στο δίκτυο, που έχει μεγάλο μέγεθος σε σχέση με ένα μήνυμα ψήφου, είναι πολύ λιγότεροι στο πλήθος.

Έμπνευση Αλγορίθμου

Η ιδέα για τον αλγόριθμο Cryptographic Sortition προήλθε από προτάσεις για την βελτίωση σχημάτων μικροπληρωμών (Micropayment Schemes). Οι μικροπληρωμές αποτελούν πληρωμές μικρού ποσού, συνήθως στο ηλεκτρονικό εμπόριο, και πραγματοποιούνται μέσω συστημάτων πληρωμών (ένας σύνολο πρωτοκόλλων που αλληλεπιδρά μεταξύ τριών τουλάχιστον πλευρών, του αγοραστή/χρήστη, του εμπόρου και της τράπεζας). Παραδείγματα μικροπληρωμών είναι οι πληρωμές "per click" σε παρόχους υπηρεσιών διαδικτύου (Skroutz, YouTube κ.λ.π.). Οι μικροπληρωμές θα μπορούσαν να πληρώνονται όπως οι μακροπληρωμές αλλά το πρόβλημα είναι στο μικρό αντίτιμο για τις υπηρεσίες και στο μεγάλο κόστος επεξεργασίας της πληρωμής λόγω της τράπεζας.

Μια λύση για το πρόβλημα των σχημάτων μικροπληρωμών προτάθηκε από τον RL.Rivest το 1997 στο άρθρο "Electronic Lottery Tickets as Micropayments" [12]. Η ιδέα ήταν αντί να πληρώνονται οι μικροπληρωμές κανονικά με την αντίστοιχη προμήθεια, να επιλέγονται κάποιες από αυτές και να πληρώνονται ως μακροπληρωμές. Ουσιαστικά, θα γίνεται μια τυχαία επιλογή μεταξύ των μικροπληρωμών με πιθανότητα επιλογής $s \epsilon (0,1)$, και οι επιλεγμένες μικροπληρωμές θα υπάρχει χρέωση $\frac{1}{s} * \textit{τιμή μικροπληρωμής}$. Για παράδειγμα, για 1000 μικροπληρωμές κόστους 0,01 με πιθανότητα επιλογής $s = \frac{1}{1000}$, θα έχουμε 999 μικροπληρωμές χωρίς χρέωση αλλά 1 μακροπληρωμή με χρέωση $\frac{1}{s} * 0,01 = 1000 * 0,01 = 10$. Το σημαντικό σε

αυτό το κομμάτι είναι ότι θα έχουμε μια μόνο χρέωση προμήθειας διότι έχουμε μια μόνο πληρωμή. Συνεπώς με αυτό το σχήμα μικροπληρωμής ο χρήστης χρεώνεται με την λογική προμήθεια, και η τράπεζα επεξεργάζεται μια μόνο συναλλαγή.

Υλοποίηση με H-chain : Θεωρούμε μια μονόδρομη συνάρτηση H, με εύκολο υπολογισμό αλλά δύσκολη αντιστροφή.

1. Ο αγοραστής/χρήστης παράγει ένα τυχαίο σπόρο x_n και υπολογίζει μια αλυσίδα σύνοψης $x_n, x_{n-1}, ..., x_0$ όπου $x_i = H(x_{i+1})$ για $i = 0, 1, 2, ..n-1$.
2. Ο πάροχος παράγει ένα τυχαίο σπόρο ω_n και υπολογίζει μια αλυσίδα σύνοψης $\omega_n, \omega_{n-1}, ..., \omega_0$ όπου $\omega_i = H(\omega_{i+1})$ για $i = 0, 1, 2, ..n-1$.
3. Ο χρήστης στέλνει στον πάροχο μόνο την υπογραφή της ρίζας της αλυσίδας x_0
4. Ο πάροχος στέλνει στον χρήστη την υπογραφή της ρίζας της αλυσίδας ω_0
5. Ο χρήστης περιλαμβάνει την ρίζα ω_0 μαζί με την x_0 στην υπογραφή του
6. Κάθε φορά που ο χρήστης πραγματοποιεί κάποια πληρωμή στέλνει και από ένα x_i ξεκινώντας από το x_1 προς το x_n.
7. Για $s = \frac{1}{1000}$, εάν $x_i mod 1000 = \omega_i mod 1000$, τότε η μικροπληρωμή έχει επιλεχθεί.
8. Ο πάροχος στέλνει στην τράπεζα το x_i, το ω_i και την υπογραφή του με τα ω_0 και x_0
9. Η τράπεζα μπορεί να κάνει επιβεβαίωση με το να χρησιμοποιήσει την συνάρτηση σύνοψης H σε όλες τις τιμές διαδοχικά ξεκινώντας από την x_i καταλήγοντας στην x_0, και από την ω_i στην ω_0.
10. Η τράπεζα μετά την επιβεβαίωση χρεώνει στον χρήστη $1000 * \textit{τιμή μικροπληρωμής}$ και την αντίστοιχη προμήθεια και πιστώνει το αντίστοιχο ποσό στον πάροχο.

Η παραπάνω πρόταση αποτελεί μια καλή λύση για το πρόβλημα των μικροπληρωμών, με κάποια αρνητικά όμως. Απαιτείται αλληλεπίδραση μεταξύ του παρόχου και του χρήστη ώστε να επιλεχθούν κάποιες από τις μικροπληρωμές και εκτός αυτού, υπάρχει πιθανότητα ο χρήστης να κληθεί να πληρώσει παραπάνω καθώς μπορεί επιλεχθούν (με μικρή πιθανότητα αλλά υπαρκτή) παραπάνω από 1 μικροπληρωμές ήδη από τις πρώτες συναλλαγές. Οι S.Micali και RL.Rivest το 2002 στο "Micropayments Revisited" [21] προτείνουν μια καλύτερη λύση.

Έχουμε ένα σχήμα μικροπληρωμών, ακόμη πιο αποδοτικό, το οποίο δεν χρειάζεται την προαναφερθείσα αλληλεπίδραση. Ο πάροχος και ο χρήστης έχουν από ένα δημόσιο κλειδί pk, και υπάρχει διαθέσιμη μια συνάρτηση $F(.)$ που δέχεται σαν είσοδο κάποιο αλφαριθμητικό και δίνει σαν έξοδο έναν αριθμό στο διάστημα $(0, 1)$. Η συναλλαγές (μικροπληρωμές) χαρακ-

τηρίζονται από το δημόσιο κλειδί του παρόχου και το χρήστη, από την τράπεζα, την τιμή, τον χρόνο και ίσως άλλες πληροφορίες.

Το σχήμα λειτουργεί ως εξής :

1. Ο χρήστης x στέλνει στον πάροχο p την συναλλαγή T, υπογεγραμμένη. $(sig_x(T))$
2. Ελέγχεται από τον πάροχο εαν $F((sig_x(T))) < s$, και σε αυτή την περίπτωση η μικροπληρωμή T επιλέγεται.
3. Ο πάροχος στέλνει στην τράπεζα τις $sig_x(T)$ και $sig_p(sig_x(T))$.
4. Εαν οι υπογραφές είναι έγκυρες τότε η τράπεζα χρεώνει τον χρήστη με $\frac{1}{s} * \textit{τιμή μικροπληρωμής}$ και την προμήθεια, και πιστώνει το αντίστοιχο ποσό στον πάροχο.

Εν τέλει έχουμε φτιάξει ένα σύστημα χωρίς αλληλεπίδραση, με το οποίο επιλέγονται τυχαία στοιχεία από ένα σύνολο σύμφωνα με μια πιθανότητα επιλογής s, και κανένας από τους συμμετέχοντες δεν μπορεί να το χρησιμοποιήσει προς όφελος του.

Περιγραφή Αλγορίθμου

Το σχήμα μικροπληρωμών που περιγράφτηκε μπορεί να τροποποιηθεί κατάλληλα ώστε να χρησιμοποιηθεί στο δίκτυο του Algorand. Οι χρήστες, χωρίς αλληλεπίδραση και ιδιωτικά, θα τρέχουν τον αλγόριθμο στην αρχή κάθε βήματος του πρωτοκόλλου σύμφωνα με τον οποίο θα βλέπουν εάν έχουν κάποιο ρόλο στο συγκεκριμένο σημείο. Εάν επιλεχθούν σε κάποιο βήμα ενός γύρου, οποιοσδήποτε χρήστης μπορεί να το ελέγξει τρέχοντας τον αλγόριθμο Verify Sortition.

Αντί για ψηφιακές υπογραφές που περιέχουν την πληροφορία της συναλλαγής, μπορούμε να χρησιμοποιήσουμε μια ψευδοτυχαία τιμή. Είναι σημαντικό η πληροφορία αυτή να φαίνεται τυχαία αλλά να μπορεί να επαληθευτεί διότι καθορίζει την επιλογή ή όχι ενός χρήστη από το σύνολο. Για τον λόγο αυτό, στο Cryptographic Sortition χρησιμοποιείται το κρυπτογραφικό εργαλείο Verifiable Random Functions που παρέχει ψευδοτυχαία αποτελέσματα αλλά επαληθευόμενα για όποιον γνωρίζει το δημόσιο κλειδί που τα παρήγαγε. Αντί για την συνάρτηση $F(.)$ που αντιστοιχίζει strings σε αριθμούς στο $(0, 1)$, στον αλγόριθμο του Algorand κανονικοποιείται το αποτέλεσμα των VRFs ώστε να ανήκει στο παραπάνω διάστημα. Τέλος, δεν υπάρχει τρίτο μέλος, όπως η τράπεζα στο Micropayment Scheme, αλλά είναι αναγκαίο να κοινοποιηθούν τα αποτελέσματα στο ομότιμο δίκτυο του Algorand.

Το πρωτόκολλο του Algorand χρησιμοποιεί σαν μηχανισμό ομοφωνίας το proof-of-stake, και άρα ο αλγόριθμος επιλογής χρηστών που θα πράξουν σε κάποιο βήμα θα πρέπει να λειτουργεί με βάση αυτόν τον άξονα. Δηλαδή, θα πρέπει οι χρήστες με κάποιο τρόπο να επιλέγονται με βάση το stake που έχουν στο σύστημα. Για να είναι δίκαιη η επιλογή και όλοι οι χρήστες να συμμετέχουν με ίσους όρους, οι μονάδες που θα συμ-

μετέχουν δεν θα είναι οι ίδιοι (το δημόσιο κλειδί τους σε σχέση με τα υπόλοιπα δημόσια κλειδιά), αλλά το κάθε νόμισμα που διαθέτουν ξεχωριστά σε σχέση με τα υπόλοιπα κρυπτονομίσματα του Algorand. Ο χρήστης θα είναι πιθανό να επιλεχθεί πάνω από μια φορά, ανάλογα με πόσα από τα νομίσματα του θα επιλεχθούν. Με άλλα λόγια, ένας χρήστης που διαθέτει w κρυπτονομίσματα, μπορεί να επιλεχθούν k από αυτά για να συμμετέχουν σε κάποιο βήμα, με πιθανότητα επιλογής p για το καθένα ξεχωριστά.

Τι κατανομή ακολουθεί η επιλογή x χρηστών;

Από τα W κρυπτονομίσματα θέλουμε να επιλεχθούν ακριβώς x. Άρα έχουμε $\binom{W}{x}$ υποσύνολα του W από x χρήστες. Για κάθε υποσύνολο η πιθανότητα να επιλεχθούν x με πιθανότητα επιλογής του καθενός p, είναι $p^x(1-p)^{W-x}$. Άρα η συνολική πιθανότητα είναι :

$$\binom{W}{x} p^x (1-p)^{W-x}$$

Τελικά η πιθανότητα να επιλεχθούν x χρήστες από τους W ακολουθεί διωνυμική κατανομή με παραμέτρους x, W και p.

Διωνυμική κατανομή

Η διωνυμική κατανομή αποτελεί μια διακριτή συνάρτηση κατανομής κάποιας τυχαίας μεταβλητής X, η οποία εκφράζει τις επιτυχίες ανεξάρτητων πειραμάτων επιτυχιών-αποτυχιών. Τότε στα n πειράματα, με πιθανότητα επιτυχίας κάθε πειράματος p, η πιθανότητα να έχουμε k επιτυχίες εκφράζεται ως εξής :

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

Τότε λέμε ότι η τ.μ. X ακολουθεί διωνυμική κατανομή με παραμέτρους k, n και p. Δηλαδή $X \sim B(k; n, p)$

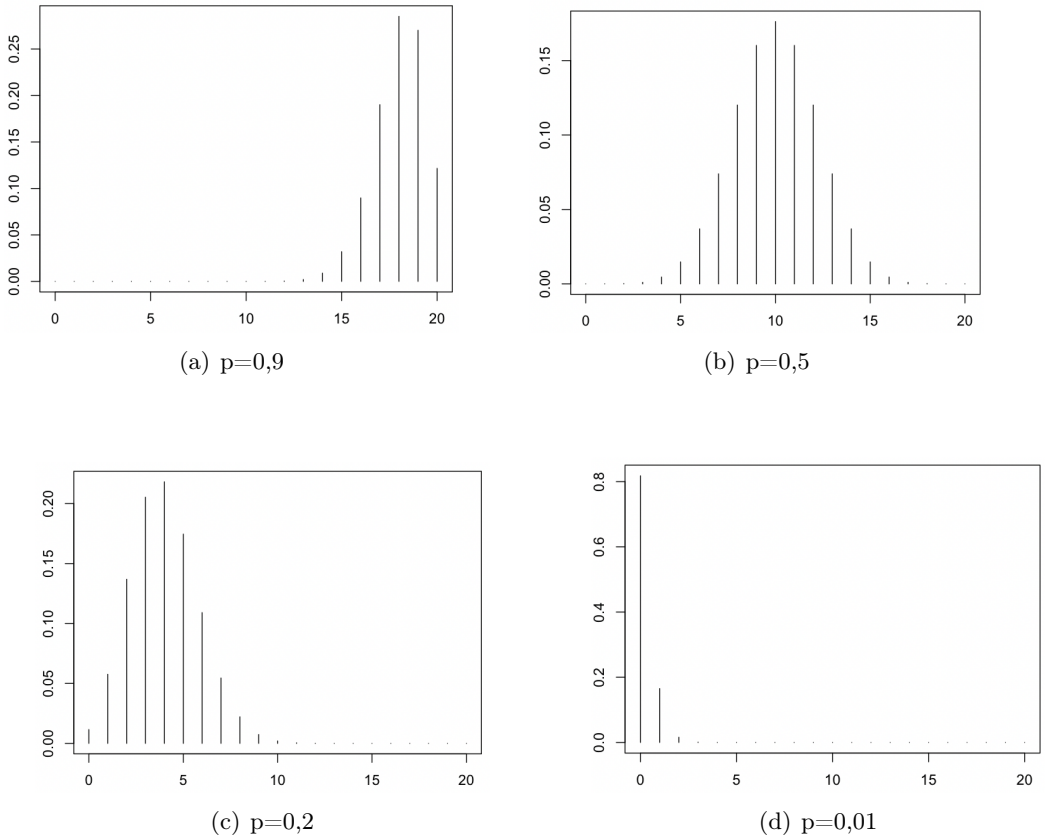
Η πιθανότητα οι επιτυχίες να είναι λιγότερες ή ίσες με τον αριθμό j εκφράζεται με την αθροιστική συνάρτηση κατανομής (cumulative distribution function) :

$$F_X(k) = P(X \leq k) = \sum_{j=0}^{k} B(j; n, p)$$

Πώς προσδιορίζεται η πιθανότητα επιλογής κάθε χρήστη p;

Ένας καλός τρόπος ελέγχου του μεγέθους της επιτροπής που θα δημιουργείται από τον αλγόριθμο Cryptographic Sortition, είναι ο έλεγχος της πιθανότητας p της διωνυμικής κατανομής. Παρακάτω υπάρχουν κάποια διαγράμματα της διωνυμικής κατανομής για τις διάφορες τιμές του p, όταν

οι υπόλοιπες παράμετροι παραμένουν σταθερές. Για μεγαλύτερη ευκολία χρησιμοποιήθηκε $W = 20$. Τα διαγράμματα έχουν γίνει με χρήση του προγράμματος R, όπως φάινεται στο σχήμα 6.3.



(a) p=0,9 (b) p=0,5

(c) p=0,2 (d) p=0,01

Σχήμα 6.2: Διαγράμματα διωνυμικής κατανομής για p=0,9/0,5/0,2/0,01

```
> plot(0:20,dbinom(0:20,size=20,prob=0.5),type="h")
> plot(0:20,dbinom(0:20,size=20,prob=0.9),type="h")
> plot(0:20,dbinom(0:20,size=20,prob=0.2),type="h")
> plot(0:20,dbinom(0:20,size=20,prob=0.01),type="h")
```

Σχήμα 6.3: Εντολές διαγραμμάτων στο περιβάλλον της R

Παρατηρούμε στο σχήμα 6.2, πως για μεγάλη p στο διάγραμμα (a) οι τιμές που είναι πιθανό να επιλεχθούν είναι κοντά στο W. Αντιθέτως, όσο μικραίνουμε την πιθανότητα p στα διαγράμματα (b) και (c) αντίστοιχα η καμπύλη κινείται προς το μηδέν. Μας ενδιαφέρει μια πολύ μικρή τιμή του p όπως βλέπουμε στο διάγραμμα (d), διότι τότε θα επιλέγεται μικρός αριθμός χρηστών σε σχέση με το σύνολο W.

Για να είναι δίκαιη η επιλογή, θα πρέπει η τιμή p να είναι ανεξάρτητη των κρυπτονομισμάτων w που διαθέτει ο κάθε χρήστης. Κάθε νόμισμα δηλαδή θα πρέπει να συμμετέχει με την ίδια πιθανότητα. Ορίζουμε λοιπόν ως p την πιθανότητα $\frac{\tau}{W}$ ώστε να μπορεί να ελεγχθεί με την παράμετρο τ το μέγεθος της επιτροπής. Συγκεκριμένα, για κάθε νόμισμα από τα W με πιθανότητα επιτυχίας p έχω συνολικά $W * p$ επιτυχίες $\Rightarrow W * \frac{\tau}{W} \Rightarrow \tau$ επιτυχίες (διότι έχω W ανεξάρτητα πειράματα).

Ποιά είναι η πιθανότητα να επιλεχθούν ακριβώς x χρήστες;

Όπως αναφέρθηκε η πιθανότητα να επιλεχθούν x χρήστες ακολουθεί διωνυμική κατανομή και η πιθανότητα p που χρησιμοποιείται σαν παράμετρος στην κατανομή είναι : $p = \frac{\tau}{W}$. Ο αριθμός των κρυπτονομισμάτων W που κυκλοφορούν στο δίκτυο του Algorand είναι αρκετά μεγάλος ($\sim$ 7.000.000.000). Συνεπώς η ποσότητα $\frac{\tau}{W}$ είναι πολύ μικρή. Άρα :

$$\binom{W}{x} p^x (1-p)^{W-x} =$$

$$\binom{W}{x} (\frac{\tau}{W})^x (1-(\frac{\tau}{W}))^{W-x} =$$

$$\frac{W!}{x!(W-x)!} (\frac{\tau}{W})^x (1-(\frac{\tau}{W}))^{W-x} =$$

$$\frac{(W-x+1)(W-x+2)...W}{W^x} \frac{\tau^x}{x!} (1-\frac{\tau}{W})^{W-x}$$

- Ο αριθμητής $(W-x+1)(W-x+2)...W$ έχει x παράγοντες.
  Επειδή $W >> x \Rightarrow (W-x+i) \approx W$ για κάθε $i = 1, 2, ...x$

  Άρα :
  $$\frac{(W-x+1)(W-x+2)...W}{W^x} \approx \frac{W^x}{W^x} \approx 1$$

- Ο τελευταίος όρος της παράστασης γίνεται για μεγάλο W:

  $$(1-\frac{\tau}{W})^{W-x} = \frac{(1-\frac{\tau}{W})^W}{(1-\frac{\tau}{W})^x} \approx \frac{(1-\frac{\tau}{W})^W}{1} \approx (1-\frac{\tau}{W})^W$$

  Και υπολογίζουμε χρησιμοποιώντας όρια :

$$\lim_{w\to\infty}(1-\frac{\tau}{W})^W = \lim_{w\to\infty} e^{log(1-\frac{\tau}{W})^W} =$$

$$\lim_{w\to\infty} e^{Wlog(1-\frac{\tau}{W})} = e^{\lim_{w\to\infty} Wlog(1-\frac{\tau}{W})} =$$

$$e^{\lim_{w\to\infty}\frac{log(1-\frac{\tau}{W})}{\frac{1}{W}}}$$

Βρίσκουμε τις παραγώγους ώστε να γίνει χρήση του κανόνα L'Hopital :

$$\frac{\partial log(1-\frac{\tau}{W})}{\partial W} = \frac{1}{1-\frac{\tau}{W}}(-(-\frac{\tau}{W^2})) = \frac{\tau}{W^2}\frac{1}{1-\frac{\tau}{W}}$$

και

$$\frac{\partial\frac{1}{W}}{\partial W} = -\frac{1}{W^2}$$

Άρα με εφαρμογή του κανόνα :

$$exp[\lim_{w\to\infty}\frac{log(1-\frac{\tau}{W})}{\frac{1}{W}}] =$$

$$exp[\lim_{w\to\infty}\frac{\frac{\tau}{W^2}\frac{1}{1-\frac{\tau}{W}}}{-\frac{1}{W^2}}] =$$

$$exp[\lim_{w\to\infty}-\frac{\tau}{1-\frac{\tau}{W}}] =$$

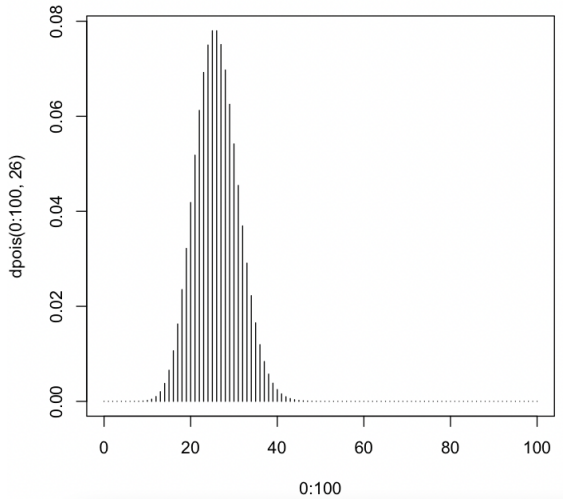
$$exp[-\tau\lim_{w\to\infty}\frac{1}{1-\frac{\tau}{W}}] =$$

$$exp[-\tau] = e^{-\tau}$$

Τελικά η πιθανότητα να επιλεχθούν ακριβώς x χρήστες είναι :

$$\frac{\tau^x}{x!}e^{-\tau}$$

Δηλαδή η πιθανότητα αυτή ακολουθεί κατανομή Poisson με παράμετρο τ. Για τιμή του τ ίση με 26, βλέπουμε στο παρακάτω διάγραμμα τη πιθανότητα να επιλεχθούν από 0 έως 100 άτομα :



Σχήμα 6.4: Κατανομή Poisson για τ=26 από 0 έως 100

Το διάγραμμα φτιάχτηκε με χρήση του προγράμματος R με την εντολή :

```
> plot(0:100,dpois(0:100,26),type="h")
```

Σχήμα 6.5: Εντολή διαγράμματος στο περιβάλλον της R

Πώς συνδέεται η τυχαιότητα των VRFs με την διωνυμική κατανομή;

Οι επαληθευόμενες συναρτήσεις ψευδοτυχαίου αποτελέσματος παράγουν μια ψευδοτυχαία τιμή, $hash$, η οποία είναι ομοιόμορφα κατανεμημένη στο διάστημα $[0, 2^{hashlen} - 1]$, όπου $hashlen$ είναι το μήκος του $hash$.

Επειδή η πιθανότητα να επιλεχθεί κάποιο νόμισμα από τα w που έχει ο χρήστης ακολουθεί διωνυμική κατανομή, θέλουμε να κανονικοποιήσουμε το $hash$ στο διάστημα [0,1) ώστε να μπορεί να συγκριθεί με την προαναφερθείσα πιθανότητα. Άρα:

$$hash \sim U[0, 2^{hashlen} - 1] \Rightarrow \frac{hash}{2^{hashlen}} \sim U[0, 1)$$

Η συνάρτηση πυκνότητας πιθανότητας της διωνυμικής κατανομής δίνει ακριβώς την τιμή της πιθανότητας. Για να μπορέσουμε να συγκρίνουμε την τιμή $hash$ με την διωνυμική κατανομή θα πρέπει να φτιάξουμε διαδοχικά διαστήματα από στο [0,1). Αυτό θα συμβεί με την χρήση της αθροιστικής συνάρτησης κατανομής της διωνυμικής : $\sum_{j=0}^{k} B(j; w, p)$, διότι $\sum_{k=0}^{w} B(k; w, p) = 1$. Κάθε διάστημα θα είναι η πιθανότητα να επιλεχθούν ακριβώς k νομίσματα από τα w που διαθέτει ο χρήστης. Τα διαστήματα είναι της μορφής :

$$I_j = \left[\sum_{k=0}^{j} B(k; w, p), \sum_{k=0}^{j+1} B(k; w, p) \right)$$

Εάν η κανονικοποιημένη τιμή $\frac{hash}{2^{hashlen}}$ πέσει σε κάποιο από τα διαδοχικά διαστήματα I_j με $j = 0, 1, ..w$, τότε ο χρήστης έχει ακριβώς j ψήφους στο βήμα αυτό.

Πώς αντιμετωπίζεται το πρόβλημα των sybil attacks;

Ο αλγόριθμος Cryptographic Sortition μπορεί να γίνει εύκολα στόχος για την εφαρμογή μιας επίθεσης τύπου sybil attack, καθώς κάποιος κακόβουλος χρήστης μπορεί να δημιουργήσει σκόπιμα πολλούς λογαριασμούς στο Algorand ώστε να έχει περισσότερους ψήφους. Έτσι θα διαμοιράσει σε κάθε λογαριασμό τα νομίσματα w και αντί να εκτελέσει μια φορά τον αλγόριθμο, θα τον εκτελέσει μια για κάθε λογαριασμό που έχει.

Η διωνυμική κατανομή όμως έχει μια πού σημαντική ιδιότητα :

$$B(k_1; w_1, p) + B(k_2; w_2, p) = B(k_1 + k_2; w_1 + w_2, p)$$

Αυτή η ιδιότητα μας λέει πως η πιθανότητα να επιλεχθούν k_1 νομίσματα από τα w_1 και k_2 νομίσματα από τα w_2 σε διαφορετικά πειράματα, είναι ίδια με το να γίνει ένα πείραμα για τα $w_1 + w_2$. Συνεπώς το πρωτόκολλο προστατεύεται από αυτού του είδους την επίθεση.

Ο αλγόριθμος

Algorithm 3: Sortition(sk, seed, τ, role, w, W)

$< hash, \pi > \leftarrow VRF_{sk}(seed||role)$

$p \leftarrow \frac{\tau}{W}$

$j \leftarrow 0$

while $\frac{hash}{2^{hashlen}} \notin \left[\sum_{k=0}^{j} B(k; w_i, p), \sum_{k=0}^{j+1} B(k; w_i, p)\right]$ **do**

$\lfloor\ j++$

return $< hash, \pi, j >$

Είσοδος

Ο αλγόριθμος *Sortition* δέχεται σαν ορίσματα το ιδιωτικό κλειδί του χρήστη (sk), έναν σπόρο τυχαιότητας $(seed)$, μια παράμετρος που οριοθετεί τον αριθμό των χρηστών που θα επιλεχθούν ανάλογα με το βήμα του πρωτοκόλλου (τ), η επιτροπή που πρόκειται να συμμετάσχει ο χρήστης $role$, τα κρυπτονομίσματα του χρήστη που χρησιμοποιεί τον αλγόριθμο (w), και τα συνολικά κρυπτονομίσματα που τρέχουν στο σύστημα (W).

Έξοδος

Η έξοδος του *Sortition* είναι τρείς τιμές. Οι $hash$ και π που προέρχονται από τις Verifiable Random Functions, και είναι ανίστοιχα η *τυχαιότητα* με την οποία έχει επιλεχθεί ο χρήστης, και η *απόδειξη* για λόγους επαλήθευσης. Η τελευταία έξοδος του αλγορίθμου είναι ένας αριθμός j, ο οποίος δείχνει πόσα από τα κρυπτονομίσματά του έχουν επιλεχθεί από τον αλγόριθμο. Με άλλα λόγια, πόσες ψήφους έχει ο χρήστης στο συγκεκριμένο βήμα αυτού του γύρου.

Περιγραφή βημάτων

Στο πρώτο βήμα ο αλγόριθμος χρησιμοποιεί τις ψευδοτυχαίες συναρτήσεις επαληθευόμενου αποτελέσματος (VRFs) ώστε να παράξει την τυχαιότητα. Η τυχαιότητα είναι μια τιμή κατακερματισμού φαινομενικά τυχαία σε όποιον δεν γνωρίζει το ζευγάρι κλειδιών του χρήστη. Οποιοσδήποτε γνωρίζει το δημόσιο κλειδί του, μπορεί μέσω της τιμής π να επαληθεύσει ότι το αποτέλεσμα $hash$ προέρχεται από τον χρήστη με κλειδί pk και σπόρο τυχαιότητας $seed$. Η τιμή $role$ χρησιμοποιείται για να προσδιορίζει τον ρόλο που θα έχει η ψήφος στον συγκεκριμένο γύρο (εάν ο χρήστης θα προτείνει block ή εάν θα ψηφίσει για κάποιο ήδη προτεινόμενο block).

Στο δεύτερο βήμα προσδιορίζεται η πιθανότητα επιλογής νομισμάτων ως $p = \frac{\tau}{W}$. Θέλουμε όλα τα νομίσματα να συμμετέχουν με την ίδια πι-

θανότητα σε σχέση με όλα τα υπόλοιπα, και για αυτό τον λόγο επιλέγεται να οριστεί η πιθανότητα ως κλάσμα των νομισμάτων W που υπάρχουν συνολικά στο Algorand. Ταυτοχρόνως, θα πρέπει να υπάρχει ένα άνω όριο στο πόσα από τα συνολικά νομίσματα θα επιλεχθούν. Να μπορεί δηλαδή να ελεγχθεί ο αριθμός των χρηστών που θα συμμετέχουν. Για αυτόν τον λόγο υπάρχει η παράμετρος τ η οποία επιλέγεται κατάλληλα, ανάλογα με το βήμα του γύρου που βρίσκεται το πρωτόκολλο (πιο αναλυτικά εδώ).

Στο τρίτο βήμα υπάρχει η αρχικοποίηση μιας μεταβλητής j με την τιμή 0, η οποία δείχνει τον αριθμό των ψήφων που έχει ο χρήστης σε αυτό το σημείο του πρωτοκόλλου. Για να βρεθεί αυτός ο αριθμός, ελέγχουμε την τυχαιότητα $hash$, που έχει παραχθεί από τις VRFs, σε ποιό σημείο του διαστήματος $[0, 1)$ πέφτει. Για να μπορέσει η τιμή $hash$ να "πέσει" μέσα στο διάστημα $[0, 1)$ θα πρέπει να κανονικοποιηθεί. Άρα παίρνουμε την τιμή $\frac{hash}{2^{hashlen}}$, όπου $hashlen$ είναι το μήκος του $hash$. Προκειμένου να ελεγχθούν όλα τα νομίσματα ξεχωριστά, δημιουργείται μια while loop στην οποία ελέγχεται εάν το κανονικοποιημένο hash υπάρχει σε κάποιο διάστημα I_j. Το κάθε διάστημα είναι η πιθανότητα να επιλεχθούν ακριβώς j νομίσματα, δηλαδή ο χρήστης να έχει j ψήφους (πιο αναλυτικά εδώ). Στο τέταρτο βήμα λοιπόν, σε περίπτωση που κάποια τιμή πέσει στο I_j το while loop τερματίζεται και επιστρέφεται η τιμή hash και η απόδειξη π των VRFs, και οι ψήφοι j του χρήστη.

Ο αλγόριθμος επαλήθευσης

Αφού ο αλγόριθμος *Sortition* εκτελείται ιδιωτικά από τον κάθε χρήστη,

οι υπόλοιποι χρήστες θα πρέπει να μπορούν να επιβεβαιώσουν την επιλογή των μελών της κάθε επιτροπής. Για αυτόν τον λόγο, υπάρχει ένας δεύτερος αλγόριθμος επιβεβαίωσης του *Sortition* , που ονομάζεται *VerifySortition*.

Algorithm 4: VerifySortition(pk, hash, π, seed, τ, role, w, W)

if $\neg VerifyVRF_{pk}(hash, \pi, seed||role)$ **then return** 0;

$p \leftarrow \frac{\tau}{W}$

$j \leftarrow 0$

while $\frac{hash}{2^{hashlen}} \notin \left[\sum_{k=0}^{j} B(k; w_i, p), \sum_{k=0}^{j+1} B(k; w_i, p)\right)$ **do**

$\lfloor\ j++$

return j

Είσοδος
Ο αλγόριθμος *VerifySortition* δέχεται σαν ορίσματα το δημόσιο κλειδί του χρήστη που πρόκειται να επαληθευτεί η επιλογή του σε κάποια επιτροπή pk, η τυχαιότητα και η απόδειξή της που είναι οι έξοδοι το αλγορίθμου *Sortition* $(hash, \pi)$, ο σπόρος τυχαιότητας $seed$, μια παράμετρος που ορισθετεί τον αριθμό των χρηστών που θα επιλεχθούν ανάλογα με το βήμα του πρωτοκόλλου (τ), η επιτροπή που συμμετέχει ο επιλαχών χρήστης $role$, τα κρυπτονομίσματα του χρήστη που πρόκειται να επαληθευτεί (w), και τα συνολικά κρυπτονομίσματα που τρέχουν στο σύστημα (W).

Έξοδος
Ο αλγόριθμος *VerifySortition* επιστρέφει ως έξοδο έναν αριθμό j που μπορεί να πάρει ακέραιες τιμές από το 0 έως τον αριθμό w (όπου w τα κρυπτονομίσματα του χρήστη που πρόκειται να επαληθευτεί). Ο αριθμός αυτός αντιπροσωπεύει τους ψήφους που έχει ο χρήστης με δημόσιο κλειδί pk. Σε περίπτωση που κάποιο από τα στοιχεία $pk, hash, \pi$ ή $seed$ δεν είναι έγκυρο, ο αλγόριθμος σταματά επιστρέφοντας 0.

Περιγραφή βημάτων
Στο πρώτο βήμα του αλγορίθμου γίνεται η επαλήθευση της τυχαιότητας $hash$ που χρησιμοποιείται στον αλγόριθμο *Sortition*. Η διαδικασία αυτή περιλαμβάνει την κλήση της συνάρτησης επαλήθευσης των *VRFs*, με είσοδο τις τιμές $pk, hash, \pi$ και $seed$. Η συνάρτηση $VerifyVRF$ επιστρέφει Yes στην περίπτωση που τα δεδομένα εισόδου είναι έγκυρα και No διαφορετικά. Σε περίπτωση που η συνάρτηση αυτή επιστρέψει Yes ο αλγόριθμος συνεχίζει στο δεύτερο βήμα, διαφορετικά επιστρέφει 0 και τερματίζει.

Τα υπόλοιπα βήματα είναι τα ίδια με αυτά του *Sortition*, μόνο που στο τελευταίο βήμα ο αλγόριθμος επιστρέφει μόνο τον αριθμό j που είναι οι ψήφοι που έχει ο χρήστης στο βήμα αυτού του πρωτοκόλλου.

6.8 Seed

Ο σπόρος (seed) είναι ένα αλφαριθμητικό (string), που χρησιμοποιείται ως είσοδο σε μια ντετερμινιστική γεννήτρια τυχαίων αριθμών ή bit. Το string αυτό χρησιμοποιείται στα πρώτα βήματα της γεννήτριας τυχαίων αριθμών ώστε να παράξει τα πρώτα bit τυχαιότητας. Για τον λόγο αυτό είναι πολύ σημαντικό να έχει παραχθεί κατάλληλα. Λόγω του ότι και οι Verifiable Random Functions είναι γεννήτριες ψευδοτυχαιότητας, χρειάζονται κάποιο ασφαλές $seed$.

Στο Algorand, ο σπόρος είναι δημοσίως γνωστός σε όλους τους χρήστες και ανανεώνεται πριν την αρχή του κάθε γύρου. Ο σπόρος τυχαιότητας του πρώτου γύρου παράγεται από μια διαμοιρασμένη γεννήτρια τυχαίων αριθμών. Στην συνέχεια σε κάθε γύρο r ανανεώνεται μέσω των VRFs ως εξής :

$$< seed_r, \pi > \leftarrow VRF_{sk}(seed_{r-1}||r)$$

Δηλαδή για να παραχθεί ο σπόρος για κάποιο γύρο r, καλείται η συνάρτηση υπολογισμού των VRF με είσοδο το ιδιωτικό κλειδί του χρήστη που υπολογίζει (sk), τον σπόρο του προηγούμενου γύρου $(seed_{r-1})$ και τον αριθμό του γύρου που βρίσκεται το πρωτόκολλο (r). Η έξοδος της συνάρτησης είναι ο καινούριος σπόρος τυχαιότητας $seed_r$ και η απόδειξη π για την επαλήθευσή του.

Για να είναι γνωστός ο σπόρος σε όλους τους χρήστες στην αρχή του κάθε γύρου r, υπολογίζεται από τους block proposers του προηγούμενου βήματος και προστίθεται στο προτεινόμενο block του γύρου $r-1$. Άρα το καινούριο block που θα προστεθεί στον γύρο $r-1$ θα περιέχει τον $seed_r$.

Γενικότερα οι σπόροι μπορούν να παράξουν τυχαιότητα για κάποιους γύρους, παρόλα αυτά έχουν περιορισμένη διάρκεια "ζωής". Στο Algorand, ο σπόρος που χρησιμοποιείται ανανεώνεται κάθε R γύρους. Πιο συγκεκριμένα στον γύρο r, χρησιμοποιείται ο σπόρος $seed_{r-1-(rmodR)}$. Άρα σε κάθε γύρο r υπολογίζεται ο $seed_r$ αλλά χρησιμοποιείται ο $seed_{r-1-(rmodR)}$.
Παράδειγμα :
Για R=5:

1. *r=10*
   - Υπολόγισε : $< seed_{10}, \pi > \leftarrow VRF_{sk}(seed_9||10)$
   - Χρησιμοποίησε : $seed_{9-(9mod5)} = seed_5$
2. *r=11*
   - Υπολόγισε : $< seed_{11}, \pi > \leftarrow VRF_{sk}(seed_{10}||11)$
   - Χρησιμοποίησε : $seed_{10-(10mod5)} = seed_{10}$
3. *r=12*
   - Υπολόγισε : $< seed_{12}, \pi > \leftarrow VRF_{sk}(seed_{11}||12)$
   - Χρησιμοποίησε : $seed_{11-(11mod5)} = seed_{10}$
4. *r=13*
   - Υπολόγισε : $< seed_{13}, \pi > \leftarrow VRF_{sk}(seed_{12}||13)$
   - Χρησιμοποίησε : $seed_{12-(12mod5)} = seed_{10}$
5. *r=14*
   - Υπολόγισε : $< seed_{14}, \pi > \leftarrow VRF_{sk}(seed_{13}||14)$
   - Χρησιμοποίησε : $seed_{13-(13mod5)} = seed_{10}$
6. *r=15*
   - Υπολόγισε : $< seed_{15}, \pi > \leftarrow VRF_{sk}(seed_{14}||15)$

- Χρησιμοποίησε : $seed_{14-(14 mod 5)} = seed_{10}$

7. <u>r=16</u>
   - Υπολόγισε : $< seed_{16}, \pi > \leftarrow VRF_{sk}(seed_{15}||16)$
   - Χρησιμοποίησε : $seed_{15-(15 mod 5)} = seed_{15}$

Γιατί ο σπόρος ανανεώνεται κάθε R γύρους;

Το δίκτυο του Algorand θεωρείται μερικώς συγχρονισμένο. Δηλαδή υπάρχει περίπτωση για ένα χρονικό διάστημα b να είναι τελείως ασύγχρονο, μέχρι να επιστρέψει σε συγχρονισμό. Σε αυτό το διάστημα ένας κακόβουλος χρήστης μπορεί να έχει τον πλήρη έλεγχο του δικτύου. Αυτό σημαίνει πως μπορεί να πείσει χρήστες να παράξουν τα μελλοντικά seeds ώστε να μπορεί να έχει έλεγχο σε μελλοντικές ψηφοφορίες. Άρα θέλουμε οι χρήστες που υπολογίζουν τον σπόρο, και άρα αυτοί που εκτελούν τον αλγόριθμο Cryptographic Sortition να επιλέγονται κατάλληλα b χρόνο πριν από κάθε γύρο r-1-(r mod R).
Οπότε σε κάθε Sortition σε γύρο r :

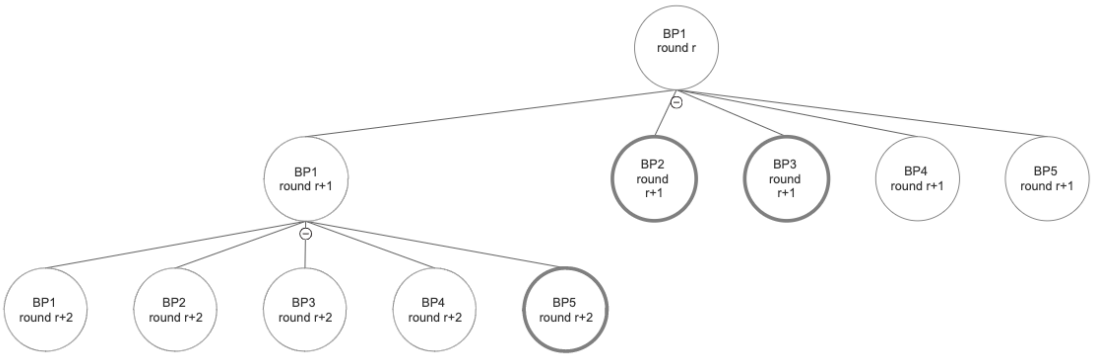
1. Ανάκτηση αριθμού γύρου του block r-1-(r mod R).
2. Χρήση λογαρισμών (κλειδιά και αντίστοιχα κρυπτονομίσματα) που υπήρχαν b-χρόνο πριν το block αυτό.

Είναι σημαντικό να γίνει υπολογισμός του αριθμού των block που μπορεί να ελέγξει ο αντίπαλος, ώστε ο σπόρος seed να είναι μην μπορεί να προβλεπτεί. Μέσα στους χρήστες με δικαίωμα πρότασης block, θα υπάρχουν και ειλικρινείς αλλά ίσως και κακόβουλοι χρήστες. Εάν ο χρήστης με την μεγαλύτερη προτεραιότητα είναι ειλικρινής, θα ακολουθήσει το πρωτόκολλο και θα προτείνει επιτυχώς ένα block, και σε όλη αυτή η διαδικασία δεν μπορεί να επέμβει ο αντίπαλος. Αντίθετα, εάν ο χρήστης με την μεγαλύτερη προτεραιότητα είναι κακόβουλος, τότε ο αντίπαλος έχει το δικαίωμα να τον συντονίσει με τους υπόλοιπους κακόβουλους χρήστες του δικτύου και να διαλέξει ποιός από αυτούς θα προτείνει το καινούριο block. Αυτό είναι ένα πολύ σημαντικό προβάδισμα για τον αντίπαλο, καθώς μπορεί να διαλέξει τον μελλοντικό σπόρο που τον συμφέρει ώστε να αυξήσει την πιθανότητα να επιλεχθεί κακόβουλος χρήστης ως block proposer στον επόμενο γύρο. Το παρακάτω θεώρημα μας διασφαλίζει ότι ο αντίπαλος δεν μπορεί με αυτόν τον τρόπο να επηρεάσει το πρωτόκολλο.

Θεώρημα 6.8.1. *Σε περίοδο αυστηρού συγχρονισμού, η πιθανότητα ένας αντίπαλος να υπολογίσει τους επόμενους k σπόρους, μειώνεται εκθετικά με το k.*

Απόδειξη
Έστω το κλάσμα των ειλικρινών χρηστών h. Εάν ο block proposer του γύρου r είναι ειλικρινής, τότε και ο σπόρος του γύρου r+1, $seed_{r+1}$, έχει παραχθεί σωστά. Άρα στον επόμενο γύρο με έμπιστο σπόρο, θα έχουμε και έμπιστες ποσότητες <hash, π> ομοιόμορφα κατανεμημένες για τον κάθε χρήστη. Συνεπώς η πιθανότητα το block με την μεγαλύτερη προτεραιότητα να ανήκει σε έμπιστο χρήστη είναι h, και αυτός θα ακολουθήσει το πρωτόκολλο. Παρομοίως, και οι υπόλοιποι εν δυνάμει block

proposers θα έχουν πιθανότητα h να είναι ειλικρινείς.
Μας ενδιαφέρει να ερευνήσουμε τα αναμενόμενα κακόβουλα blocks που υπάρχουν μεταξύ δύο ειλικρινών block. Προφανώς εάν οι block proposers του γύρου r και r+1 είναι ειλικρινείς, τότε υπάρχουν 0 τέτοια block. Εάν ο block proposer του γύρου r+1 είναι κακόβουλος, τότε τα αναμενόμενα κακόβουλα blocks είναι σίγουρα ≥ 1. Ο ακριβής αριθμός σχετίζεται με τον αριθμό των υπόλοιπων κακόβουλων χρηστών που υπάρχουν στην ακολουθία των εν δυνάμει block proposers του γύρου αυτού, πριν από τους ειλικρινείς χρήστες. Εάν έχουμε c τέτοιους χρήστες, τότε τα αναμενόμενα κακόβουλα blocks είναι $c + 1$.
Η απόδειξη θα γίνει με την χρήση δέντρου. Κάθε κόμβος είναι ένας ειλικρινής ή κακόβουλος χρήστης που έχει επιλεχθεί από τον αλγόριθμο *Sortition*. Η ρίζα του δέντρου είναι ο τελευταίος ειλικρινής block proposer με την μεγαλύτερη προτεραιότητα. Τα παιδιά κάθε κόμβου είναι η ακολουθία των block proposers ταξινομημένη απο τα αριστερά με τον χρήστη με την μεγαλύτερη προτεραιότητα, προς τα δεξιά. ¨Ενα παράδειγμα τέτοιου δέντρου είναι στο σχήμα 3.6, όπου οι κόμβοι με τον έντονο κύκλο είναι κόμβοι κακόβουλων χρηστών, ενώ οι υπόλοιποι είναι ειλικρινείς χρήστες.



Σχήμα 6.6: Δέντρο block proposing

Η πιθανότητα εμφάνισης ειλικρινούς block proposer στην ακολουθία των εν δυνάμει block proposers σε κάθε γύρο είναι h. Αντίστοιχα, η πιθανότητα εμφάνισης ενός κακόβουλου block proposer στην ακολουθία είναι 1-h. Έστω $f(k)$ η πιθανότητα να υπάρχουν τουλάχιστον k συνεχόμενα μη-έμπιστα block στην αλυσίδα, πριν από τον πρώτο έμπιστο. Άρα επειδή η πιθανότητα να υπάρχει ένας κακόβουλος χρήστης στην ακολουθία πριν από κάποιο έμπιστο είναι 1-h, η πιθανότητα να υπάρχει ένα κακόβουλο block πριν από το επόμενο έμπιστο είναι : $f(1) = 1 - h$. Θα λειτουργήσουμε επαγωγικά για να φράξουμε την $f(k)$, χρησιμοποιώντας ως βάση την $f(1)$. Άρα:

- ***Βάση***: Η πιθανότητα να υπάρχει ένα κακόβουλο block πριν από το επόμενο έμπιστο είναι $f(1) = 1 - h$
- ***Επαγωγική υπόθεση***: Έστω ότι ισχύει η υπόθεση πιθανότητας $f(k)$, για τουλάχιστον k συνεχόμενα block.

- ***Επαγωγικό βήμα***: Έστω ο γύρος r στον οποίο έχει παραχθεί ένα έμπιστο block και είναι n ρίζα του δέντρου μας. Έστω ότι υπάρχουν c κακόβουλα παιδιά, αριστερά από το πρώτο έμπιστο. Τότε ο αντίπαλος έχει να επιλέξει μεταξύ c block και ενός κενού, άρα $c+1$ επιλογές για τον σπόρο $seed_{r+1}$. Για να μπορέσει να φτιάξει $k+1$ συνεχόμενα block στην αλυσίδα, θα πρέπει κάποια από τις $c+1$ επιλογές να του δίνει το δικαίωμα να φτιάξει k συνεχόμενα block. Η πιθανότητα τουλάχιστον μια από τις $c+1$ επιλογές να δώσει k συνεχόμενα κακόβουλα block, είναι σύμφωνα με την ανισότητα Boole :

$$\leq \sum_{i=1}^{c} \Pr[\text{child i gives k malicious blocks}]$$

$$= (c+1)f(k)$$

Επιπλέον, η πιθανότητα τα πρώτα c παιδιά της ρίζας να είναι κακόβουλα είναι :

$$(1-h)^c h$$

Άρα συνολικά η πιθανότητα να έχουμε $k+1$ συνεχόμενα κακόβουλα block στην αλυσίδα είναι:

$$f(k+1) \leq \sum_{c=1}^{\infty} (1-h)^c h(c+1)f(k)$$

(1) Άρα πρέπει να υπολογιστεί η ποσότητα :

$$\sum_{c=1}^{\infty} (1-h)^c (c+1)$$

Από την γεωμετρική σειρά έχουμε :

$$\sum_{c=1}^{\infty} c(1-h)^{c-1} = \frac{1}{1-(1-h)^2} \Rightarrow$$

$$\sum_{c=1}^{\infty} c(1-h)^{c-1} = \frac{1}{h^2} \Rightarrow$$

$$\sum_{c=1}^{\infty} c\frac{(1-h)^c}{1-h} = \frac{1}{h^2} \Rightarrow$$

$$\frac{1}{1-h}\sum_{c=1}^{\infty} c(1-h)^c = \frac{1}{h^2} \Rightarrow$$

$$\sum_{c=1}^{\infty} c(1-h)^c = \frac{1-h}{h^2} \Rightarrow$$

Άρα :

$$\sum_{c=1}^{\infty}(c+1)(1-h)^c = \sum_{c=1}^{\infty}[c(1-h)^c + (1-h)^c] =$$

$$= \sum_{c=1}^{\infty} c(1-h)^c + \sum_{c=1}^{\infty}(1-h)^c = \frac{1-h}{h^2} + \sum_{c=0}^{\infty}(1-h)^c - 1 =$$

$$= \frac{1-h}{h^2} + \frac{1}{1-(1-h)} - 1 = \frac{1-h}{h^2} + \frac{1}{h} - 1 = \frac{1-h}{h^2} + \frac{1-h}{h} =$$

$$= \frac{(1-h)+(1-h)h}{h^2} = \frac{(1-h)(1+h)}{h^2}$$

Τελικά :

$$\sum_{c=1}^{\infty}(1-h)^c(c+1) = \frac{(1-h)(1+h)}{h^2}$$

και οπότε η (1) γίνεται :

$$f(k+1) \leq \frac{(1-h)(1+h)}{h^2} h f(k) \Rightarrow$$

$$f(k+1) \leq \frac{(1-h)(1+h)}{h} f(k) \Rightarrow$$

όπου k μπαίνει το k-1 :

$$f(k) \leq \frac{(1-h)(1+h)}{h} f(k-1) \text{ (a)}$$

όπου k μπαίνει το k-1 :

$$f(k-1) \leq \frac{(1-h)(1+h)}{h} f(k-2) \text{ (b)}$$

άρα από (a),(b) και επαγωγικά:

$$f(k) \leq [\frac{(1-h)(1+h)}{h}]^2 f(k-2) \Rightarrow$$

$$\vdots$$

$$f(k) \leq [\frac{(1-h)(1+h)}{h}]^{k-1} f(1) \Rightarrow$$

$$f(k) \leq [\frac{(1-h)(1+h)}{h}]^{k-1}(1-h)$$

Στο πρωτόκολλο υπάρχει η γενική υπόθεση ότι $h \geq \frac{2}{3}$ οπότε:

$$f(k) \leq \frac{1}{3}(0,84)^{k-1} \Rightarrow f(k) \leq (0,84)^k$$

□

Άρα ο αντίπαλος έχει πιθανότητα να ελέγξει 20 συνεχόμενα seed $\leq 0,84^{20} \leq 0,0306$.

Τι γίνεται στην περίπτωση μη έγκυρου seed;
Επειδή ο σπόρος του κάθε γύρου παράγεται από τους χρήστες που προτείνουν block, υπάρχει περίπτωση να μην παραχθεί σωστά και άρα να μην είναι έγκυρος. Επειδή οι χρήστες πάντα ελέγχουν τον σπόρο σε κάθε block και σε κάθε ψηφοφορία, εάν υπάρχει μη έγκυρος σπόρος τότε το αντίστοιχο block θεωρείται κενό, και ο σπόρος που χρησιμοποιείται υπολογίζεται ως εξής :

$$seed_r = H(seed_{r-1}||r)$$

Παρατήρηση 6.8.1. Η παραγωγή μη έγκυρου σπόρου είναι αδύνατο να συμβεί εάν χρησιμοποιηθούν οι Verifiable Random Functions, καθώς έχουν την ιδιότητα να παράγουν ψευδοτυχαία αποτελέσματα ακόμη και στην περίπτωση που ο χρήστης είναι κακόβουλος και τα κλειδιά του δεν έχουν παραχθεί σωστά.

6.9 Block Proposal

Όπως προαναφέρθηκε, προκειμένου να διατηρείται ασφαλές το πρωτόκολλο δεδομένου του μερικώς συγχρονισμένου δικτύου, οι χρήστες που προτείνουν block προέρχονται από προηγούμενο γύρο (παράμετρος k). Οι χρήστες που έχουν επιλεχθεί από τον αλγόριθμο Sortition συγκεντρώνουν τις συναλλαγές που πρόκειται να προστεθούν στην αλυσίδα στην μορφή PAY^r, και φτιάχνουν το καινούριο block. Αυτό περιέχει όλες τις απαραίτητες πληροφορίες που σχετίζονται με τον χρήστη, τις συναλλαγές και τον γύρο που βρίσκεται το πρωτόκολλο.

Κάθε χρήστης i από τον γύρο r-k:

1. Παραλαμβάνει τον $seed_{r-1}$ και το $H(B^{r-1})$
2. Τρέχει τον αλγόριθμο Cryptographic Sortition
3. Εάν έχει επιλεχθεί $(j \neq 0)$:
   (a) Δημιουργεί ένα σύνολο από συναλλαγές που υπάρχουν στο σύστημα PAY^r
   (b) Δημιουργεί το καινούριο block $B_i^r = (r, H(B^{r-1}), SIG_i(seed_{r-1}), PAY_i^r)$
   (c) Υπολογίζει $\sigma_i^{r,s} = (hash, \pi)$
   (d) Διαμοιράζεται το $m_i^r = (B_i^r, SIG_i(B_i^r), \sigma_i^{r,s})$ και το $\sigma_i^{r,s}$ ξεχωριστά

Μια σημαντική λειτουργία σε αυτό το σημείο του πρωτοκόλλου είναι το τελευταίο βήμα 3(d). Αντί να γίνεται διαμοιρασμός ενός μηνύματος από κάθε χρήστη (m_i^r), διαμοιράζονται δύο μηνύματα. Το ένα είναι το m_i^r, που περιλαμβάνει το προτεινόμενο B_i^r και τις σχετικές πληροφορίες, ενώ το δεύτερο είναι το διαπιστευτήριο ή credential του χρήστη $\sigma_i^{r,s}$, που

περιλαμβάνει το hash και το proof του Sortition. Ο λόγος που συμβαίνει ο ξεχωριστός διαμοιρασμός είναι ότι το μήνυμα m_i^r είναι αρκετά μεγάλο διότι περιλαμβάνει όλο το προτεινόμενο block, ενώ το μήνυμα $\sigma_i^{r,s}$ είναι πολύ μικρό και εύκολο να διαδοθεί στο δίκτυο. Με βάση αυτόν τον διαχωρισμό γίνεται και ο επιλεκτικός διαμοιρασμός (selective propagation), ορίζεται μια προτεραιότητα στα μηνύματα που λαμβάνουν και διαδίδουν οι χρήστες κατα το Block Proposal.

Επιλεκτικός Διαμοιρασμός

Από τον αλγόριθμο Cryptographic Sortition έχουμε ένα σύνολο χρηστών που θα προτείνουν κάποιο block σύμφωνα με την διαδικασία block proposal. Επειδή τα μηνύματα αυτά είναι σχετικά μεγάλου μεγέθους, και εν τέλει θα προκύψει ένα block συναλλαγών, ορίζεται μια προτεραιότητα. Χρησιμοποιείται το αποτέλεσμα hash του αλγορίθμου Cryptographic Sortition, και μαζί με τον αριθμό των ψήφων που χρήστη j, ορίζεται η προτεραιότητα ως : $H(hash||j)$. Το μήνυμα με την μεγαλύτερη προτεραιότητα είναι αυτό με την μικρότερη τιμή κατακερματισμού. Δηλαδή αν $H(hash||i) < H(hash||j)$ τότε μεγαλύτερη προτεραιότητα έχει το μήνυμα του χρήστη i.

Παρατήρηση 6.9.1. Παρατηρούμε πως η προτεραιότητα προκύπτει από ποσότητες που υπάρχουν στο διαπιστευτήριο (credential) του χρήστη, ένα μικρό μηνύμα που διαμοιράζεται εύκολα . Άρα η διαδικασία της εύρεσης προτεραιότητας είναι μια γρήγορη διαδικασία για όλους τους χρήστες, και πολύ οικονομική για το δίκτυο.

Άρα ο επιλεκτικός διαμοιρασμός γίνεται ως εξής :

Κάθε χρήστης για χρόνο $\lambda_{stepvar} + \lambda_{priority}$:

- Το πρώτο μήνυμα που λαμβάνει, το επαληθεύει και το διαμοιράζεται.
- Για κάθε επόμενο μήνυμα, υπολογίζει την προτεραιότητα του $H(hash||j)$ και διαμοιράζεται μόνο εάν έχει την μεγαλύτερη προτεραιότητα.
- Εάν δεχτεί δύο διαφορετικά μηνύματα $m_i^{r,1}$ από τον ίδιο χρήστη i, το δεύτερο απορρίπτεται.

Πόσο χρόνο μένουν οι χρήστες στην φάση του block proposal;
Είναι σημαντικό για το πρωτόκολλο να οριστεί ένα χρονικό περιθώριο στο οποίο οι χρήστες θα περιμένουν για τις διάφορες προτάσεις block. Ο χρόνος θα πρέπει να είναι αρκετός ώστε να λάβουν προτάσεις όσο περισσότεροι χρήστες γίνεται, αλλά και προσαρμοσμένος κατάλληλα ώστε να μην επηρεάζεται η απόδοση. Στο χειρότερο σενάριο, ένας χρήστης της επιτροπής του τελευταίου βήματος του BA★ έρχεται σε ομοφωνία πριν από τους υπόλοιπους. Άρα προχωράει στον επόμενο γύρο όπου ξεκινάει το block proposing. Εκεί, υπάρχει περίπτωση να λάβει πολύ γρήγορα μια τιμή μεγαλύτερης προτεραιότητας. Όμως, οι χρήστες που ήταν σε κοινή

επιτροπή μαζί του είναι ένα βήμα πίσω. Άρα για το σενάριο αυτό ορίζεται ο χρόνος παραμονής του στο block proposal ως $\lambda_{stepvar} + \lambda_{priority}$. Το $\lambda_{stepvar}$ είναι η διαφορά του χρόνου στην ολοκλήρωση του τελευταίου βήματος του BA★, ενώ ο χρόνος $\lambda_{priority}$ είναι όσο χρειάζεται στους block proposers να διαμοιραστούν τα διαπιστευτήριά τους (credentials). Πειραματικά έχουν επιλεχεί οι χρόνοι $\lambda_{stepvar} = 5sec$ και $\lambda_{priority} = 5sec$, άρα ο χρόνος παραμονής στο Block Proposal είναι 10seconds.

Εάν υπάρχει αυστηρός συγχρονισμός, είναι πιθανό οι περισσότεροι χρήστες να έχουν λάβει το μήνυμα με την μεγαλύτερη προτεραιότητα και άρα όλοι να συμφωνούν σε ένα κοινό block που θα μπει στο blockchain. Επειδή το δίκτυο είναι διαμοιρασμένο και μερικώς συγχρονισμένο, χωρίς κάποια κεντρική αρχή να ελέγχει την παράδοση των μηνυμάτων ή τον επιλεκτικό διαμοιρασμό, χρειάζεται ένας αλγόριθμος ομοφωνίας ώστε όλοι οι χρήστες να είναι ενημερωμένοι για την τρέχουσα κατάσταση του blockchain. Ο αλγόριθμος αυτός ονομάζεται BA ★.

6.10 BA ★

Το πρωτόκολλο BA ★, πρόκειται για ένα πρωτόκολλο βυζαντινής συμφωνίας με το οποίο οι χρήστες έρχονται σε ομοφωνία για το τελικό block που θα μπει στην αλυσίδα του Algorand σε κάθε γύρο. Πρόκειται για ένα σύνολο τριών αλγορίθμων στο οποίο οι χρήστες ψηφίζουν για το block που έχουν λάβει μέσω της διαδικασίας Block Proposal. Οι τρείς αλγόριθμοι ονομάζονται *Reduction*, *Binary Byzantine Agreement* και *Voting*. Οι δύο βασικές διαδικασίες είναι οι αλγόριθμοι *Reduction* και *Binary Byzantine Agreement*, ενώ το *Voting* είναι ένας βοηθητικός αλγόριθμος που χρησιμοποιείται και στις δύο προαναφερθείσες διαδικασίες.

Για να ξεκινήσει κάποιος χρήστης το πρωτόκολλο αυτό, θα πρέπει να έχει λάβει μήνυμα μεγαλύτερης προτεραιότητας από το Block Proposal. Το BA ★ ξεκινάει με τον αλγόριθμο *Reduction*, ο οποίος βάσει του Block Proposal μαζεύει ψήφους ώστε να υπάρχει ομοφωνία είτε σε ένα κενό block είτε σε ένα μη κενό. Αυτός είναι και ο λόγος που ονομάζεται έτσι, διότι το δίκτυο πλέον περιορίζει την ομοφωνία σε κάποιο από τα δύο block, το κενό και το μη κενό.

Το αποτέλεσμα του *Reduction* είναι η είσοδος για τον αλγόριθμο *Binary Byzantine Agreement* ή αλλιώς *BinaryBA★*. Στον αλγόριθμο αυτό, το δίκτυο έρχεται σε ομοφωνία σε ένα block. Η κατάσταση του συστήματος σχετικά με την ομοφωνία προσδιορίζεται στο τελευταίο μέρος του BA★, όπου ανάλογα με τους ψήφους στο τελευταίο βήμα του επήλθε η ομοφωνία, έχουμε είτε τελική (*Final*) είτε προσωρινή (*Tentative*) ομοφωνία. Στην πέριπτωση της τελικής ομοφωνίας, *Final Consensus*, όλοι οι χρήστες έχουν καταλήξει ασφαλώς σε ένα block που θα προστεθεί στην αλυσίδα. Στην περίπτωση της προσωρινής ομοφωνίας, *Tentative Consensus*, η ασφάλεια δεν μπορεί να εγγυηθεί.

Algorithm 5: BA★(ctx, $round$, $block$)

$hblock \leftarrow Reduction(ctx, round, H(block))$;

$hblock_\star \leftarrow BinaryBA\star\ (ctx, round, hblock))$;

$r \leftarrow CountVotes(round, FINAL, T_{Final}, \tau_{Final}, \lambda_{Step})$;

if $hblock_\star = r$ **then**;

⌊ **return** $< Final, BlockOfHash(\ hblock_\star\ ) >$

else

⌊ **return** $< Tentative, BlockOfHash(\ hblock_\star\ ) >$

Είσοδος

Ο αλγόριθμος BA★ δέχεται σαν είσοδο τις απαραίτητες πληροφορίες για την μετάβαση από το Block Proposal. Το ctx περιλαμβάνει τον σπόρο του τρέχοντος γύρου $seed_r$, τα βάρη των χρηστών w_i, τα συνολικά κρυπτονομίσματα στο σύστημα W_i, και το τελευταίο block της αλυσίδας B^{r-1}. Το $round$ είναι ο τρέχον γύρος r, και το $block$ είναι το block μεγαλύτερης προτεραιότητας που έχει λάβει ο χρήστης από το Block Proposal του γύρου αυτού.

Έξοδος

Ο αλγόριθμος επιστρέφει το τελικό block και την τιμή "$FINAL$", εάν έχει επιτευχθεί ασφαλής ομοφωνία, αλλιώς επιστρέφει την τιμή "$TENTATIVE$" με το αντίστοιχο block.

Περιγραφή βημάτων

Στο πρώτο βήμα εκτελείται ο αλγόριθμος $Reduction$, με τα δεδομένα από το την διαδικασία Block Proposal, και λαμβάνεται η τιμή κατακερματισμού $hblock$. Στο δεύτερο βήμα εκτελείται ο αλγόριθμος $BinaryBA\star$ με είσοδο το $hblock$, για να καταλήξει το δίκτυο σε ομοφωνία ενός block $hblock_\star$. Προκειμένου να επιβεβαιωθεί η ομοφωνία, στο τρίτο βήμα καταμετρούνται οι ψήφοι του τελευταίου βήματος του $BinaryBA\star$, και επιστρέφεται η τιμή του block που έχει ληφθεί ομοφωνία r. Εάν το block αυτό είναι ίδιο με αυτό που προέκυψε από το $BinaryBA\star$, έχουμε τελική ομοφωνία στο πέμπτο βήμα. Εάν $hblock_\star \neq r$, τότε η ομοφωνία είναι προσωρινή και δεν εγγυάται ότι όλοι βλέπουν το ίδιο block $hblock_\star$. Το BlockOfHash(.) είναι μια συνάρτηση που δέχεται σαν όρισμα την τιμή κατακερματισμού κάποιου block και επιστρέφει το ίδιο το block.

Παρατήρηση 6.10.1. Παρατηρούμε πως μέσα στον αλγόριθμο BA★ δεν χρησιμοποιείται όλο το block αλλά το hash του block. Αυτό είναι ένα πολύ λογικό εγχείρημα καθώς τα μηνύματα που περιέχουν ολόκληρα block είναι πολύ μεγάλα και δεν συμφέρει να χρησιμοποιούνται συχνά. Αντ' αυτού λοιπόν αξιοποιείται η τιμή κατακερματισμού, η ομοφωνία γίνεται πάνω σε αυτή την τιμή και στο τέλος με την συνάρτηση BlockOfHash(.) λαμβάνεται το πραγματικό block.

Voting

Σε αυτή την ενότητα θα αναλύσουμε τον αλγόριθμο με τον οποίο γίνεται

η ψηφοφορία στο πρωτόκολλο BA★. Για να πραγματοποιηθεί η ψηφοφορία υπάρχουν τρείς αλγόριθμοι. Ο πρώτος ονομάζεται CommiteeVote και είναι ο βασικός αλγόριθμος με τον οποίο κάθε χρήστης μπορεί να ψηφίσει σε κάποιο βήμα του BA★. Ο δεύτερος αλγόριθμος ονομάζεται CountVotes και είναι ο αλγόριθμος με τον οποίο κάθε χρήστης μπορεί να μετρήσει τος ψήφους που λεχει λάβει. Ο τελευταίος αλγόριθμος είναι ο ProcessMsg, σύμφωνα με τον οποίο κάθε μήνυμα ψήφου που λαμβάνει ένας χρήστης στο BA★ φιλτράρεται.

Η ψηφοφορία είναι μια ιδιωτική διαδικασία, και πραγματοποιείται από μια επιτροπή που εκλέγεται σε κάθε βήμα (*committee*). Κάθε χρήστης έχει δικάιωμα να συμμετάσχει στην ψηφοροφία, και ο σχηματισμός της επιτροπής γίνεται με τον αλγόριθμο Cryptographic Sortition. Οι χρήστες που δεν ανήκουν στο *committee* και άρα δεν ψηφίζουν, παρακολουθούν παθητικά την διαδικασία μαζεύοντας ψήφους από τα μέλη της επιτροπής. Ένα έγκυρο αποτέλεσμα στο πρωτόκολλο BA★, είναι αυτό που έχει αρκετές ψήφους από τα μέλη του *committee*. Ο αριθμός των ψήφων που πρέπει να έχει ένα αποτέλεσμα για να θεωρείται έγκυρο καθορίζεται από μια κατάλληλη σταθερά που ονομάζεται threshold.

Committee Vote

Είναι ο αλγόριθμος της ψηφοφορίας για το πρωτόκολλο βυζαντινής συμ-

φωνίας του Algorand. Προκειμένου να γίνει η ψηφοφορία, επιλέγεται κατάλληλα μια επιτροπή *committee* σε κάθε βήμα. Η επιλογή γίνεται κρυπτογραφικά με το Cryptographic Sortition όπως ακριβώς στο Block Proposal, με δύο διαφορές. Η επιτροπή του BA★ είναι μεγαλύτερη (επιλέγονται περισσότεροι χρήστες) και ο ρόλος (role) είναι διαφορετικός (block proposer στο Block Proposal, committee στο BA★). Εάν κάποιος χρήστης έχει επιλεχθεί, διαμοιράζει ένα μήνυμα ψήφου ενός hash κάποιου block.

Algorithm 6: CommitteeVote(*ctx, round, step, τ, value*)

$role \leftarrow <\text{"}committee\text{"}, round, step>$

$<sorthash, \pi, j> \leftarrow Sortition(user.sk, ctx.seed, \tau, role, ctx.weight[user.pk], ctx.W)$

if $j > 0$ **then**

  $Gossip(<user.pk, Signed_{user.sk}(round, step, sorthash, \pi, H(ctx.lastblock), value)>)$

Είσοδος

Ο αλγόριθμος CommitteeVote δέχεται σαν είσοδο το ctx που περιλαμβάνει τον σπόρο του τρέχοντος γύρου $seed_r$, τα βάρη των χρηστών w_i, τα συνολικά κρυπτονομίσματα στο σύστημα W_i, και το τελευταίο block της αλυσίδας B^{r-1}. Έχει επίσης σαν είσοδο τον γύρο r $(round)$, το βήμα $step$, μια τιμή τ που φράσει το μέγεθος της επιτροπής committee, και την τιμή $value$ που είναι το hash του block που θα ψηφιστεί.

Έξοδος

Ο αλγόριθμος έχει σαν έξοδο ένα μήνυμα ψήφου. Το μήνυμα αυτό περιλαμβάνει τα στοιχεία του χρήστη (δημόσιο κλειδί $user.pk$), την κατάσταση του blockchain όταν συμβαίνει το μήνυμα $(round, step, H(ctx.lastblock))$, την απόδειξη ότι ο χρήστης ανήκει στην επιτροπή για αυτόν τον γύρο στο συγκεκριμένο βήμα $(sorthash, \pi)$ και την ψήφο του χρήστη $(value)$.

Περιγραφή αλγορίθμου

Ο αλγόριθμος στο πρώτο βήμα του ορίζει τον ρόλο του χρήστη που πρόκειται να ψηφίσει. Δηλαδή ότι θα ανήκει σε επιτροπή του BA★ ("$committee$"), για τον γύρο $round$ στο βήμα $step$. Είναι σημαντικό να ορίζεται αυτό το βήμα καθώς το πρωτόκολλο αποτελείται από πολλές ψηφοφορίες και επιτροπές, και θα πρέπει τα μηνύματα να μην συγχέονται μεταξύ τους.
Στο δεύτερο βήμα, εκτελείται ο αλγόριθμος Sortition για τον ρόλο που προσδιορίστηκε στο πρώτο βήμα, ώστε να σχηματιστεί η επιτροπή $committee$.
Ο κάθε χρήστης ιδιωτικά με τα στοιχεία του, ελέγχει εάν ανήκει στην επιτροπή και αποθηκεύει το αποτέλεσμα που είναι η έξοδος των Verifiable Random Functions $(sorthash)$, η απόδειξη του αποτελέσματος (π), και οι συμμετοχές του στην επιτροπή.
Στο τρίτο βήμα, εάν ο χρήστης έχει επιλεχθεί έστω και μια φορά ως μέλος της επιτροπής $(j > 0)$, τότε στέλνει ένα μήνυμα ψήφου, με τα στοιχεία του και την ψήφο του υπογεγραμμένα. Τα στοιχεία που χρειάζεται να συμπεριλάβει στο μήνυμα είναι: το δημόσιο κλειδί του ώστε οποιοσδήποτε να μπορεί να επιβεβαιώσει την υπογραφή, την κατάσταση του blockchain, και την ψήφο του, που είναι το hash ενός block. Είναι σημαντικό πέρα από την ψήφο και τα στοιχεία του να υπάρχει και η κατάσταση του blockchain, διότι έτσι θα αποφευχθεί η δημιουργία διχάλας στην αλυσίδα, με το να σταλθεί κάποιο μήνυμα με λάθος γύρο ή λάθος hash τελευταίου επιβεβαιωμενου block $(H(B^{r-1}))$.

CountVotes

Με τον αλγόριθμο CountVotes, οι χρήστες μπορούν να αποθηκεύουν τις ψήφους που λαμβάνουν και να έρχονται σε ομοφωνία. Η διαδικασία περιλαμβάνει τον έλεγχο των μηνυμάτων και την προσμέτρησή τους εάν είναι έγκυρα και έγκαιρα. Το πρώτο block hash που θα λάβει αρκετές

ψήφους θα είναι και αυτό που επιστρέφει ο αλγόριθμος.

Algorithm 7: CountVotes$(ctx, round, step, T, \tau, \lambda)$

```
start ← Time()
counts ← {}
voters ← {}
msgs ← incomingMsgs[round, step].iterator()
while TRUE do
    m ← msgs.next()
    if m = ⊥ then
        if Time() > start + λ then
            return TIMEOUT
    else
        < votes, value, sorthash > ← ProcessMsg(ctx, τ, m)
        if pk ∈ voters or votes = 0 then
            continue;
        voters∪ = {pk}
        counts[value]+ = votes
        if counts[value] > T · τ then
            returnvalue
```

Είσοδος

Ο αλγόριθμος CountVotes δέχεται σαν είσοδο τις πληροφορίες του χρήστη και της αλυσίδας (ctx), τον γύρο και το βήμα του πρωτοκόλλου $(round, step)$, και τρείς παραμέτρους T, τ και λ.

Έξοδος

Η έξοδος του αλγορίθμου είναι η τιμή κατακερματισμού του block που έχει λάβει τις περισσότερες ψήφους. Οι ψήφοι που λαμβάνει κάθε χρήστης αποθηκεύονται σε ένα hash table, και εάν κάποια τιμή αυτού του πίνακα έχει λάβει αρκετές ψήφους επιστρέφεται. Υπάρχει και η περίπτωση ο αλγόριθμος να επιστρέψει $TIMEOUT$ εάν ο χρήστης δεν έχει λάβει ψήφους στο χρονικό διάστημα λ, ή να τερματιστεί εάν κάποιο από τα μηνύματα δεν είναι έγκυρο.

Περιγραφή αλγορίθμου

Το πρώτο βήμα του αλγορίθμου είναι η αρχικοποίηση του χρόνου στην μεταβλητή start. Είναι σημαντικό τα μηνύματα ψήφων να λαμβάνονται σε ένα περιορισμένο χρονικό διάστημα λ, που είναι αρκετό ώστε όλοι οι χρήστες να έχουν παραλάβει μηνύματα ψήφων ακόμη και αν είναι πίσω έναν ολόκληρο γύρο.
Το δεύτερο βήμα είναι η αρχικοποίηση ενός πίνακα τιμών κατακερμα-

τισμού ($counts$), στον οποίο θα αποθηκεύεται ο αριθμός των ψήφων κάθε κανούριου block hash που δέχεται ο χρήστης. Στόχος είναι να συγκεντρώνονται εκεί οι ψήφοι για το κάθε block και στο τέλος να μπορεί να εξαχθεί η τιμή που έχει αρκετές ψήφους.
Το τρίτο βήμα είναι η αρχικοποίηση ενός διανύσματος το οποίο ονομάζεται $voters$ και είναι κάτι σαν λίστα των δημοσίων κλειδιών που έχουν ψηφίσει και ο χρήστης έχει λάβει μήνυμά τους. Ουσιαστικά εάν κάποιος έχει στείλει τις ψήφους του και έχει καταγραφεί στην λίστα $voters$, τότε κάθε επόμενο μήνυμά του θα απορρίπτεται.
Η τελευταία αρχικοποίηση, στο τέταρτο βήμα, σχετίζεται με τα εισερχόμενα μηνύματα. Τα μηνύματα που στέλνονται αποθηκεύονται προσωρινά με βάση τον γύρο και το βήμα που αντιστοιχίζονται. Η αρχικοποίηση $msgs$ είναι η λίστα μηνυμάτων που έχει λάβει ο χρήστης και σχετίζονται με τον τρέχον γύρο και βήμα του πρωτοκόλλου.
Στην συνέχεια ο αλγόριθμος επεξεργάζεται τα μηνύματα που λαμβάνονται και καταχωρεί τον αριθμό των ψήφων. Για να γίνει αυτό πρώτα διαβάζει ένα ένα τα μηνύματα m από την λίστα $msgs$. Εάν δεν έχουν ληφθεί αρκετά μηνύματα στο χρονικό διάστημα λ, ο αλγόριθμος επιστρέφει $TIMEOUT$. Αλλιώς, ελέγχεται η εγκυρότητα και συλλέγονται οι ψήφοι και οι πληροφορίες του ψηφοφόρου από τον αλγόριθμο ProcessMsg(). Εάν ο χρήστης έχει ξαναψηφίσει για αυτόν τον γύρο στο συγκεκριμένο βήμα, ($pk \in voters$) ή το μήνυμα δεν έχει ψήφους, τότε το συγκεκριμένο μήνυμα παραλείπεται. Σε κάθε άλλη περίπτωση, ο ψηφοφόρος προστίθεται στην λίστα με τους $voters$ και οι ψήφοι συνυπολογίζονται. Εάν για κάποια τιμή ενός μηνύματος έχουν μαζευτεί αρκετοί ψήφοι, ο αλγόριθμος σταματά και επιστρέφεται η τιμή αυτή. Αρκετοί ψήφοι θεωρούνται αυτοί που ξεπερνάνε τον αριθμό $T \cdot \tau$.

Πώς επιλέγεται το όριο των ψήφων;

Το κάτω όριο των ψήφων με το οποίο μπορεί να επιτευχθεί ομοφωνία, ορίζεται ως το γινόμενο $T \cdot \tau$. Το γινόμενο αυτό εξαρτάται από το ποσοστό των ψήφων που χρειάζεται το πρωτόκολλο για να έρθουν σε ομοφωνία οι χρήστες και από το μέγεθος της επιτροπής που ψηφίζει. Πιο συγκεκριμένα το ποσοστό των ψήφων που χρειάζεται το σύστημα θα πρέπει να είναι σχετικά μικρό ώστε να έχουμε γρήγορη ομοφωνία, και να είναι κοντά στην τιμή h που είναι το ποσοστό των ειλικρινών χρηστών στο σύστημα. Η δεύτερη παράμετρος είναι το αναμενόμενο μέγεθος της επιτροπής του Sortition, και όσο μικρότερο είναι το ποσοστό των ψήφων που χρειάζονται οι χρήστες για να συμφωνήσουν σε κάποια τιμή, τόσο μεγαλύτερο θα πρέπει να είναι το μέγεθος της επιτροπής. Το κάτω όριο έτσι όπως είναι φτιαγμένο διασφαλίζει ότι όλοι οι ειλικρινείς χρήστες που θα τρέξουν τον αλγόριθμο CountVotes() θα επιστρέψουν την ίδια τιμή block.

Θεώρημα 6.10.1. *Η πιθανότητα οι παρακάτω συνθήκες να παραβιάζονται είναι αμελητέα :*

1. *Ο αριθμός των ειλικρινών μελών της επιτροπής είναι* $> T_{step} \cdot \tau_{step}$.
2. *Το ήμισι των ειλικρινών μελών της επιτροπής μαζί με τον αριθμό των κακόβουλων μελών είναι* $\leq T_{step} \cdot \tau_{step}$.

Απόδειξη

1. Θα δείξουμε πότε παραβιάζεται η συνθήκη $\#good > T_{step} \cdot \tau_{step}$. Για να παραβιαστεί θα πρέπει $\#good \leq T_{step} \cdot \tau_{step}$. Τα ειλικρινή μέλη της επιτροπής με αναμενόμενο αριθμό μελών τ_{step} είναι : $h \cdot \tau_{step}$. Η πιθανότητα να έχουμε K ειλικρινή μέλη στην επιτροπή είναι :

$$Pr[\#good = K] = \frac{h\tau_{step}{}^{K}}{K!} e^{-h\tau_{step}}$$

(από την σχέση 3.7.1)
Άρα η πιθανότητα να παραβιάζεται η συνθήκη είναι η πιθανότητα οι ειλικρινείς χρήστες να είναι το πολύ μέχρι $T_{step} \cdot \tau_{step}$. Δηλαδή :

$$Pr[\#good \leq T_{step} \cdot \tau_{step}] = \sum_{K=0}^{T_{step} \cdot \tau_{step}} \frac{h\tau_{step}{}^{K}}{K!} e^{-h\tau_{step}}$$

2. Θα δείξουμε πότε παραβιάζεται η συνθήκη $\#\frac{1}{2}good + \#bad \leq T_{step} \cdot \tau_{step}$. Αφού τα ειλικρινή μέλη της επιτροπής με αναμενόμενο αριθμό μελών τ_{step} είναι : $h \cdot \tau_{step}$, τα κακόβουλα μέλη της επιτροπής θα είναι : $(1-h) \cdot \tau_{step}$. Η πιθανότητα να έχουμε L μη έμπιστα μέλη στην επιτροπή είναι :

$$Pr[\#bad = L] = \frac{(1-h)\tau_{step}{}^{L}}{L!} e^{-(1-h)\tau_{step}}$$

(από την σχέση 3.7.1) Άρα η πιθανότητα να υπάρχουν K έμπιστα μέλη και L κακόβουλα είναι:

$$\frac{h\tau_{step}{}^{K}}{K!} e^{-h\tau_{step}} \frac{(1-h)\tau_{step}{}^{L}}{L!} e^{-(1-h)\tau_{step}} =$$

$$\frac{h\tau_{step}{}^{K}}{K!} \frac{(1-h)\tau_{step}{}^{L}}{L!} e^{h\tau_{step}-(1-h)\tau_{step}} =$$

$$\frac{h\tau_{step}{}^{K}}{K!} \frac{(1-h)\tau_{step}{}^{L}}{L!} e^{-\tau_{step}}$$

Η σχέση $\#\frac{1}{2}good + \#bad \leq T_{step} \cdot \tau_{step}$ παραβιάζεται όταν παραβιάζεται και η $\#good + 2\#bad \leq 2T_{step} \cdot \tau_{step}$. Δηλαδή όταν ισχύει : $\#good + 2\#bad > 2T_{step} \cdot \tau_{step} \Rightarrow K + 2L > 2T_{step} \cdot \tau_{step}$.

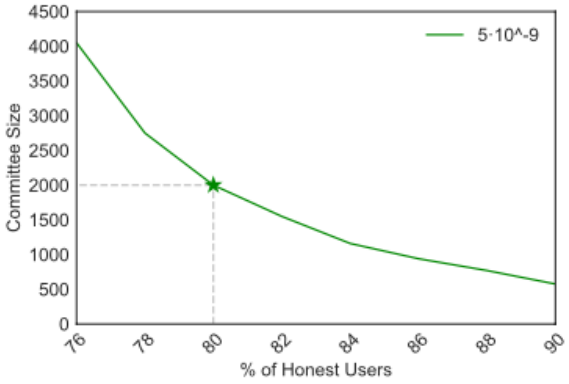
Άρα η πιθανότητα να παραβιάζεται η συνθήκη $\#\frac{1}{2}good + \#bad \leq T_{step} \cdot \tau_{step}$ είναι :

$$\sum_{K+2L > 2T_{step} \cdot \tau_{step}}^{\infty} \frac{h\tau_{step}{}^{K}}{K!} \frac{(1-h)\tau_{step}{}^{L}}{L!} e^{-\tau_{step}}$$

Η σειρά συγκλίνει και άρα μπορεί να προσεγγιστεί η τιμή της με κατάλληλες τιμές στις παραμέτρους h, τ_{step}, T_{step}.

Οι τιμές των παραμέτρων επιλέγονται έτσι ώστε η παραβίαση των παραπάνω συνθηκών να γίνεται με πολύ μικρή πιθανότητα $F = 10^{-12}$ ή $F = 10^{-18}$. □

Παρακάτω βλέπουμε ένα διάγραμμα του αναμενόμενου αριθμού της επιτροπής και του ποσοστού των ειλικρινών χρηστών στη σύστημα (τ_{step} και h). Σε αυτό το διάγραμμα φαίνεται το αναμενόμενο μέγεθος της επιτροπής τ που θα πρέπει να έχουμε ώστε οι παραπάνω παραβιάσεις να γίνονται με πολύ μικρή πιθανότητα F.



Σχήμα 6.7: Διάγραμμα τ με h

Στο διάγραμμα φαίνεται πως όσο αυξάνεται το ποσοστό των ειλικρινών χρηστών, ο αριθμός των μελών της επιτροπής τ μειώνεται. Μάλιστα όσο η τιμή h ξεπερνάει την τιμή $\frac{2}{3}$, τόσο πιο μεγάλη είναι η μείωση του τ. Μια επιτροπή μεγέθους 2000 χρηστών αντιστοιχίζεται σε ένα καλό ποσοστό ειλικρίνών χρηστών 80%, ώστε η πιθανότητα να παραβιάζονται οι συνθήκες να είναι πολύ μικρή $F = 5x10^{-9}$. (η παράμετρος T_{step}=0,685)

Λήμμα 6.10.1. Έστω ότι καλείται ο αλγόριθμος CountVotes() από δύο ειλικρινείς χρήστες του ίδιου γύρου και βήματος, και επιστρέφει την τιμή $u_1 \neq TIMEOUT$ και $u_2 \neq TIMEOUT$ στον καθένα αντίστοιχα. Τότε η πιθανότητα να ισχύει : $u_1 \neq u_2$ είναι αμελητέα.

Απόδειξη
Για να έχει επιστρέψει την τιμή u_1 ένας ειλικρινής χρήστης, έχει λάβει περισσότερες από $T_{step} \cdot \tau_{step}$ ψήφους. Αποδείχθηκε στο προηγούμενο θεώρημα πως $\#\frac{1}{2}good+\#bad \leq T_{step} \cdot \tau_{step}$ και $\#good > T_{step} \cdot \tau_{step}$, άρα ο χρήστης έχει ψήφους από ποσοστό μεγαλύτερο των 50% των ειλικρινών ψηφοφόρων. Κάθε ειλικρινής χρήστης όμως στέλνει ψήφο για μια μόνο τιμή. Άρα αφού περισσότεροι από τους μισούς ειλικρινείς έχουν στείλει

την τιμή u_1, δεν μπορεί να υπάρξει άλλη τιμή u_2 στην οποία έχει καταλήξει ο δεύτερος ειλικρινής χρήστης. Άρα $u_1 = u_2$.

□

ProcessMsg

Είναι ένας αλγόριθμος με τον οποίο κάθε μήνυμα που λαμβάνει ένας χρήστης στην λίστα μηνυμάτων του, επεξεργάζεται και ελέγχεται η εγκυρότητα του. Συγκεκριμένα χρησιμοποιείται στον αλγόριθμο CountVotes(), όπου μαζεύονται οι ψήφοι-μηνύματα σε ένα buffer και ένα ένα ελέγχεται και προσμετράται στις ψήφους ή απορρίπτεται. Η διαδικασία ελέγχου περιλαμβάνει τον έλεγχο των υπογραφών, του hash του τελευταίου block της αλυσίδας, και των στοιχείων του ψηφοφόρου.

Algorithm 8: ProcessMsg($ctx,\ \tau,\ m$)

$< pk, signed\_m > \leftarrow\ m$

if $VerifySignature(pk, signed\_m) \neq OK$ **then**
  return $< 0, \perp, \perp >$

$< round, step, sorthash, \pi, hprev, value > \leftarrow signed\_m$

if $hprev \neq H(ctx.last\_block)$ **then**
  return $< 0, \perp, \perp >$;

$votes \leftarrow VerifySort(pk, sorthash, \pi, ctx.seed, \tau,$
$< \text{"committee"}, round, step >, ctx.weight[pk], ctx.W)$

return $< votes, value, sorthash >$

Είσοδος

Ο αλγόριθμος επεξεργασίας μηνυμάτων έχει σαν είσοδο τις πληροφορίες του χρήστη και της κατάστασης της αλυσίδας (ctx), το μέγεθος της επιτροπής τ του συγκεκριμένου βήματος, και το μήνυμα m που πρόκειται να επεξεργαστεί.

Έξοδος

Ο αλγόριθμος ProcessMsg() αφού επεξεργαστεί το μήνυμα, εάν είναι έγκυρες οι πληροφορίες του χρήστη και συμβατές με την κατάσταση που βρίσκεται η αλυσίδα, επιστρέφει τον αριθμό των ψήφων που περιλαμβάνει ($votes$), την τιμή του block που αφορούν οι ψήφοι ($value$), και την ψευδοτυχαία τιμή των VRFs με την οποία επιλέχθηκε στην επιτροπή ($sorthash$). Εάν δεν είναι έγκυρη η υπογραφή του μηνύματος ή το μήνυμα αυτό αυξάνει άλλη αλυσίδα (άλλος δείκτης προηγούμενου block), τότε επιστρέφεται η τιμή $< 0, \perp, \perp >$. Τέλος, υπάρχει περίπτωση να επιστρέψει και την τιμή 0, εάν ο αλγόριθμος επαλήθευσης των VRF δεν μπορεί

να αντιστοιχίσει στον χρήστη την ψευδοτυχαιότητα $sorthash$ (πχ λάθος απόδειξη π, ή λάθος $seed$).

Περιγραφή αλγορίθμου

Ο αλγόριθμος ProcessMsg() αρχικά λαμβάνει το δημόσιο κλειδί και την υπογραφή του χρήστη $pk, signed\_m$ από το μήνυμα m. Εάν η υπογραφή δεν είναι έγκυρη, δηλαδή δεν αντιστοιχίζεται στον χρήστη με δημόσιο κλειδί pk, τότε ο αλγόριθμος τερματίζεται και επιστρέφεται η τιμή $< 0, \perp, \perp >$.
Εάν η υπογραφή του μηνύματος είναι έγκυρη, τότε λαμβάνεται από την υπογραφή το περιεχόμενο του μηνύματος. Δηλαδή ο γύρος, το βήμα του πρωτοκόλλου, τα δεδομένα του χρήστη από το Sortition, η τιμή του τελευταίου block που έχει προστεθεί στην αλυσίδα, και η ψήφος.
Στην συνέχεια, είναι πολύ σημαντικό να γίνει έλεγχος για τιμή του τελευταίου block $hprev$. Εάν αυτή η τιμή δεν είναι το τελευταιο block που έχει προστεθεί στο blockchain τότε επίσης τερματίζεται ο αλγόριθμος με επιστροφή $< 0, \perp, \perp >$. Αυτό συμβαίνει διότι το blockchain είναι μια αλυσίδα από blocks, που για να είναι ασφαλής θα πρέπει να αυξάνεται γραμμικά χωρίς διακλαδώσεις. Εάν το καινούριο block συνδεθεί με λάθος προηγούμενο block, τότε θα δημιουργηθεί fork.
Εάν λοιπόν όλα αυτά πάνε καλά ελέγχονται και τα στοιχεία του χρήστη από τον αλγόριθμο VerifySort(). Πιο συγκεκριμένα ελέγχεται εάν όντως ο χρήστης έχει επιλεχθεί στην επιτροπή του συγκεκριμένου βήματος για τον γύρο αυτό, και πόσες φορές έχει το δικαίωμα να ψηφίσει. Από αυτόν τον έλεγχο προκύπτει ο αριθμός των ψήφων $votes$.
Τέλος, επιστρέφεται ο αριθμός των ψήφων, η τιμή του block για την οποία ψηφίζει ο χρήστης, και η ψευδοτυχαιότητα που αντιπροσωπεύει το δικαίωμα ψήφου του. $(votes, value.sorthash)$

Reduction

Ο αλγόριθμος Reduction() είναι το πρώτο σκέλος του πρωτοκόλλου Βυζαντινής Συμφωνίας του Algorand (BA★). Σκοπός του αλγορίθμου αυτού είναι να έρθουν οι χρήστες σε ομοφωνία από το στάδιο του Block Proposal. Όπως είχε αναφερθεί στο σχετικό κεφάλαιο, ο χρήστης που έχει την μεγαλύτερη προτεραιότητα, έχει και το block που θα λάβουν οι περισσότεροι χρήστες. Λόγω της αρχιτεκτονικής του δικτύου επικοινωνίας ή κακόβουλων χρηστών, υπάρχει πιθανότητα να μην έχουν λάβει όλοι οι χρήστες το block μεγαλύτερης προτεραιότητας. Στον αλγόριθμο Reduction() γίνεται ομοφωνία είτε στο block αυτό είτε σε ένα κενό block. Η ομοφωνία επιτυγχάνεται με συνεχόμενες ψηφοφορίες κατάλληλα επιλεγμένης επιτροπής (χρησιμοποιώντας τους αλγόριθμους του Voting)

Algorithm 9: Reduction(*ctx, round, hblock*)

$CommitteeVote(ctx, round, REDUCTION_{one}, \tau_{step}, hblock)$

$hblock_1 \leftarrow$
$CountVotes(ctx, round, REDUCTION_{one}, T_{step}, \tau_{step}, \lambda_{block} + \lambda_{step})$

$empty\_hash \leftarrow H(Empty(round, H(ctx.last\_block)))$

if $hblock_1 = TIMEOUT$ **then**
  $CommitteeVote(ctx, round, REDUCTION_{two}, \tau_{step}, empty\_hash)$
else
  $CommitteeVote(ctx, round, REDUCTION_{two}, \tau_{step}, hblock_1)$

$hblock_2 \leftarrow$
$CountVotes(ctx, round, REDUCTION_{two}, T_{step}, \tau_{step}, \lambda_{step})$

if $hblock_2 = TIMEOUT$ **then**
  return $empty\_hash$;
else
  return $hblock_2$;

Είσοδος
Ο αλγόριθμος δέχεται σαν είσοδο τις πληροφορίες του χρήστη και της αλυσίδας (*ctx*), τον αριθμό του γύρου *round* και το block που έχει λάβει ο κάθε χρήστης μετά την διαδικασία Block Proposal.

Έξοδος
Ο Reduction() έχει δύο πιθανές εξόδους. Η πρώτη πιθανή έξοδος είναι ένα hash ενός κενού block ($empty\_hash$), και η δεύτερη πιθανή έξοδος είναι ένα hash ενός μη κενού block που έχει προελθει από το Block Proposal ($hblock_2$). τα δύο πιθανά block έχουν προέλθει από ψηφοφορίες μιας επιτροπής χρηστών και άρα έχουν προκύψει ομόφωνα.

Περιγραφή
Ο αλγόριθμος αποτελείται από συνεχόμενες ψηφοφορίες μιας επιτροπής που δημιουργείται από τον αλγόριθμο Sortition. Η διαδικασία που ακολουθείται για την ψηφοφορία έχει περιγραφτεί στην ανάλυση του αλγορίθμου CommitteeVote().
Στο πρώτο βήμα του αλγορίθμου πραγματοποιείται η πρώτη ψηφοφορία, και το βήμα αυτό ονομάζεται $REDUCTION_{one}$. Η ψηφοφορία αυτή γίνεται πάνω στο block που έχουν λάβει τα μέλη της επιτροπής από το Block Proposal ($hblock$). Η ψηφοφορία γίνεται καλώντας τον αλγόριθμο CommitteeVote().
Στο δεύτερο βήμα του αλγορίθμου προσμετρούνται οι ψήφοι του $REDUCTION_{one}$. Αυτό πραγματοποιείται με τον αλγόριθμο CountVotes(), και επιστρέφεται το block που έχει λάβει τις περισσότερες ψήφους ($hblock_1$). Υπενθυμίζεται ότι ο αριθμός των ψήφων που απαιτούνται για την ομοφωνία είναι

κατάλληλα επιλεγμένος αναλογικά με τους ειλικρινείς χρήστες του δικτύου. Το σημαντικότερο κομμάτι σε αυτό το σημείο του αλγορίθμου είναι ο χρόνος λ που δίνεται στο σύστημα μέχρι να ληφθούν όλες οι ψήφοι. Ορίζεται ως $\lambda_{block} + \lambda_{step}$, και θα γίνει στην συνέχεια ανάλυση της επιλογής των παραμέτρων αυτών.
Στην συνέχεια ορίζεται ένα κενό block με την εντολή $Empty()$. Σε αυτό το block προσδιορίζεται ο γύρος που έχει δημιουργηθεί και το hash του τελευταίου block που προστέθηκε στην αλυσίδα. Παρ'όλο που το block δεν περιέχει συναλλαγές, επειδή ο αλγόριθμος επεξεργάζεται τιμές σύνοψης και όχι ολόκληρα block, παίρνουμε την hash τιμή του εφαρμόζοντας την συνάρτηση H(). Η τελική τιμή ονομάζεται $empty\_hash$.
Στο τέταρτο βήμα του αλγορίθμου, ελέγχεται το αποτέλεσμα της ψηφοφορίας $REDUCTION_{one}$. Εάν η τιμή που προέκυψε από τον αλγόριθμο CountVotes() δεν έλαβε αρκετές ψήφους στο χρονικό διάστημα $\lambda_{block} + \lambda_{step}$ και επεστράφη η τιμή $TIMEOUT$, τότε η δεύτερη ψηφοφορία γίνεται με το κενό block hash $empty\_hash$. Ουσιαστικά εάν έχει επιστραφεί $TIMEOUT$, τότε οι χρήστες της επιτροπής του $REDUCTION_{one}$ δεν έχουν συμπληρώσει πάνω από $T \cdot \tau$ ψήφους για κάποια τιμή. Αφού δεν έχει επέλθει ομοφωνία, για λόγους ασφάλειας το Algorand προτείνει την ομοφωνία σε ένα κενό block. Βέβαια εάν η τιμή που προέκυψε δεν είναι το $TIMEOUT$ αλλά κάποιο block, με αυτό το block $hblock_1$ ξεκινάει η δεύτερη ψηφοφορία $REDUCTION_{two}$.
Στην δεύτερη ψηφοφορία που ονομάζεται $REDUCTION_{two}$, ακολουθείται ακριβώς η ίδια διαδικασία. Εάν έχουν μαζευτεί αρκετές ψήφοι για κάποια τιμή $hblock_1$ επιστρέφεται αυτή, εάν όμως ο αλγόριθμος έχει αρχικοποιηθεί με το $empty\_hash$ ή επιστραφεί $TIMEOUT$, η τελική τιμή που δίνει ο αλγόριθμος Reduction() θα είναι ένα hash κενού block.

Πώς επιλέγονται οι παράμετροι χρόνου λ;

Οι παράμετροι επιλέγονται με τέτοιο τρόπο ώστε σε ένα ασύγχρονο δίκτυο, όπως αυτό του Algorand που μπορεί να υπάρχουν καθυστερήσεις, η επικοινωνία να μπορέσει να εδραιωθεί. Επειδή ορισμένοι χρήστες μπορεί να έχουν λάβει γρήγορα το block μεγαλύτερης προτεραιότητας από το προηγούμενο βήμα, έχουν ξεκινήσει το Reduction() και περιμένουν ψήφους από την υπόλοιπη επιτροπή. Κάποιοι χρήστες όμως μπορεί να περιμένουν ακόμη να λάβουν το block από το Block Proposal, καθώς είναι ένα μήνυμα μεγάλου μεγέθους. Για αυτόν τον λόγο ο χρόνος αναμονής στο βήμα $REDUCTION_{one}$ ορίζεται ως τον χρόνο που περιμένει κάποιος να λάβει ένα block, μαζί με τον χρόνο που περιμένει κάποιος να λάβει ψήφους σε κάποιο βήμα. Δηλαδή $\lambda_{block} + \lambda_{step}$. Η παράμετρος λ_{block} εκτιμάται περίπου 1 λεπτό, ενώ η παράμετρος $\lambda_{step} = 20sec$.

BinaryBA★

Ο τελευταίος αλγόριθμος του BA★ είναι ο Binary Byzantine Agreement. Ο αλγόριθμος αυτός ονομάζεται έτσι διότι οι χρήστες έρχονται σε ομοφωνία ανάμεσα σε δύο τιμές, είτε στην τιμή του block που προήλθε

από το Reduction(), είτε σε ένα κενό block. Ουσιαστικά, ο αλγόριθμος καταλήγει σε ένα από τα δύο προτεινόμενα block του Reduction(), το οποίο θα μπεί στο blockchain του Algorand. Ο BinaryBA★() είναι ένας αλγόριθμος από συνεχόμενες ψηφοφορίες μέχρι να επιτευχθεί η ομοφωνία. Κάθε βήμα είναι και μια ψηφοφορία, και για λόγους ασφάλειας και απόδοσης τα βήματα έχουν ένα άνω όριο ($MaxSteps$). Ο αλγόριθμος είναι κατάλληλα φτιαγμένος ώστε να αντιμετωπίζει τις χειρότερες περιπτώσεις συγχρονισμού του δικτύου (όπως έχει αναφερθεί το δίκτυο είναι μερικώς συγχρονισμένο με ασύγχρονες περιόδους b και αυστηρά συγχρονισμένες περιόδους s). Αποτελείται από 3 βήματα ψηφοφορίας, τα οποία επαναλαμβάνονται μέχρι την ομοφωνία ή τα $MaxSteps$.

Είσοδος

Ο αλγόριθμος δέχεται σαν είσοδο τις πληροφορίες του χρήστη και της κατάστασης του blockchain (ctx), τον γύρο ($round$) και το block του Reduction().

Έξοδος

Ο αλγόριθμος έχει δύο πιθανές εξόδους. Είτε ένα block hash που θα είναι το κενό block ή το block του Reduction(), είτε την έξοδο "$TIMEOUT$". Επίσης υπάρχει και η περίπτωση να επιστραφεί η τιμή 0 εάν κάποιο από τα στοιχεία του χρήστη δεν επιβεβαιώνεται από τους αλγορίθμους CountVotes() ή ProcessMsg.

Algorithm 10: BinaryBA★(*ctx, round, block_hash*)

```
step ← 1
r ← block_hash
empty_hash ← H(Empty(round, H(ctx.last_block)))
while step < MaxSteps do
    CommitteeVote(ctx, round, step, τ_step, r)
    r ← CountVotes(ctx, round, step, T_step, τ_step, λ_step)
    if r = TIMEOUT then
        r ← block_hash
    else if r ≠ empty_hash then
        for step < s' ≤ step + 3 do
            CommitteeVote(ctx, round, s', τ_step, r)
        if step = 1 then
            CommitteeVote(ctx, round, FINAL, τ_final, r)
        return r
    step++

    CommitteeVote(ctx, round, step, τ_step, r)
    r ← CountVotes(ctx, round, step, T_step, τ_step, λ_step)
    if r = TIMEOUT then
        r ← empty_hash
    else if r = empty_hash then
        for step < s' ≤ step + 3 do
            CommitteeVote(ctx, round, s', τ_step, r)
        return r
    step++

    CommitteeVote(ctx, round, step, τ_step, r)
    r ← CountVotes(ctx, round, step, T_step, τ_step, λ_step)
    if r = TIMEOUT then
        if CommonCoin(ctx, round, step, τ_step) = 0 then
            r ← block_hash
        else
            r ← empty_hash
    step++
HangForever()
```

Περιγραφή Αλγορίθμου

Στα πρώτα βήματα του αλγορίθμου αρχικοποιούνται οι μεταβλητές που θα χρησιμοποιηθούν. Η πρώτη αρχικοποίηση είναι αυτή των βημάτων (*step*), ώστε να μην ξεπεραστούν τα *MaxSteps* του αλγορίθμου, και τους δίνεται η τιμή 1. Η δεύτερη αρχικοποίηση είναι αυτή του μη κενού block, στο οποίο θα έρθει σε ομοφωνία το δίκτυο του Algorand με το

BA★. Η μεταβλητή αυτή ονομάζεται r και δέχεται την τιμή $block\_hash$ του Reduction(). Τελευταία αρχικοποίηση είναι αυτή του κενού block ($empty\_hash$), στο οποίο προσδιορίζεται ο γύρος και το τελευταίο κουτί συναλλαγών του Algorand.
Το επόμενο κομμάτι στον αλγόριθμο BA★ είναι να γίνουν οι κατάλληλες ψηφοφορίες ώστε να έρθει το δίκτυο σε ομοφωνία σε βήματα λιγότερα από $MaxSteps$. Για τον λόγο αυτό όλες οι ψηφοφορίες υπάρχουν μέσα σε ένα while loop, το οποίο επαναλαμβάνεται για όσο η μεταβητή $step$ είναι μικρότερη του μέγιστου αριθμού βημάτων. Μέσα στο loop υπάρχουν τρείς φάσεις ψηφοφοριών, που έχουν δημιουργηθεί ώστε να συμπεριλάβουν τις περιπτώσεις συγχρονισμού του δικτύου. Σε κάθε βήμα εκλέγεται και μια νέα επιτροπή ψηφοφορίας.
Η πρώτη φάση ψηφοφορίας αντικατοπτρίζει την περίπτωση αυστηρού συγχρονισμού. Έχει φτιαχτεί έτσι ώστε ακόμη και κάποιοι από τους ειλικρινείς χρήστες να επιστρέφουν $TIMEOUT$ επειδή δεν έλαβαν αρκετές ψήφους, να μπορεί να φτάσει το δίκτυο σε ομοφωνία. Αναλυτικότερα, σε αυτή την φάση εκλέγεται πρώτα μια επιτροπή και στην συνέχεια ψηφίζεται ένα block r σύμφωνα με τον CommitteeVote(). Έπειτα, προσμετρώνται οι ψήφοι. Στην περίπτωση αυστηρού συγχρονισμού, το 95% των μηνυμάτων των ειλικρινών χρηστών θα παραδωθεί κανονικά, και επειδή ο αριθμός των ειλικρινών χρηστών στην επιτροπή είναι πάντα μεγαλύτερος από τον αριθμό των κακόβουλων χρηστών, οι περισσότεροι χρήστες θα λάβουν πάνω από $T \cdot \tau$ ψήφους.

- Περίπτωση 1.1 : Εάν από την καταμέτρηση με τον αλγόριθμο CountVotes() επιστραφεί $TIMEOUT$ σε κάποιο χρήστη, αυτός εμπιστεύεται το υπόλοιπο δίκτυο και υιοθετεί την τιμή $block\_hash$. Έπειτα προχωρά στο επόμενο βήμα του BinaryBA★, όπου θα ψηφίσει για την τιμή $r = block\_hash$.

- Περίπτωση 1.2 : Εάν από τον CountVotes() προκύψει κενό block, ο αλγόριθμος προχωρά στο επόμενο βήμα όπου ξαναγίνεται ψηφοφορία. Οι χρήστες στο επόμενο $step$ αρχικοποιούν το CommitteeVote() με αυτή την τιμή $empty\_hash$. Εάν λοιπόν το block είναι κενό το σύστημα δεν μπορεί να φτάσει σε "FINAL" consensus.

- Περίπτωση 1.3 : Οι υπόλοιποι χρήστες που έλαβαν μη κενό block, δημιουργούν 3 επιτροπές και ψηφίζουν για την τιμή που έλαβαν, και εάν είναι το πρώτο βήμα κάνουν την τελική ψηφοφορία και επιστρέφουν την τιμή r. Παρατηρείται πως οι χρήστες παρ' όλο που έχουν έρθει σε ομοφωνία, παραμένουν για άλλες 3 ψηφοφορίες στο σύστημα. Αυτό συμβαίνει διότι με αυτόν τον τρόπο θα υπάρχουν αρκετές ψήφοι στο δίκτυο για το block r, ώστε ακόμη και ένα ποσοστό χρηστών να μην έλαβε ψήφους, το σύστημα να μπορεί να συμφωνήσει σε μια τιμή. Οι χρήστες πλέον έχουν φτάσει σε ομοφωνία και τερματίζουν τον αλγόριθμο.

Η δεύτερη φάση του BinaryBA★, εμφανίζεται στην περίπτωση του ασύγχρονου δικτύου. Ουσιαστικά από την πρώτη φάση, είτε θα υπάρχουν ψηφοφόροι που θα επιστρέψουν $TIMEOUT$ και άρα θα έχουν $r = block\_hash$ (περίπτωση 1.1), είτε θα επέστρεψαν ένα κενό block και θα έχουν $r = empty\_block$ (περίπτωση 1.2). Όπως και αν έχει, όλοι ψηφίζουν με τον CommitteeVote() την δική τους τιμή r, και τέλος μετράνε τους ψήφους με τον CountVotes().

- Περίπτωση 2.1 : Εάν από την καταμέτρηση με τον αλγόριθμο CountVotes() επιστραφεί $TIMEOUT$ σε κάποιο χρήστη, είτε γιατί δεν έλαβε αρκετές ψήφους είτε γιατί υπήρχαν χρονικές καθυστερήσεις, αυτός υιοθετεί την τιμή $r = empty\_hash$. Έπειτα προχωρά στο επόμενο βήμα του BinaryBA★, όπου θα ψηφίσει για την τιμή $r = empty\_hash$.

- Περίπτωση 2.2 : Εάν από τον CountVotes() προκύψει κενό block, οι χρήστες συνεχιζουν να ψηφίζουν για το κενό block για άλλες 3 ψηφοφορίες. Έπειτα επιστρέφουν την τιμή r του κενού block και τερματίζουν τον αλγόριθμο. Με αυτό το τέχνασμα των τριών ψηφοφοριών, ακόμη και αν το δίκτυο είναι ασύγχρονο, οι ειλικρινείς χρήστες έρχονται σε ομοφωνία για ένα $empty\_hash$ καθώς θα υπάρχουν αρκετές ψήφοι στο σύστημα και στην περίπτωση που ο Adversary καθυστερήσει τις ψήφους του CountVotes .

- Περίπτωση 2.3 : Εάν για κάποιο λόγο οι ψήφοι για το μη κενό block σταλθούν στους χρήστες που επέστρεψαν TIMEOUT στο πρώτο βήμα, θα πρέπει να κρατήσουν την τιμή $r = block\_hash$ και να ψηφίσουν πάλι για αυτή στο επόμενο βήμα. Δηλαδή στο βήμα αυτό δεν γίνεται να επέλθει ομοφωνία για την τιμή $block\_hash$ ακόμη και αν η καθυστέρηση του πρώτου βήματος ήταν απλά θέμα δικτύου και όχι κάποια κίνηση κακόβουλου χρήστη.

Παρατήρηση 6.10.2. Ακόμη και στην περίπτωση 2.1, η ομοφωνία που έχει επέλθει δεν είναι οριστική. Πιο συγκεκριμένα, ο αλγόριθμος BA★ θα επιστρέψει *tentative consensus*. Αναφέρεται στο [3] η εξής περίπτωση : Εάν το δίκτυο είναι τελείως ασύγχρονο, υπάρχει περίπτωση οι χρήστες να έχουν διχαστεί σε δύο ομάδες. Η πρώτη ομάδα έρχεται σε ομοφωνία για ένα $block\_hash$, αλλά η δεύτερη ομάδα δεν λαμβάνει τους ψήφους λόγω δικτύου. Η δεύτερη ομάδα ξαναψηφίζει για το ίδιο $block\_hash$, αλλά πάλι χάνονται οι ψήφοι διότι το δίκτυο δεν έχει αποκατασταθεί. Έτσι η πρώτη ομάδα έχει έρθει σε ομοφωνία για ένα $block\_hash$ ενώ η δεύτερη για ένα $empty\_hash$. Σε αυτή την περίπτωση παρόλο που και οι δύο ομάδες έχουν καταλήξει σε ομοφωνία, θέλουμε αυτού του είδους η ομοφωνία να φαίνεται ότι δεν έχει γίνει με ασφάλεια. Για αυτόν τον λόγο ο BA★ διαχωρίζει την ομοφωνία σε "$TENTATIVE$" και "$FINAL$". Γενικά "$FINAL$" consensus έχουμε μόνο στο πρώτο βήμα του αλγορίθμου, δηλαδή στην περίπτωση 1.3.

Όπως περιγράφτηκε στην παρατήρηση παρατήρηση 6.10.2, στην χειρότερη περίπτωση το δίκτυο θα είναι ασύγχρονο και οι χρήστες θα έχουν διχαστεί σε ομάδες όπου η μια θα δέχεται αρκετές ψήφους για την τιμή $block\_hash$ ενώ η δεύτερη δεν θα λαμβάνει τις ψήφους και θα επιστρέφει TIMEOUT και $r = empty\_hash$. Σε αυτή την περίπτωση που ένας κακόβουλος χρήστης εμποδίζει την δεύτερη ομάδα να έρθει σε ομοφωνία, το δίκτυο θα έχει χωριστεί στα δύο και αυτή η διαδικασία με τις πρώτες δύο φάσεις θα συνεχίζεται χωρίς κάποιο αποτέλεσμα. Αυτό είναι μια εκδοχή που δεν πρέπει να συμβεί και για αυτόν τον λόγο το σύστημα του Algorand χρησιμοποιεί την τρίτη φάση με τον αλγόριθμο CommonCoin().

CommonCoin

Πρόκειται για έναν βοηθητικό αλγόριθμο που χρησιμοποιείται στον BinaryBA★,

και βοηθάει τους χρήστες να συμφωνήσουν σε μια τιμή εάν έχουν διχαστεί. Πιο συγκεκριμένα ο αλγόριθμος αυτός αφορά την δημιουργία ενός κοινού νομίσματος για όλους τους χρήστες. Εάν οι χρήστες είναι ειλικρινείς θα παρατηρούν την ίδια όψη του νομίσματος και άρα θα συμφωνούν σε μια κοινή τιμή.

Algorithm 11: CommonCoin(ctx, $round$, $step$, τ)

```
minhash ← 2^hashlen
for m ϵ incomingMsgs[round, step] do
    < votes, value, sorthash > ← ProcessMsg(ctx, τ, m)
    for 1 ≤ j < votes do
        h ← H(sorthash||j) if h < minhash then
            minhash ← h;
return minhash mod 2
```

Είσοδος

Ο αλγόριθμος δέχεται σαν είσοδο τις πληροφορίες για την κατάσταση του blockchain (ctx), τον γύρο και το βήμα του πρωτοκόλλου ($round, step$) και το αναμενόμενο μέγεθος της επιτροπής τ.

Έξοδος

Όπως αναφέρθηκε ο αλγόριθμος αυτός δημιουργεί ένα νόμισμα στο οποίο οι ειλικρινείς χρήστες θα παρατηρούν την ίδια όψη. Δηλαδή την ίδια τιμή. Ο αλγόριθμος έχει σαν έξοδο την τιμή που παρατηρεί ο χρήστης όταν τον χρησιμοποιήσει. Η τιμή αυτή είναι είτε το 0 είτε το 1.

Περιγραφή αλγόρίθμου

Στον αλγόριθμο CommonCoin() το νόμισμα που δημιουργείται είναι το least significant bit (LSB) μιας τιμής σύνοψης (hash value). Η τιμή αυτή είναι το μικρότερο hash της ψευδοτυχαιότητας $sorthash$ του κάθε χρήστη με τις ψήφους που έχει. Η τιμή $sorthash$ δημιουργείται από τον αλγόριθμο VRF και εξάγεται από τον αλγόριθμο CommitteeVote(), και είναι η ψευδοτυχαιότητα της επιτροπής. Δημιουργείται το hash της ένωσης $sorthash||j$, για κάθε ψήφο j που έχει και επιλέγεται η μικρότερη τιμή. Αυτό πραγματοποιείται για κάθε μέλος της επιτροπής και ελέγχεται το hash όλων των ψήφων. Αφού βρεθεί η μικρότερη τέτοια τιμή $H(sorthash||j)$, στην συνέχεια απομονώνεται το least significant bit αυτής της τιμής. Επειδή οι τιμές αυτές είναι δυαδικοί αριθμοί, το LSB παράγεται με την εφαρμογή του τελεστή mod2. Άρα εφαρμόζεται η πράξη mod2 στο μικρότερο hash $H(sorthash||j)$. Ο αλγόριθμος τέλος θα επιστρέψει την τιμή αυτή : $H(sorthash||j)mod2$.

Στην τρίτη φάση λοιπόν του αλγορίθμου BinaryBA★, όλοι οι χρήστες τρέχουν τον αλγόριθμο CommitteeVote(), φτιάχνουν μια επιτροπή και ψηφίζουν για την τιμή r που προσδιορίστηκε από το προηγούμενο $step$. Έπειτα προσμετρούν τις ψήφους τους σε αυτό το βήμα.

- Εάν πάλι προκύψει TIMEOUT και άρα οι χρήστες δεν μπορούν να συμφωνήσουν σε κάποια τιμή, τότε εφαρμόζεται ο αλγόριθμος CommonCoin(). Όλοι οι χρήστες υπολογίζουν το LSB ενός hash που προκύπτει από τις εξατομικευμένες πληροφορίες της επιτροπής $committee$, και εάν το LSB είναι το ψηφίο 0 τότε ορίζουν ως υποψήφιο block ομοφωνίας το μη κενό $block\_hash$, ενώ αν το LSB είναι το ψηφίο 1 τότε ορίζουν ως υποψήφιο block ομοφωνίας το $empty\_hash$. Στην συνέχεια προχωράνε στο επόμενο βήμα και ψηφίζουν για αυτή την τιμή.

- Εάν ο αλγόριθμος CountVotes() επιστρέψει κάποιο block (είτε κενό είτε μη κενό), οι χρήστες βγαίνουν από την τρίτη φάση με αυτό το block, προχωράνε στο επόμενο βήμα και ξαναψηφίζουν για την ομοφωνία.

Παρατήρηση 6.10.3. Εάν ο χρήστης στον οποίο αντιστοιχεί το μικρότερο $H(sorthash||j)$ είναι ειλικρινής, τότε όλοι οι υπόλοιποι χρήστες θα παρατηρούν το ίδιο hash, και αφού έχει παραχθεί τυχαία μεσω των VRFs,θα είναι και το LSB τυχαίο. Παρ'όλα αυτά υπαρχει και η περίπτωση ο χρήστης που έχει το μικρότερο $H(sorthash||j)$ να είναι κακόβουλος. Τότε ίσως να στείλει την τιμή αυτή σε λίγους χρήστες και άρα να μην βλέπουν όλοι το ίδιο ψηφίο 0 ή 1. Εάν το δίκτυο έχει φτάσει στην τρίτη φάση του BinaryBA★, η πιθανότητα να έρθουν οι χρήστες σε ομοφωνία περιγράφεται παρακάτω.

Ποιά είναι η πιθανότητα μετά την τρίτη φάση του BinaryBA⋆ να έχουμε ομοφωνία υπό αυστηρό συγχρονισμό;

- Εάν οι χρήστες ψηφίσουν για μια τιμή r και όλοι λάβουν αρκετές ψήφους ($> T \cdot \tau$), τότε ο αλγόριθμος CountVotes() δεν θα χρησιμοποιηθεί και στα επόμενα δύο βήματα το δίκτυο θα έρθει σε ομοφωνία για το r.

- Εάν ο χρήστης που καθορίζει το LSB, δηλαδή το νόμισμα, είναι ειλικρινής τότε διακρίνονται οι εξής περιπτώσεις :

  1. **Κάποιοι ειλικρινείς χρήστες μάζεψαν ψήφους για ένα μη κενό $block\_hash$ ενώ οι υπόλοιποι ειλικρινείς χρήστες επέστρεψαν TIMEOUT και υλοποίησαν τον CommonCoin().** Η πιθανότητα να έρθουν σε ομοφωνία όλοι οι ειλικρινείς χρήστες είναι και η ομάδα που είδε ψήφους για το $block\_hash$ αλλά και η ομάδα που υλοποίησε τον CommonCoin(), να ψηφίσουν για το $block\_hash$ στο επόμενο βήμα. Άρα θα πρέπει και το κοινό νόμισμα να επιστρέψει την τιμή για το μη κενό block. Δηλαδή θα πρέπει το LSB να είναι το 0. Η πιθανότητα να είναι 0 είναι $\frac{1}{2}$. Άρα σε αυτή την περίπτωση οι ειλικρινείς χρήστες θα έρθουν σε ομοφωνία με πιθανότητα $\frac{1}{2}$.

  2. **Κάποιοι ειλικρινείς χρήστες μάζεψαν ψήφους για ένα κενό $empty\_hash$ ενώ οι υπόλοιποι ειλικρινείς χρήστες επέστρεψαν TIMEOUT και υλοποίησαν τον CommonCoin().** Η πιθανότητα να έρθουν σε ομοφωνία όλοι οι ειλικρινείς χρήστες είναι και η ομάδα που είδε ψήφους για το $empty\_hash$ αλλά και η ομάδα που υλοποίησε τον CommonCoin(), να ψηφίσουν για το $empty\_hash$ στο επόμενο βήμα. Άρα θα πρέπει και το κοινό νόμισμα να επιστρέψει την τιμή για το κενό block. Δηλαδή θα πρέπει το LSB να είναι το 1. Η πιθανότητα να είναι 1 είναι $\frac{1}{2}$. Άρα σε αυτή την περίπτωση οι ειλικρινείς χρήστες θα έρθουν σε ομοφωνία με πιθανότητα $\frac{1}{2}$.

  3. **Όλοι οι ειλικρινείς χρήστες επέστρεψαν TIMEOUT και υλοποίησαν τον CommonCoin().** Η πιθανότητα να έρθουν σε ομοφωνία όλοι οι ειλικρινείς χρήστες είναι ο CommonCoin() να επιστρέψει σε όλους την ίδια τιμή και να ψηφίσουν για αυτή στο επόμενο βήμα. Αυτό είναι σίγουρο ότι θα συμβεί αφού ο χρήστης που καθορίζει το νόμισμα είναι ειλικρινής.

- Η πιθανότητα δύο ειλικρινείς χρήστες να επέστρεψαν διαφορετική τιμή από τον αλγόριθμο CountVotes(), εκτός από το TIMEOUT, είναι αμελητέα. (αποδείχθηκε στο Λήμμα 6.10.1)

Τελικά σε οποιαδήποτε περίπτωση εάν ο χρήστης που καθορίζει το νόμισμα

είναι ειλικρινής, η πιθανότητα το σύστημα να έρθει σε ομοφωνία είναι μεγαλύτερη ή ίση με $\frac{1}{2}$. Η πιθανότητα ο χρήστης αυτός να είναι ειλικρινής είναι $> \frac{2}{3}$ καθώς οι ειλικρινείς χρήστες σε οποιαδήποτε επιτροπή είναι πάντα περισσότεροι από τα $\frac{2}{3}$ των μελών. Τελικά η πιθανότητα για ομοφωνία είναι $> \frac{2}{3} \cdot \frac{1}{2}$ άρα $> \frac{1}{3}$.

Λήμμα 6.10.2. Υπό αυστηρό συγχρονισμό, εάν όλοι οι ειλικρινείς χρήστες ξεκινήσουν το BA★ με την ίδια τιμή r στον ίδιο γύρο, τότε θα επιστρέψουν $FINAL$ *consensus* για αυτή την τιμή r.

Απόδειξη
Έστω ότι όλοι οι ειλικρινείς χρήστες αρχικοποιούν τον αλγόριθμο Reduction() με την ίδια τιμή $block\_hash$. Τότε λόγω της υπόθεσης αυστηρού συγχρονισμού, όλοι οι ειλικρινείς χρήστες θα στείλουν ψήφους για αυτή την τιμή και αντίστοιχα όλοι οι ειλικρινείς χρήστες θα λάβουν τις ψήφους. Αυτό θα συμβεί και στο πρώτο και στο δεύτερο βήμα του Reduction() όπου θα παρατηρήσουν όλοι περισσότερες από $T \cdot \tau$ ψήφους. Τελικά η τιμή που θα αρχικοποιηθεί το BinaryBA★ θα είναι κοινή για τους ειλικρινείς χρήστες $r = block\_hash$, και από το πρώτο κιόλας βήμα με αυστηρό συγχρονισμό θα έχουμε αρκετές ψήφους για $FINAL$ *consensus*.
□

Θεώρημα 6.10.2. *Υπό αυστηρό συγχρονισμό, ο BA★ φτάνει σε finalconsensus σε 4 βήματα εάν ο χρήστης με την μεγαλύτερη προτεραιότητα στο Block Proposal() ήταν ειλικρινής, αλλιώς φτάνει σε ομοφωνία σε 13 βήματα. Η πιθανότητα ο BinaryBA★ να μην ολοκληρωθεί σε $MaxSteps = 150$ είναι αμελητέα.*

Απόδειξη
Η πρώτη περίπτωση όπου ο χρήστης με την μεγαλύτερη προτεραιότητα στο Block Proposal() είναι ειλικρινής, αποδείχτηκε στο Λήμμα 3.10.2. Στην χειρότερη περίπτωση όπου ο χρήστης με την μεγαλύτερη προτεραιότητα είναι κακόβουλος, θα καθυστερήσει τις ψήφους κατάλληλα ώστε στις δύο πρώτες φάσεις του BinaryBA★ να μην μπορούν οι χρήστες να έρθουν σε ομοφωνία. Παρ' όλα αυτά, στην τρίτη φάση με το κοινό νόμισμα, όλοι οι ειλικρινείς χρήστες έρχονται σε ομοφωνία με πιθανότητα $> \frac{1}{3}$ στα επόμενα 2 βήματα (αποδείχθηκε εδώ). Άρα εκτός από τα δύο πρώτα βήματα του πρωτοκόλλου BinaryBA★, κάθε 3 βήματα η πιθανότητα ομοφωνίας είναι $\frac{1}{3}$. Άρα για ομοφωνία ο αλγόριθμος χρειάζεται $2 + 3 \cdot 3 = 11$ βήματα, και άρα ο BA★ μαζί με τα δύο βήματα του Reduction() χρειάζεται $11+2 = 13$ βήματα για κακόβουλο block proposer.
Η πιθανότητα ο αλγόριθμος BinaryBA★ να μην ολοκηρωθεί σε 150 βήματα, μπορεί να υπολογιστεί με την διωνυμική κατανομή. Επειδή όπως αναφέρθηκε, η ομοφωνία έρχεται κάθε 3 βήματα με πιθανότητα $> \frac{1}{3}$, άρα

έχουμε $\frac{150}{3}$ πειράματα με πιθανότητα επιτυχίας $\frac{1}{3}$. Άρα η πιθανότητα να έχουμε 0 επιτυχίες σε $\frac{150}{3}$ πειράματα είναι :

$$P(X=0) = \binom{\frac{150}{3}}{0} \frac{1}{3}^0 (1-\frac{1}{3})^{\frac{150}{3}-0} \Rightarrow$$

$$P(X=0) = 1 \cdot 1 \cdot (\frac{2}{3})^{50} \Rightarrow$$

$$P(X=0) \approx 1,57 \cdot e^{-9}$$

□

6.11 Recovery Protocol

Το δίκτυο του Algorand είναι ασφαλές για την υπόθεση μερικού συγχρονισμού, ότι δηλαδή μετά από μια σχετικά μεγάλη ασύγχρονη περίοδο b, θα ακολουθεί πάντα μια μικρότερη αλλά αναγκαία περίοδος s αυστηρού συγχρονισμού. Το πρόβλημα είναι ότι σε μια ασύγχρονη περίοδο, που οι χρήστες μπορεί να καταλήξουν σε διαφορετικά blocks το πρωτόκολλο BA★ δεν θα μπορεί να επιστρέψει ομορφωνία, για παράδειγμα εάν ξεπεραστούν τα $MaxSteps$ του BinaryBA★. Το ζήτημα είναι το δίκτυο να φτάνει σε Final Consensus και οι χρήστες να μην υπάρχει πιθανότητα να μπερδευτούν για το ποιο είναι το τελευταίο block στην αλυσίδα. Πάντα θα πρέπει να υπάρχει μια ξεκάθαρη αλληλουχία στα κουτιά συναλλαγών που επικυρώνονται, και αν προκύψουν δύο block σε έναν γύρο το πιο πιθανό είναι να δημιουργηθεί fork στην αλυσίδα. Αυτό συμβαίνει διότι για να προσμετρήσει ένας χρήστης τις ψήφους που δέχεται στον αλγόριθμο CountVotes() και άρα στον ProcessMsg(), θα πρέπει ο ψηφοφόρος και ο παραλήπτης να βλέπουν το ίδιο τελευταίο block ($hprev$), αλλιώς επιστρέφεται η τιμή 0 και η ψήφος δεν προσμετράται (5η σειρά αλγόριθμος ProcessMsg()). Προκειμένου να λυθεί αυτό το fork μέχρι να επέλθει αυστηρός συγχρονισμός, ακολουθείται το Recovery Protocol.
Στόχος του Recovery Protocol είναι οι χρήστες να μπορέσουν να συμφωνήσουν σε κάποιο από τα forks που δημιουργούνται, ομόφωνα, ώστε να ακολουθήσουν αυτό το "μονοπάτι". Προκειμένου να γίνει αυτό, το Algorand διαβάζει κάθε ψήφο και έχει λίστα από τα πιθανά forks που υπάρχουν στο σύστημα. Ανά κάποια χρονικά διαστήματα (αναφέρεται στο [3] ανά μία ώρα), οι χρήστες μέσω των loosely synchronized clocks που διαθέτουν, κάνουν παύση στο πρωτόκολλο και αποφασίζουν με το Recovery Protocol, ποιό είναι το fork που θα ακολουθηθεί.

Το πρωτόκολλο αυτό ξεκινάει με τους χρήστες να προτείνουν fork χρησιμοποιώντας τον μηχανισμό BlockProposal(). Άρα έχουμε μια επιτροπή από $fork\ proposers$. Ο τρόπος με τον οποίο προτείνουν fork είναι χρησιμοποιώντας ένα κενό block του οποίου ο δείκτης δείχνει την μεγαλύτερη αλυσίδα που έχουν παρατηρήσει. Από αυτές τις προτάσεις, κρατάνε αυτή με την μεγαλύτερη προτεραιότητα που ακολουθεί τον κανόνα της μεγαλύτερης αλυσίδας. Ο λόγος για τον οποίο συμβαίνει αυτό είναι επειδή είναι αναγκαίο η αλυσίδα να έχει όσο το δυνατόν περισσότερες επικυρωμένες συναλλαγές, και άρα επικυρωμένα blocks. Αφού όλοι έχουν λάβει το ανάλογο κενό block, το χρησιμοποιούν για να αρχικοποιήσουν τον αλγόριθμο BA★.
Αφού οι χρήστες περάσουν στο κομμάτι της Βυζαντινής συμφωνίας, ψηφίζουν σύμφωνα με τους αλγορίθμους του BA★. Επειδή όμως το πρωτόκολλο βρίσκεται σε κατάσταση ασύγχρονου δικτύου, είναι σημαντικό οι πληροφορίες για τα κρυπτονομίσματα των λογαριασμών ($weights$) και η τυχαιότητα που ονομάζεται $seed$, να είναι αδιάβλητα για λόγους ασφάλειας. Για αυτό, οι δύο αυτές πληροφορίες αντλούνται από την τελευταία περίοδο αυστηρού συγχρονισμού του δικτύου. Το Algorand είναι χωρισμένο σε περιόδους μήκους b οπότε διαλέγουμε τον $seed$ από την προτελευταία ολοκληρωμένη περίοδο b, και τα $weights$ από το τελευταίο block b χρόνο πριν. Ο λόγος που χρησιμοποιείται η προτελευταία περίοδος είναι διότι η τελευταία μπορεί να έχει ακόμη κάποια διακλάδωση που δεν έχει λυθεί. Οι χρήστες προσπαθούν συνεχώς να έρθουν σε ομοφωνία για κάποιο fork. Για αυτό τον λόγο υπάρχει η ανάγκη να δημιουργηθούν πολλές επιτροπές και άρα χρειάζεται ο σπόρος $seed$ να ανανεώνεται. Η ανανέωση γίνεται με την εφαρμογή της συνάρτησης σύνοψης/κατακερματισμού $H()$.

6.12 Πρακτικά Ζητήματα

Πώς ενημερώνονται τα νέα μέλη που μπαίνουν στο σύστημα ;

Εκτός από τους ήδη υπάρχοντες χρήστες, οποιοσδήποτε λάβει κάποια κρυπτονομίσματα Algorand μπορεί να δημιουργήσει λογαριασμό, κλειδιά και να συμμετάσχει στο πρωτόκολλο ομοφωνίας του δικτύου οποιαδήποτε στιγμή. Θα πρέπει να υπάρχει αξιόπιστος τρόπος ενημέρωσης των νέων χρηστών για την κατάσταση της αλυσίδας. Το Algorand αντιμετωπίζει αυτή την κατάσταση με την χρήση πιστοποιητικών ($certificates$). Κάθε block που μπαίνει στο blockchain συνοδεύεται και από ένα πιστοποιητικό που περιλαμβάνει το σύνολο των ψήφων που έλαβε στο τελευταίο βήμα του BA★, και σε πόσα βήματα επήλθε η ομοφωνία. Άρα οποιοσδήποτε νέος ή υπάρχων χρήστης του πρωτοκόλλου μπορεί να ζητήσει το πιστοποιητικό οποιουδήποτε επικυρωμένου block και να μετρήσει τις ψήφους, να ελέγξει τις υπογραφές, την τυχαιότητα vrf_hash, και τον γύρο και το βήμα που συνέβησαν.
Ενώ η ιδέα του $certificate$ είναι πολύ αποτελεσματική, θα πρέπει να

ελεγχθεί και η πιθανότητα τροποποίησης κάποιου πιστοποιητικού από κακόβουλο χρήστη προς όφελος του. Ένα είδος επίθεσης που θα μπορούσε να συμβεί είναι στην περίπτωση που ο Adversary καταφέρει να αποδείξει ότι ο αλγόριθμος BA★ έληξε σε ένα μεγάλο βήμα όπου αυτός έχει τον έλεγχο των περισσότερων ψηφοφόρων. Σε αυτή την περίπτωση μπορεί, αφού τους ελέγχει, να δημιουργήσει ένα πιστοποιητικό με τις υπογραφές τους. Η επιλογή του μεγέθους της επιτροπής τ είναι τέτοια ώστε να μην υπάρχει σημαντική πιθανότητα ο Adversary να βρεί ένα τέτοιο βήμα (αναφέρεται ότι η πιθανότητα να συμβεί αυτή η επίθεση σε κάθε βήμα είναι $< 2^{-166}$).

Μπορεί ο αντίπαλος να στοχοποιήσει χρήστες ;

Δεδομένου ότι το πρωτόκολλο λειτουργεί με επιτροπές που ψηφίζουν, μια προφανής επίθεση θα ήταν να διαφθαρούν τα μέλη της επιτροπής ώστε να μπορεί να ελεγχθεί μεγάλο μέρος των ψήφων. Αυτό αντιμετωπίζεται συγχρόνως με δύο τρόπους :

- Τα μέλη της επιτροπής δημοσιοποιούν την ταυτότητα και τον ρόλο μαζί με την ψήφο τους, άρα ουσιαστικά ενεργούν μία φορά δημόσια από την στιγμή που θα εκλεχθούν. Αυτό συμβαίνει διότι διαδικασία εκλογής είναι ιδιωτική για τον κάθε χρήστη.

- Η επιτροπή αλλάζει σε κάθε βήμα οποιουδήποτε αλγορίθμου στο Algorand. Αυτό σημαίνει πως και να διαφθαρεί κάποιος χρήστης αφού εμφανίσει την ταυτότητα του, ο ρόλος του στην επιτροπή έχει τελειώσει.

Στην περίπτωση όμως που αρκετοί χρήστες στοχοποιηθούν και διαφθαρούν μακροπρόθεσμα, ο αντίπαλος θα μπορεί να φτιάξει ένα ψεύτικο πιστοποιητικό με τα κλειδιά τους, και άρα να διχάσει το σύστημα. Για τον λόγο αυτό προτείνεται η χρήση εφήμερων κλειδιών τόσο στο white paper [3] όσο και στο [4]. Η λύση είναι καθε φορά που ένας χρήστης στέλνει κάποιο μήνυμα, ειδικότερα ψήφου, θα το υπογράφει με ένα εφήμερο κλειδί. Για την δημιουργία αυτών των κλειδιών μπορούν να χρησιμοποιηθούν identity based signature schemes [49], [50].
Πιο συγκεκριμένα, κάθε χρήστης i παράγει ένα ζεύγος κλειδιών PMK (*public master key*) και SMK (*secret master key*) από ένα identity based signature scheme. Το PMK μπορεί να είναι το δημόσιο κλειδί (pk) που έχει ήδη ο χρήστης. Από το SMK παράγεται μια σειρά ιδιωτικών κλειδιών sk_i, ένα για κάθε γύρο του πρωτοκόλλου. Υπάρχει δηλαδή μια τριπλέτα χρήστη-γύρου-βήματος (i, r, s):

$$S = \{i\}x\{r, r+1, ..., r+10^6\}x\{1, 2, ..., \mu\}$$

, σύμφωνα με την οποία μπορούν να παραχθούν και να ελεγχθούν τα sk_i από οποιονδήποτε χρήστη γνωρίζει το pk_i για τους επόμενους 10^6

γύρους. Ο όρος μ είναι τα μέγιστα βήματα που μπορεί να φτάσει ένας γύρος του πρωτοκόλλου. Οπότε επιλέγεται ένα στοιχείο $(i, r, s)\epsilon S$ και για κάθε τέτοιο στοιχείο φτιάχνεται ένα $sk_i^{r,s}$. Αφού παραχθούν όλα αυτά τα κλειδιά για κάθε (i, r, s), το SMK καταστρέφεται. Ο χρήστης έχει δημοσιοποιήσει το PMK, δηλαδή το pk_i, και όταν εκτελεί ένα βήμα του πρωτοκόλλου στο οποίο πρέπει να υπογράψει, υπογράφει με το αντίστοιχο $sk_i^{r,s}$, στην συνέχεια το καταστρέφει και κοινοποιεί το μήνυμα. Όταν σταλθεί το μήνυμα στο δίκτυο, το εφήμερο κλειδί που είναι μιας χρήσης έχει καταστραφεί, άρα και να τον διαφθείρει ο Adversary, δεν θα μπορεί να χρησιμοποιήσει το ιδιωτικό κλειδί του.

Παράδειγμα χρήσης εφήμερων κλειδιών στον αλγόριθμο BlockProposal():

Ορίζουμε την υπογραφή του μηνύματος m από τον χρήστη i με εφήμερα κλειδιά ως : $esig_i(m)$.

Κάθε χρήστης i από τον γύρο r-k που έχει το πιστοποιητικό του προηγούμενου γύρου $CERT^{-1}$:

1. Παραλαμβάνει τον $seed_{r-1}$ και το $H(B^{r-1})$ από το $CERT^{-1}$.
2. Τρέχει τον αλγόριθμο Cryptographic Sortition
3. Εάν έχει επιλεχθεί $(j \neq 0)$ ως πιθανός block proposer:
   (a) Δημιουργεί ένα σύνολο από συναλλαγές που υπάρχουν στο διαθέσιμες σύστημα PAY_i^r
   (b) Δημιουργεί το καινούριο block $B_i^r = (r, H(B^{r-1}), SIG_i(seed_{r-1}), PAY_i^r)$
   (c) Υπολογίζει $\sigma_i^{r,1} = (hash, \pi)$
   (d) Φτιάχνει το $m_i^r = (B_i^r, esig_i(B_i^r), \sigma_i^{r,1})$.
   (e) Καταστρέφει το εφήμερο κλειδί $sk_i^{r,1}$.
   (f) Διαμοιράζεται ξεχωριστά τα δύο μηνύματα m_i^r και $SIG_i(seed_{r-1}, \sigma_i^{r,1})$.

Κεφάλαιο 7

Ouroboros

Το Ouroboros πρόκειται για ένα πρωτόκολλο κρυπτονομίσματος που χρησιμοποιεί την τεχνολογία blockchain για την αποθήκευση των συναλλαγών και τον αλγόριθμο proof-of-stake για την επικύρωση και προσθήκη τους στην αλυσίδα. Η κλασική εκδοχή το Ouroboros παρουσιάστηκε για πρώτη φορά στο συνέδριο Annual International Cryptology Conference το 2017, με συγγραφείς τους Pr.Aggelos Kiayias, Pr.Alexander Russell, Dr. Bernardo David, και Pr.Roman Oliynykov [1]. Έκτοτε αποτελεί βασικό πρωτόκολλο πίσω από το κρυπτονίμισμα Ada του Cardano blockchain. Πλέον το κλασικό Ouroboros, λόγω της χρησιμότητάς και της ακαδημαϊκής σημαντικότητας του, έχει αναπτυχθεί σε μια οικογένεια πρωτοκόλλων. Σε αυτή την εργασία θα αναλυθούν οι πυλώνες του Ouroboros που είναι το Ouroboros Classic [1] και το Ouroboros Praos [2].

7.1 Ouroboros Classic

Η κλασική εκδοχή του Ouroboros [1] πρόκειται για ένα proof-of-stake πρωτόκολλο που επιτυγχάνει να ικανοποιήσει ιδιότητες ασφάλειας παρόμοιες με αυτές του proof-of-work blockchain του Bitcoin. Στην αρχή δίνεται βάση στο πώς λειτουργεί το blockchain του Ouroboros σε μια στατική κατάσταση ομοφωνίας σε ένα block, στην συνέχεια περιγράφεται η δυναμική εξέλιξη του πρωτοκόλλου, με σταδιακή προσθήκη μηχανισμών για πιο ρεαλιστική προσέγγιση. Επίσης έχει αναπτυχθεί ένας μηχανισμός επιβράβευσης για όσους χρήστες συμμετέχουν ενεργά στο consensus , ο οποίος έχει αποδειχθεί αποδοτικός και έμπιστος μέσω της Θεωρίας Παιγνίων.

7.1.1 Εισαγωγή

Το Ouroboros Classic λειτουργεί σε slots και σε κάθε slot παράγεται ένα block. Μια διαδοχή από slots φτιάχνουν με περίοδο που ονομάζεται εποχή. Σε κάθε εποχή εκλέγεται μια επιτροπή χριστών, η οποία περιλαμβάνει το σύνολο των ατόμων που θα παράξουν τα blocks που ανήκουν σε κάθε slot αυτής της εποχής. Η επιτροπή δημιουργείται με την χρήση μιας τυχαιότητας, που παράγεται στο πρώτο block και στην συνέχεια ανανεώνεται σε κάθε εποχή. Όπως ανανεώνεται η τυχαιότητα έτσι ανανεώνεται και η επιτροπή σε κάθε εποχή.

7.1.2 Περιγραφή μοντέλου

Χρόνος
Ο χρόνος χωρίζεται σε διακεκριμένες μονάδες που ονομάζονται slots. Σε

κάθε slot παράγεται το πολύ ένα block που θα μπει στο blockchain. Τον χρόνο μπορούν να τον αντιληφθούν οι χρήστες μέσω συγχρονισμένων ρολογιών που διαθέτουν (roughly synchronized), τα οποία υποδεικνύουν το κάθε slot. Κάθε χρήστης έχει πρόσβαση στον χρόνο και τυχούσες αποκλίσεις που μπορεί να υπάρχουν είναι αμελητέες μπροστά στο μήκος ενός slot. Το μήκος κάθε μονάδας χρόνου στο Ouroboros Classic είναι τέτοιο ώστε οποιοσδήποτε έμπιστος χρήστης να μπορεί να στείλει αλλά και να παραλάβει κάποιο μήνυμα έμπιστου χρήστη ανεξάρτητα από τις καθυστερήσεις που μπορεί να υπάρχουν στο δίκτυο. Το πρωτόκολλο επίσης ονομάζει το συνολο από R γειτονικά slots ως εποχή ($epoch$), και είναι μια περίοδος στην οποία θεωρείται πως η κατανομή των χρημάτων θεωρείται σταθερή και σε κάθε εποχή εκλέγεται η επιτροπή που θα παράξει τα νέα block.

Βασική υπόθεση ειλικρινούς πλειοψηφίας
Το Ouroboros είναι ένα proof-of-stake πρωτόκολλο. Αυτό σημαίνει πως

οι χρήστες που συμμετέχουν και επικυρώνον συναλλαγές επιλέγονται με πιθανότητα ανάλογη των χρημάτων ($stake$) που διαθέτουν στο σύστημα. Αυτό σημαίνει πως και για να λειτουργεί σωστά το πρωτόκολλο θα πρέπει οι ειλικρινείς χρήστες μαζί να διαθέτουν πάνω από το 50% των χρημάτων του συστήματος. Αυτό είναι μια βασική υπόθεση και δεν θα πρέπει να παραβιάζεται καθ' όλη την διάρκεια του πρωτοκόλλου. Προκειμένου να διατηρείται αυτή η ειλικρινής πλειοψηφία χρημάτων, υπάρχει στο πρωτόκολλο μια κατανομή χρημάτων σε κάθε slot, με το $stake$ που ανήκει σε οποιονδήποτε χρήστη στο σύστημα. Η μέγιστη στατιστική διαφορά

μεταξύ δύο τέτοιων κατανομών ονομάζεται $stake\ shift$.

Μοντέλο αντιπάλου
Ο αντίπαλος $\mathcal{A}$ ($Adversary$) είναι ελεύθερος να διαφθείρει και να ελέγξει

οποιονδήποτε χρήστη θέλει. Επίσης μπορεί να διαβάσει όλα τα μηνύματα που στέλνονται, να αλλάξει την σειρά αποστολής, καθώς και να καθυστερήσει μηνύματα πάντα σύμφωνα με την υπόθεση συγχρονισμού του δικτύου. Όλα αυτά όμως μπορούν να συμβούν με δύο περιορισμούς. Ο πρώτος αφορά την καθυστέρηση διαφθοράς, δηλαδή από την στιγμή που θα αιτηθεί στο σύστημα την διαφθορά ενός χρήστη θα περάσουν D slots μέχρι να αποκαλυφθεί το ιδιωτικό κλειδί του χρήστη στον $\mathcal{A}$. Ο δεύτερος αφορά το ποσοστό των χρηστών που μπορεί να διαφθείρει αναλογικά με τα χρήματα που διαθέτουν στο σύστημα. Με άλλα λόγια, επειδή το Ouroboros είναι ένα proof-of-stake σύστημα που λειτουργεί υπό την υπόθεση ότι πάνω από 50% των χρημάτων ανήκουν σε έμπιστους χρήστες, ο $\mathcal{A}$ είναι περιορισμένος να ελέγχει μέχρι ένα συγκεκριμένο ποσοστό χρημάτων, $\frac{1}{2} - \epsilon$. Ο $\mathcal{A}$ ονομάζεται $\frac{1}{2} - \varepsilon$-*αρχικά-περιορισμένος* όσων αφορά την αρχική κατανομή των χρημάτων στο πρώτο block, και $\frac{1}{2} - \varepsilon$-$\sigma$-*περιορισμένος* κατά την εκτέλεση του πρωτοκόλλου, όπου σ είναι το stake shift. Από την στιγμή που περάσει η καθυστέρηση διαφθοράς για τον $\mathcal{A}$, τότε αυτός μπορεί να πάρει εξ' ολοκλήρου την θέση του χρήστη που ελέγχει.

Ιδιότητες ενός blockchain συναλλαγών
Για να θεωρείται ένα κατανεμημένο σύστημα εγγραφών ισχυρό, θα πρέπει

το πρωτόκολλο που το εφαρμόζει να φροντίζει να είναι χωρισμένο σε blocks που καθορίζουν την σειρά που οι συναλλαγές εγγράφονται στο σύστημα. Με άλλα λόγια τα blocks στο blockchain θα πρέπει να έχουν μια αυστηρή αλληλουχία ως προς τον χρόνο που προστέθηκαν στο σύστημα. Υπάρχουν δύο βασικές ιδιότητες που είναι αναγκαίο να ικανοποιούνται για οποιοδήποτε σύστημα εγγραφής:

- **Persistence** (με παράμετρο $k \epsilon \mathbb{N}$)
  Η ιδιότητα αυτή έχει να κάνει με την *σταθερότητα* μιας συναλλαγής. Εάν ένας κόμβος θεωρήσει μια συναλλαγή *σταθερή* ($stable$) τότε και οι υπόλοιποι κόμβοι όταν ερωτηθούν θα πρέπει να βλέπουν την συγκεκριμένη συναλλαγή σταθερή και συμφωνούν σε ό,τι προϋπάρχει πριν από αυτή. Η συναλλαγή θεωρείται $\sigma ταθερ$ αν και μόνο αν η συναλλαγή βρίσκεται k blocks βαθιά στο blockchain.

- **Liveness** (με παράμετρο $u \epsilon \mathbb{N}$)
  Αν όλοι οι έμπιστοι χρήστες στο σύστημα συμφωνήσουν στο να συμπεριληφθεί μια καινούρια συναλλαγή στο blockchain, τότε μετά

από χρόνο u slot όλοι οι κόμβοι που θα ερωτηθούν αν απαντήσουν ειλικρινώς θα κρίνουν την συναλλαγή αυτή ως *σταθερή*.

Είναι απαραίτητο, για να θεωρείται το blockchain του πρωτοκόλλου Ouroboros ασφαλές, να ικανοποιούνται επίσης οι τρείς παρακάτω ιδιότητες :

- **Common prefix** (CP: $k \epsilon \mathbb{N}$)
  Δύο αλυσίδες C_1, C_2 υιοθετημένες από δύο έμπιστες ομάδες χρηστών κατά την έναρξη των $sl_1 \leq sl_2$, είναι τέτοιες ώστε $C_1^{\lceil k} \preceq C_2$. Με τον όρο $C_1^{\lceil k}$ εννοούμε την αλυσίδα που προκύπτει εάν αφαιρέσουμε από την C_1 τα k τελευταία blocks. Η ένδειξη $\preceq$ συμβολίζει την σχέση προθήματος.

- **Honest Chain Growth** (HCG: $\tau \epsilon [0,1],\ s \epsilon \mathbb{N}$)
  Αν C είναι μια αλυσίδα υιοθετημένη από έμπιστη ομάδα χρηστών, και sl_2 είναι το slot με το τελευταίο block της C, και sl_1 ένα προηγούμενο slot με έμπιστο block. Αν $sl_2 \geq sl_1 + s$, τότε ο αριθμός των block που υπάρχουν μετά το sl_1 στην C είναι τουλάχιστον $\tau \cdot s$. Το τ ονομάζεται συντελεστής ταχύτητας.

- **Existential Chain Quality** ($\exists$ CQ: $s \epsilon \mathbb{N}$)
  Έστω μια αλυσίδα C υιοθετημένη από μια έμπιστη ομάδα κατα την έναρξη ενός slot. Τότε οποιοδήποτε τμήμα της C επεκτείνει s προηγούμενα slot, περιέχει τουλάχιστον ένα έμπιστα δημιουργημένο block.

Υποθέσεις ασφάλειας
Το πρωτόκολλο του Ouroboros Classic είναι ασφαλές κάτω από τις εξής

υποθέσεις :

- Το πρωτόκολλο λειτουργεί υπό συγχρονισμένο δίκτυο. Δηλαδή οποιοσδήποτε ειλικρινής χρήστης μπορεί να επικοινωνήσει με οποιονδήποτε άλλο με γνωστή πεπερασμένη καθυστέρηση.

- Οι χρήστες μπορούν να είναι ενεργοί ή και οχι. Υποθέτουμε πως σε κάθε slot υπάρχει ενεργή τουλάχιστον μία έμπιστη ομάδα χρηστών.

- Υπάρχει μέγιστος αριθμός slot που ένας χρήστης μπορεί να είναι ανενεργός. Με άλλα λόγια οι χρήστες δεν παραμένουν για πολλή ώρα εκτός σύνδεσης, εάν επιλεγούν να συμμετάσχουν θα πρέπει να είναι διαθέσιμοι για όσο χρειαστεί.

- Ο αντίπαλος $\mathcal{A}$ επιδέχεται περιορισμό στο ποσοστό των χρηστών που μπορεί να ελέγξει ($\frac{1}{2}$*−ε-αρχικά-περιορισμένος και* $\frac{1}{2}$*−ε-σ-περιορισμένος*)

και καθυστέρηση στην αίτηση διαφθοράς χρηστών. (αναλυτικότερα εδώ)

Παράμετροι πρωτοκόλλου

(i) Παράμετρος $k \in \mathbb{N}$: είναι ο αριθμός των block που θα πρέπει να ακολουθούν το block μιας συναλλαγής, ώστε αυτή να θεωρείται αμετάβλητο κομμάτι του blockchain.

(ii) Παράμετροι $\varepsilon, \sigma \in [0,1]$: είναι αυτοί που φράσουν το ποσοστό των χρημάτων που μπορεί να ελέγξει ο $\mathcal{A}$ μέσω των χρηστών που διαφθείρει.

(iii) Παράμετρος $D \in \mathbb{N}$: είναι ο αριθμός των slot που αντιπροσωπεύουν την καθυστέρηση διαφθοράς κάποιου χρήστη από τον $\mathcal{A}$ μετά το μήνυμα διαφθοράς του προς το σύστημα.

(iv) Παράμετρος $L \in \mathbb{N}$: είναι η ζωή του συστήματος μετρημένη σε slot.

(v) Παράμετρος $R \in \mathbb{N}$: είναι το μήκος μιας εποχής μετρημένο σε slot.

Βασικοί ορισμοί
Θα παρουσιαστούν κάποιοι βασικοί ορισμοί όπως αναγράφονται στο [1]

με σκοπό την εξοικείωση με τους συμβολισμούς του πρωτοκόλλου.

Ορισμός 7.1.1 (**Genesis Block**)**.** Το Genesis Block (B_0) περιλαμβάνει τα δημόσια κλειδιά των χρηστών προς αντιστοιχία με τα χρηματικά ποσά που διαθέτουν στο σύστημα $(pk_1, s_1), (pk_2, s_2), ...(pk_n, s_n)$. Επίσης περιλαμβάνει βοηθητικές πληροφορίες ρ οι οποίες χρησιμεύουν στο να παραχθεί ο σπόρος τυχαιότητας για την εκλογή της επιτροπής που θα παράξει τα επόμενα blocks.

Ορισμός 7.1.2 (**State**)**.** Είναι ένα αλφαριθμητικό $st \in \{0,1\}^{\lambda}$ και προσδιορίζει μια κατάσταση της αλυσίδας.

Ορισμός 7.1.3 (**Block**)**.** Κάθε block B που δημιουργείται σε ένα slot περιέχει την τρέχουσα κατάσταση της αλυσίδας st, τις πληροφορίες που θα μπούν στην αλυσίδα $d \in \{0,1\}^*$, τον χρόνο που δημιουργήθηκε $sl \in \{1, ..., R\}$ και την υπογραφή του χρήστη που το παρήγαγε $\sigma = Sign_{sk}(st, d, sl)$.

Ορισμός 7.1.4 (**Blockchain**)**.** Μια αλυσίδα blockchain C με αρχή το genesis block B_0, είναι μια ακολουθία από blocks $B_1, ...B_n$ συσχετιζόμενα με μια αντίστοιχη ακολουθία από slots που είναι αυστηρά αυξανόμενη. Σε κάθε B_i, η κατάσταση προσδιορίζεται ως η τιμή κατακερματισμού του προηγούμενου block $H(B^{i-1})$. Το μήκος της αλυσίδας συμβολίζεται ως $len(C) = n$ και είναι το νούμερο n από blocks που περιέχει. Το τελευταίο block της C, B_n, ονομάζεται $head(C)$. Ένα κενό string το ορίζουμε ως $head(\varepsilon) = \varepsilon$.

Ορισμός 7.1.5 (**Πρόθημα**)**.** Σε μια αλυσίδα blockchain C μήκους n, εάν αφαιρέσουμε τα k δεξιότερα blocks, προκύπτει η $C^{\lceil k}$. Για $k \geq len(C)$, έχουμε $C^{\lceil k} = \varepsilon$. Για δύο αλυσίδες C_1, C_2, λέμε ότι η C_1 είναι πρόθημα της C_2 και συμβολίζουμε με $C_1 \preceq C_2$, εάν αφαιρώντας κάποια blocks από την C_2 προκύψει η C_1.

Ορισμός 7.1.6 (**Εποχή**)**.** Ως εποχή (epoch), ορίζεται ένα σύνολο από R διαδοχικά slots $S = \{sl_1, ..., sl_R\}$.

Ορισμός 7.1.7 (**Ποσοστό χρημάτων αντιπάλου**)**.** Έστω U_A το σύνολο των χρηστών που ελέγχονται από τον αντίπαλο $\mathcal{A}$. Τότε το ποσοστό των χρημάτων του αντιπάλου στο σύστημα n χρηστών είναι :

$$\alpha = \frac{\sum_{j \in U_A} s_j}{\sum_{i=1}^{n} s_i}$$

Ορισμός 7.1.8 (**Έγκυρη συναλλαγή**)**.** Μια συναλλαγή (tx, σ) θεωρείται έγκυρη εάν:

- Έχει μορφή : "Ο χρήστης pk_1 μεταφέρει στον χρήστη pk_2 ένα ποσό x νομισμάτων με σειριακό νούμερο συναλλαγής sn".
- Η υπογραφή σ που έχει η συναλλαγή επαληθεύεται δωσμένου του αντίστοιχου δημοσίου κλειδιού.
- Δεν υπάρχουν δύο συναλλαγές από τον ίδιο χρήστη με το ίδιο νούμερο συναλλαγής sn.

Ορισμός 7.1.9 (**Έγκυρη αλυσίδα**)**.** Μια αλυσίδα C θεωρείται έγκυρη αλυσίδα blockchain εάν όλες οι συναλλαγές που έχουν προστεθεί στα

block της είναι έγκυρες.

Ορισμός 7.1.10 (**Function maxvalid**)**.** Η συνάρτηση *maxvalid* είναι μια βασική συνάρτηση του πρωτοκόλλου που επιτρέπει στους χρήστες να επιλέξουν μια από τις διαθέσιμες αλυσίδες που έχουν δημιουργηθεί από τους χρήστες. Η συνάρτηση αυτή δέχεται σαν ορίσματα την τρέχουσα αλυσίδα του συστήματος C και το σύνολο των έγκυρων αλυσιδών (valid chains) $\mathbb{C}$. Ο κανόνας σύμφωνα με τον οποίο λειτουργεί είναι ότι ο χρήστης διαλέγει την μεγαλύτερη σε μήκος αλυσίδα από τις $\mathbb{C} \cup \{C\}$.

7.1.3 Δημιουργία επιτροπής

Προκειμένου να παραχθεί ένα καινούριο block και να ενσωματωθεί στην αλυσίδα, είναι αναγκαίο να εκλεχθεί μια ομάδα χρηστών οι οποίοι θα διατελέσουν αυτό το έργο. Οι χρήστες που θα προτείνουν ένα καινούριο block για ένα slot του Ouroboros ονομάζονται *slot leaders* ή αρχηγοί slot. Πριν από την αρχή κάθε εποχής, δημιουργείται η επιτροπή με τους slot leaders των R slot που την αποτελούν.

Η επιλογή των slot leaders δεν είναι τυχαία, αλλά βασίζεται σε μια πιθανότητα επιλογής p_i, που είναι ανάλογη του stake s_i που διαθέτει ο χρήστης U_i. Οι πληροφορίες για τα stake των χρηστών προέρχονται από την κατανομή των χρημάτων $S = (pk_1, s_1), (pk_2, s_2), ...(pk_n, s_n)$. Η πιθανότητα αυτή p_i ορίζεται ως ο λόγος του stake s_i του χρήστη U_i με δημόσιο κλειδί pk_i, προς τα συνολικά χρήματα που κυκλοφορούν στο σύστημα $\sum_{k=1}^{n} s_k$. Μπορεί λοιπόν να προσδιοριστεί μια συνάρτηση $F(.)$ η οποία θα προσομοιώνει την διαδικασία επιλογής slot leader ως εξής :

Ορισμός 7.1.11 (**Συνάρτηση εκλογής slot leader**)**.** Η ντετερμινιστική συνάρτηση $F(.)$, με την βοήθεια μιας κατανομής D που παράγει τυχαιότητα ρ, και της κατανομής των χρημάτων $\mathbb{S} = (pk_1, s_1), (pk_2, s_2), ...(pk_n, s_n)$, για κάθε slot $sl_j \in \{sl_1, ...sl_R\}$ υποδεικνύει έναν χρήστη $U_i \in \{U_1, ..., U_n\}$ να ενεργήσει ως *slot leader* με πιθανότητα $p_i = \frac{s_i}{\sum_{k=1}^{n} s_k}$. Οι εκλογές $F(\mathbb{S}, \rho, sl_j)$ για κάθε $j \in \{1, ..., R\}$ είναι ανεξάρτητες μεταξύ τους.

Follow The Satoshi - FTS

Ο αλγόριθμος Follow The Satoshi (ή FTS) επιτρέπει την επιλογή ενός χρήστη ανάμεσα στους υπόλοιπους ανάλογα με το stake που διαθέτει. Δέχεται σαν είσοδο ένα σπόρο τυχαιότητας, επιλέγει ένα νούμερο από ένα

συγκεκριμένο εύρος και το επιστρέφει. Το νούμερο ανήκει στο διάστημα $[1, \sum_{k=1}^{n} s_n]$, όπου $\sum_{k=1}^{n} s_n$ είναι τα συνολικά νομίσματα των n χρηστών του Ouroboros. Κάθε νούμερο είναι και ένα μοναδικό νόμισμα και οι χρήστες μπορεί να διαθέτουν πολλά νομίσματα. Άρα διαλέγεται τυχαία ένα νούμερο μεταξύ του 1 και του πλήθους των συνολικών νομισμάτων του δικτύου, το οποίο είναι ένα νόμισμα που ανήκει σε κάποιο χρήστη. Ο χρήστης αυτός έχει επιλεχθεί ως slot leader. Η διαδικασία μπορεί να αποδοθεί σχηματικά ως εξής :

Οι πληροφορίες για τα νομίσματα και τις αντιστοιχίες στους χρήστες, αντλούνται από την κατανομή χρημάτων $\mathbb{S} = (pk_1, s_1), (pk_2, s_2), ...(pk_n, s_n)$ που είναι αποθηκευμένη πάνω στο blockchain, και άρα προσβάσιμη σε όλους. Ο αλγόριθμος επίσης είναι διαθέσιμος σε όσους συμμετέχουν ενεργά ή μη στο consensus, συνεπώς η διαδικασία εκλογής σύμφωνα με τον αλγόριθμο Follow The Satoshi είναι δημοσίως επαληθευόμενη. Αυτό σημαίνει ότι οποιοσδήποτε χρήστης και οποιαδήποτε στιγμή μπορεί να διαβάσει το blockchain, να βρει την κατανομή $\mathbb{S}$, και με τον FTS να δει σε ποιον είχε ανατεθεί η παραγωγή κάποιου συγκεκριμένου block.

Ο αλγόριθμος FTS έρχεται σε συμφωνία με το consensus τύπου proof-of-stake, διότι οι χρήστες μέσω του FTS εκλέγονται με βάση το stake που διαθέτουν στο σύστημα, όπως ακριβώς ορίζει ο PoS. Ο κάθε χρήστης που διαθέτει s_i νομίσματα, έχει s_i πιθανά νομίσματα που μπορεί να τον κάνουν slot leader. Άρα ο FTS έχει s_i πιθανές επιλογές για τον χρήστη pk_i, προς τα συνολικά $\sum_{k=1}^{n} s_n$ νομίσματα. Με άλλα λόγια, η πιθανότητα εκλογής του χρήστη είναι ακριβώς $p_i = \frac{s_i}{\sum_{k=1}^{n} s_k}$, όσο περισσότερο stake έχει, τόσο πιο πιθανό είναι να επιλεχθεί από τον FTS.

Το μόνο που μένει να διερευνηθεί, και είναι σημαντικό κομμάτι για την ασφάλεια του πρωτοκόλλου, είναι η παραγωγή της τυχαιότητας. Ο σπόρος που χρησιμοποιείται θα πρέπει να έχει αρκετή εντροπία ώστε να μην μπορεί να επηρεαστεί η διαδικασία εκλογής αρχηγού. Ο τρόπος με τον οποίο παράγεται η τυχαιότητα διερευνάται ενδελεχώς στην συνέχεια. Προς το παρόν σε αυτή την κατάσταση του πρωτοκόλλου θα θεωρηθεί ότι ο ρ προέρχεται από μια κατανομή D.

Εάν υποθέσουμε λοιπόν ότι η κατανομή D μας δίνει την κατάλληλη τυχαιότητα ρ για την εκλογή αρχηγού, μπορούμε να σχεδιάσουμε έναν αλγόριθμο LeaderSelection() :

Algorithm 12: LeaderSelection(ϱ, $\mathbb{S}$, sl)

```
newFunction F;
F ← ∅
a ← 1
for 1 ≤ i ≤ n do
    (pk_i, s_i) ← 𝕊
    for a ≤ s_i do
        F(a) ← pk_i
        a++
h ← H(ρ||sl) mod t
pk ← F(h)
return pk
```

Ο αλγόριθμος δέχεται σαν ορίσματα τα εξής :

- Τυχαιότητα ρ που προέρχεται από μια κατανομή D.
- Κατανομή $\mathbb{S}$ η οποία υπάρχει διαθέσιμη στην αλυσίδα blockchain και περιλαμβάνει τα δημόσια κλειδιά και τα νομίσματα των χρηστών του συστήματος σε αυτή την μορφή : $S = (pk_1, s_1), (pk_2, s_2), ...(pk_n, s_n)$.
- Αριθμός slot στον οποίο επιέγεται ο slot leader.

Στα πρώτα βήματα αρχικοποιούνται οι παράμετροι. Πιο συγκεκριμένα, ορίζεται μια συνάρτηση F και αρχικοποιείται ως κενή. Επίσης αρχικοποιείται μια παράμετρος a με την τιμή 1. Στόχος του αλγορίθμου είναι πρώτα να αντιστοιχίσει τα νομίσματα του συστήματος σε δημόσια κλειδιά με την F, και στην συνέχεια αυτή δεδομένης της τυχαιότητας να επιστρέψει τον slot leader της εποχής για το slot sl. Για αυτόν τον λόγο υπάρχει η παράμετρος a, η οποία αντιπροσωπεύει αυτά τα $\sum_{k=1}^{n} s_n$ νομίσματα. Στην επανάληψη For, για κάθε χρήστη, μετρώνται τα νομίσματα που διαθέτει, και καταγράφονται στην F. Η επανάληψη σταματά όταν ελεγχθεί από την κατανομή $\mathbb{S}$ κάθε ζευγάρι δημοσίου κλειδιού-stake. Το τελευταίο κομμάτι του αλγορίθμου αφορά την χρήστη του σπόρου ρ ώστε να επιλεχθεί ένας τυχαίος αριθμός $h \in [1, \sum_{k=1}^{n} s_n]$, με τον οποίο η F θα επιδείξει το δημόσιο κλειδί του slot leader. Πιο συγκεκριμένα, θεωρούμε μια συνάρτηση κατακερματισμού H που δωσμένης μιας τιμής, επιστρέφει ακέραιους αριθμούς $\in \{1, 2, ..., \sum_{k=1}^{n} s_n\}$. Ο σπόρος ρ και ο αριθμός του slot μπαίνουν μέσα στην συνάρτηση H και επιστρέφουν μια τιμή $h \in \{1, 2, ..., \sum_{k=1}^{n} s_n\}$, που είναι το τυχερό νόμισμα. Δημιουργούμε λοιπόν μια τυχαία τιμή h και βρίσκουμε το δημόσιο κλειδί pk που της αντιστοιχεί μέσω της $F(h)$. Επιστρέφεται το δημόσιο κλειδί pk που είναι ο slot leader για το slot sl.

Παραθέτουμε και έναν αλγόριθμο επαλήθευσης Slot Leader, σε περίπτωση

που κάποιος χρήστης θέλει να ελέγξει ότι όντως ένα δημόσιο κλειδί pk έχει επιλεχθεί για slot leader στο $slot$. Ο αλγόριθμος αυτός ονομάζεται LeaderVerify() :

Algorithm 13: LeaderVerify(ϱ, $\mathbb{S}$, pk, sl)

```
newFunction F;
F ← ∅
a ← 1
for 1 ≤ i ≤ n do
    (pk_i, s_i) ← 𝕊
    for a ≤ s_i do
        F(a) ← pk_i
        a++
h ← H(ρ||sl) mod t
if F(h) == pk then
    return YES
else
    return NO
```

Ο αλγόριθμος δέχεται σαν ορίσματα τα εξής :

- Τυχαιότητα ρ που προέρχεται από μια κατανομή D.
- Κατανομή $\mathbb{S}$ η οποία υπάρχει διαθέσιμη στην αλυσίδα blockchain και περιλαμβάνει τα δημόσια κλειδιά και τα νομίσματα των χρηστών του συστήματος σε αυτή την μορφή : $S = (pk_1, s_1), (pk_2, s_2), ...(pk_n, s_n)$.
- Δημόσιο κλειδί pk που θα ελεγχθεί εάν είναι slot leader σε κάποιο slot.
- Ο αριθμός του slot sl για το οποίο ελέγχεται εάν είναι slot leader το pk.

Στα πρώτα βήματα αρχικοποιούνται οι παράμετροι. Πιο συγκεκριμένα, ορίζεται μια συνάρτηση F και αρχικοποιείται ως κενή. Επίσης αρχικοποιείται μια παράμετρος a με την τιμή 1. Στόχος του αλγορίθμου είναι πρώτα να αντιστοιχίσει τα νομίσματα του συστήματος σε δημόσια κλειδιά με την F, και στην συνέχεια αυτή δεδομένης της τυχαιότητας να επιστρέψει τους slot leader της εποχής. Για αυτόν τον λόγο υπάρχει η παράμετρος a, η οποία αντιπροσωπεύει αυτά τα $\sum_{k=1}^{n} s_n$ νομίσματα. Στην επανάληψη For, για κάθε χρήστη, μετρώνται τα νομίσματα που διαθέτει, και καταγράφονται στην F. Η επανάληψη σταματά όταν ελεγχθεί από την

κατανομή $\mathbb{S}$ κάθε ζευγάρι δημοσίου κλειδιού-stake. Το τελευταίο κομμάτι του αλγορίθμου αφορά την χρήστη του σπόρου ρ ώστε να επιλεχθεί ένας τυχαίος αριθμός $h \in [1, \sum_{k=1}^{n} s_n]$ για το sl, με τον οποίο η F θα επιδείξει ποιό δημόσιο κλειδί είναι slot leader του sl. Πιο συγκεκριμένα, θεωρούμε μια συνάρτηση κατακερματισμού H που δωσμένης μιας τιμής, επιστρέφει ακέραιους αριθμούς $\in \{1, 2, ..., \sum_{k=1}^{n} s_n\}$. Ο σπόρος ρ και ο αριθμός sl μπαίνουν μέσα στην συνάρτηση H και επιστρέφουν μια τιμή $h \in \{1, 2, ..., \sum_{k=1}^{n} s_n\}$, που είναι το τυχερό νόμισμα. Εάν η συνάρτηση F με όρισμα το τυχερό νόμισμα h υποδείξει ότι ανήκει στο δημόσιο κλειδί pk της αρχικοποίησης του αλγορίθμου LeaderVerify(), τότε όντως ο χρήστης με δημόσιο κλειδί pk είναι slot leader και επιστρέφεται η τιμή YES. Εάν η τιμή $F(h)$ είναι διαφορετική της pk, επιστρέφεται NO.

7.1.4 Περιγραφή του πρωτοκόλλου σε μια εποχή

Με βάση το μοντέλο, τους ορισμούς και την συνάρτηση εκλογής slot leader, θα περιγραφτεί το πρωτόκολλο στην πιο απλή μορφή του. Η πιο απλή μορφή είναι η ανάλυση των βημάτων μέσα σε μια εποχή με R slots, όπου η κατανομή των χρημάτων παραμένει σταθερή. Θεωρείται δηλαδή ότι μέσα στην εποχή δεν υπάρχει αλλαγή στο stake distribution, και δεν υπάρχει stake shift. Οι πληροφορίες για την κατανομή των χρημάτων είναι αποθηκευμένες μέσα στο blockchain, στο πρώτο slot της εποχής.

Αρχικοποίηση

Κατα την αρχικοποίηση του βασικού πρωτοκόλλου, ο κάθε χρήστης φροντίζει να αποκτήσει τις απαραίτητες πληροφορίες ώστε να προετοιμαστεί για το στάδιο επέκτασης της αλυσίδας. Πρώτα λαμβάνει από το περιβάλλον το ζευγάρι δημοσίων και ιδιωτικών κλειδιών του (pk, sk). Στην συνέχεια ενημερώνεται για το τρέχων slot που κάνει την αρχικοποίηση. Εάν είναι το πρώτο slot sl_1, λαμβάνει το genesis block $genblock$, την κατανομή των stake $\mathbb{S}_0$, τον σπόρο τυχαιότητας ρ και την συνάρτηση εκλογής leader F και κρατάει ένα τοπικό blockchain $C = B_0$. Αν το τρέχων slot δεν είναι το πρώτο, λαμβάνει την αλυσίδα από το σύστημα και φτιάχνει τοπικό blockchain C . Στην πρώτη περίπτωση ορίζει $st = H(B_0)$, αλλιώς $st = H(head(C))$.

Με λίγα λόγια κατα την αρχικοποίηση προσδιορίζονται τα εξής :

1. Ζευγάρι κλειδιών
2. Τρέχων slot και αντίστοιχες πληροφορίες
3. Τοπικό Blockchain
4. Κατάσταση state

Επέκταση Αλυσίδας

Αφού ο χρήστης έχει δημιουργήσει μια τοπική αλυσίδα και έχει όλες τις απαραίτητες πληροφορίες, μπαίνει στην διαδικασία επέκτασης της αλυσίδας. Πρόκειται για μια επαναλαμβανόμενη διαδικασία που συμβαίνει R φορές, όσα και τα slot της εποχής, και πραγματοποιείται από κάθε χρήστη, είτε είναι slot leader είτε όχι.
Ο χρήστης λαμβάνει τις συναλλαγές $d \in \{0,1\}^*$ που πρόκειται να βάλει σε κάποιο block του blockchain. Στην συνέχεια ενημερώνεται για όλες τις νέες αλυσίδες που υπάρχουν διαθέσιμες, και ορίζει ως νέα αλυσίδα αυτή που προκύπτει από την συνάρτηση $\text{maxvalid}(\mathbb{C}, C)$, εφόσον επαληθεύονται οι υπογραφές των block που περιέχει και οι slot leader επαληθεύονται από τον αλγόριθμο LeaderVerify$(\rho,\ \mathbb{S},\ pk,\ sl)$. Τότε $C' = \text{maxvalid}(\mathbb{C}, C)$ και C' είναι η καινούρια τοπική αλυσίδα, και $st = H(head(C'))$. Στην συνέχεια, στην περίπτωση που ο αλγόριθμος LeaderSelection$(\rho,\ \mathbb{S},\ sl)$ έχει επιστρέψει το δημόσιο κλειδί του, φτιάχνει το $B = (st, d, sl, \sigma)$ όπου d οι συναλλαγές και σ η υπογραφή $\sigma = Sign_{sk}(st, d, sl)$. Δημιουργεί την καινούργια τοπική αλυσίδα $C'' = C'|B$ και την μεταδίδει στο σύστημα. Επίσης ορίζεται $st = H(head(C''))$.

Με λίγα λόγια κατα την επέκταση αλυσίδας έχουμε τα βήματα :

1. Δεδομένα συναλλαγών
2. Συλλογή έγκυρων αλυσιδών
3. Εφαρμογή maxvalid()
4. Ενημέρωση τοπικής αλυσίδας και κατάστασης
5. Έλεγχος LeaderSelection$(\rho,\ \mathbb{S},\ sl)$
6. Για θετική απάντηση :
   - Δημιουργία καινούριου block
   - Δημιουργία καινούριας αλυσίδας με το block
   - Μετάδοση καινούριας αλυσίδας
   - Ενημέρωση τοπικής αλυσίδας και κατάστασης

Παραγωγή συναλλαγών

Σε αυτό το στάδιο του πρωτοκόλλου ο χρήστης μπορεί να παράξει συναλλαγές, σύμφωνα πάντα με ένα υπόδειγμα έγκυρης συναλλαγής. Για αυτόν τον λόγο δημιουργεί και μια υπογραφή μαζί με την συναλλαγή tx, και στην συνέχεια αποστέλνει το ζευγάρι (tx, σ) στο περιβάλλον.

Παρατήρηση 7.1.1. Οι χρήστες του πρωτοκόλλου από υπόθεση, μπορεί να είναι ενεργοί ή και οχι. Αυτό σημαίνει πως οι slot leaders δεν είναι απαραίτητα σε θέση να παράξουν κάποιο block, οπότε υπάρχει περίπτωση κάποιο slot να μείνει κενό. Για λόγους ασφάλειας, εάν δεν παραχθεί κάποιο block, αυτό θεωρείται κίνηση του αντιπάλου $\mathcal{A}$.

7.1.5 Ανάλυση ασφάλειας σε μια εποχή

Η ανάλυση ασφάλειας περιλαμβάνει την ανάλυση των ιδιοτήτων της αλυσίδας σε μια εποχή, όπως ορίζονται στην Περιγραφή μοντέλου, δηλαδή των Common Prefix, Existential Chain Quality και Honest Chain Growth. Προκειμένου να γίνει αυτή η ανάλυση της αλυσίδας, θα μεταφραστεί η ακολουθία των blocks σε μια δυαδική ακολουθία της μορφής $\{0,1\}^n$. Αυτή η ακολουθία ονομάζεται χαρακτηριστική συμβολοσειρά και κάθε αριθμός αντιπροσωπεύει ένα slot, άρα μια συμβολοσειρά μήκους n είναι μια αλυσίδα με n slot. Ο αριθμός 0 αντιπροσωπεύει ένα slot του οποίου ο αρχηγός είναι ειλικρινής χρήστης, ενώ ο αριθμός 1 αντιπροσωπεύει ένα slot του οποίου ο αρχηγός είναι κακόβουλος χρήστης. Παρακάτω δίνεται ένας ορισμός όπως περιγράφεται στο πρωτόκολλο.

Ορισμός 7.1.12 (**Χαρακτηριστική συμβολοσειρά**)**.** Έστω μια αλυσίδα C του συστήματος, με genesis block B_0, και ένας αντίπαλος $\mathcal{A}$. Έστω $S = \{i+1, ..., i+n\}$ μια ακολουθία από slot μήκους $length|S| = n$. Η χαρακτηριστική συμβολοσειρά $\omega \in \{0,1\}^n$ του S είναι τέτοια ώστε $\omega_k = 1$ αν και μόνο αν ο $\mathcal{A}$ ελέγχει τον slot leader του slot $i+k$. Για μια τέτοια συμβολοσειρά $\omega \in \{0,1\}^*$, αν $\omega_i = 1$ τότε το περιεχόμενο του i είναι μη έμπιστο, αλλιώς θεωρείται έμπιστο.

Παρατήρηση 7.1.2. Το πρώτο slot του genesis block θεωρείται πάντα έμπιστο. Τα slot που θεωρούνται κακόβουλα και άρα $\omega_i = 1$, είναι αυτά που ο slot leader τους έχει τεθεί υπο τον έλεγχο του αντιπάλου $\mathcal{A}$ τουλάχιστον μια φορά σε όλη την διάρκεια του πρωτοκόλλου. Ο λόγος που συμβαίνει αυτό είναι προκειμένου να περιοριστούν τα long-range-attacks όπου οι slot leaders στοχοποιούνται και διαφθείρονται αφού έχουν ενεργήσει. Η λογική πίσω από αυτό είναι ότι αφού έχουν επιλεχθεί μια φορά θα υπάρχουν πιθανότητες να ξαναεπιλεχθούν δεδομένου ότι έχουν αρκετό stake ώστε ο αλγόριθμος επιλογής slot leader να τους επέλεξε.

Προκειμένου να κινητοποιηθεί η σκέψη πίσω από την ανάλυση αυτή, θεωρούνται δύο παθητικοί χρήστες της αλυσίδας που ήταν ανενεργοί όταν ξεκίνησε η ακολουθία S των n slot, έβλεπαν την ίδια αλυσίδα C_0 πριν γίνουν ανενεργοί, και όταν γίνονται πάλι ενεργοί στο τέλος των n slot ενημερώνουν την αλυσίδα τους. Σκοπός της απόδειξης του common prefix είναι να υπάρχει πολύ μικρή πιθανότητα οι δύο αυτοί χρήστες να βλέπουν διαφορετικές αλυσίδες C_1 και C_2 αντίστοιχα με κοινό πρόθημα C_0. Η απόδειξη αυτή θα γίνει με την χρήση των χαρακτηριστικών συμβολοσειρών και της έννοιας του fork ή αλλιώς της "διχάλας".

Ορισμός 7.1.13 (**Fork**). Έστω μια χαρακτηριστική συμβολοσειρά $\omega \in \{0,1\}^n$ και $H = \{i \mid \omega_i = 0\}$ το σύνολο των slot με έμπιστο περιεχόμενο. $Fork$ μιας συμβολοσειράς ω ονομάζεται το κατευθυνόμενο δέντρο με ρίζα, $F = (V, E)$ και ετικέτες $\boldsymbol{\ell} : V \to \{0, 1, ..., n\}$ ώστε:

- Κάθε ακμή του δέντρου κατευθύνεται αντίθετα της ρίζας του.
- Η ρίζα του $r \in V$ έχει ετικέτα $\boldsymbol{\ell}(r) = 0$.
- Οι ετικέτες κατα μήκος οποιουδήποτε κατευθυνόμενου μονοπατιού αυξάνονται αυστηρώς.
- Κάθε έμπιστο περιεχόμενο $i \in H$ είναι n ετικέτα ακριβώς ενός κόμβου του δέντρου F.
- Η συνάρτηση $d : H \to \{1, ..., n\}$, όπου $d(i)$ είναι το βάθος του κόμβου υ με ετικέτα $\boldsymbol{\ell}(\upsilon) = i$, αυξάνεται αυστηρώς. (αν $i < j$ τότε $d(i) < d(j)$)

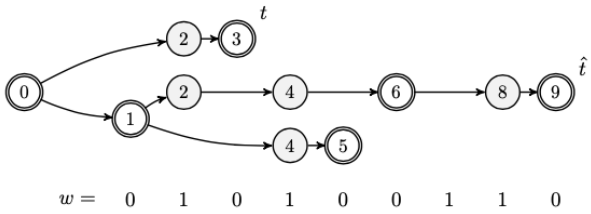
Όταν μια αλυσίδα με συμβολοσειρά ω έχει fork F, τότε συμβολίζουμε $F \vdash \omega$. Ένα fork είναι τετριμένο εάν το F περιλαμβάνει μόνο έναν κόμβο, που είναι n ρίζα.

Ορισμός 7.1.14 (**Tine, depth, height**). Ένα μονοπάτι μιας διακλάδωσης (fork) που ξεκινά από την ρίζα του F ονομάζεται *tine*. Το μήκος ενός *tine* t είναι οι ακμές που έχει το μονοπάτι αυτό και συμβολίζεται ως $length(t)$. Βάθος ενός κόμβου υ ονομάζεται το μήκος του μοναδικού *tine* που καταλήγει σε αυτόν τον κόμβο και συμβολίζεται ως $depth(\upsilon)$. Το ύψος ενός fork ονομάζεται το μεγαλύτερο *tine* που έχει (αυτό δηλαδή με το μεγαλύτερο μήκος).

Η χρήση της ετικέτας $\boldsymbol{\ell}$ μπορεί να χρησιμοποιηθεί και για τα tines, και $\boldsymbol{\ell}(t) = \boldsymbol{\ell}(\upsilon)$, δηλαδή n ετικέτα ενός tine είναι το τελευταίο του block υ. Επιπλέον ορίζουμε για ένα tine t, το tine $t^{\lceil k}$ που είναι αυτό που απομένει εάν αφαιρεθούν οι τελευταίες k ακμές. Εάν το μήκος ενός tine είναι μικρότερο από k, τότε το $t^{\lceil k}$ είναι n ρίζα του F.

Ορισμός 7.1.15 (**Closed forks**). Ένα fork λέγεται *closed fork* εάν κάθε φύλλο του, δηλαδή τα τελευταία blocks σε όλα τα κλαδιά του, είναι slots ειλικρινών χρηστών. Το fork που αποτελείται μόνο από την ρίζα θεωρείται closed fork.

Ορισμός 7.1.16 (**Viable tine**)**.** Έστω ένα fork στο χαρακτηριστικό string $\omega \in \{0,1\}^n$, $F \vdash \omega$, και έστω t ένα tine του F. Τότε το tine λέγεται *viable* εάν για όλα τα $h \leq \boldsymbol{\ell}(t)$ έχουμε $d(h) \leq length(t)$.



Παρατηρούμε ένα παράδειγμα ενός χαρακτηριστικού string $\omega = 010100110$. Οι κόμβοι με διπλό περίγραμμα (δηλαδή οι 0 , 1, 3, 5 , 6 , 9) αντιστοιχίζονται με την τιμή 0 στο ω και ανήκουν σε έμπιστο slot leader. Σε αυτή την περίπτωση, ένας έμπιστος χρήστης προσθέτει το block στην αλυσίδα που βλέπει μεγαλύτερη. Οι κόμβοι με ένα περίγραμμα (δηλαδή οι 2, 4, 8) ελέγχονται από κακόβουλο χρήστη. Σε αυτές τις περιπτώσεις, βλέπουμε ότι ο $\mathcal{A}$ δημιουργεί forks στην αλυσίδα αποπροσανατολίζοντας τους ειλικρινείς χρήστες. Στην περίπτωση του κόμβου 2, ο $\mathcal{A}$ προσθέτει το block 2 μετά το 1, και αποφασίζει να δημιουργήσει και μια καινούρια αλυσίδα. Ο έμπιστος χρήστης του κόμβου 3 αυξάνει την καινούρια αλυσίδα πιθανώς επειδή ο $\mathcal{A}$ φρόντισε να λάβει μόνο αυτή. Παρόμοια κατάσταση συμβαίνει με τον κόμβο 4. Ο κόμβος 7 δεν υπάρχει στην αναπαράσταση αλλά υπάρχει στο ω. Αυτό που έχει συμβεί είναι ότι ο slot leader του κόμβου 7 επέλεξε να μην παράξει block, το οποίο επίσης θεωρείται κακόβουλη ενέργεια.

Με βάση τους παραπάνω ορισμούς το ω έχει fork με 3 tines. Το πρώτο tine είναι το 0-2-3, το δεύτερο είναι το 0-1-2-4-6-8-9 και το τρίτο tine είναι το 0-1-4-5. Καθένα από αυτά έχει μήκος 2, 6 και 3 ανίστοιχα. Κάθε φύλλο του fork αυτού (3,9,5) έχει σημειωθεί με διπλό περίγραμμα και άρα ανήκει σε ειλικρινή χρήστη, δηλαδή το ω περιγράφει ένα closed fork.

Ιδιότητες blockchain αναδιατυπωμένες βάσει της θεωρίας των forks :

- **Common prefix** (cp: $k \epsilon \mathbb{N}$)
  Μια χαρακτηριστική συμβολοσειρά ω λέμε ότι ικανοποιεί $k - cp$, εάν για κάθε $F \vdash \omega$ και κάθε ζευγάρι από viable tines t_1 και t_2 με $\boldsymbol{\ell}(t_1) \leq \boldsymbol{\ell}(t_2)$, το $t^{\lceil k}$ είναι πρόθημα τουt_2 . Ή αλλιώς $length(t_1) - length(t_1 \cap t_2) \leq k$, όπου το $t_1 \cap t_2$ είναι το κοινό πρόθημα των δύο tines.

- **Honest-Bounded Chain Growth** (hcg: $\tau \epsilon [0,1]$, $s \epsilon \mathbb{N}$)
  Μια χαρακτηριστική συμβολοσειρά ω λέμε ότι ικανοποιεί hcg με

παραμέτρους τ και s, αν για κάθε fork $F \vdash \omega$, κάθε viable tine t του F και κάθε ειλικρινής κόμβος υ του t ώστε $\boldsymbol{\ell}(\upsilon) + s \leq \boldsymbol{\ell}(t)$, το μονοπάτι $t(\boldsymbol{\ell}(\upsilon), \boldsymbol{\ell}(t)]$ περιέχει τουλάχιστον τs κόμβους .

- **Chain Growth** (cg: $\tau \epsilon [0,1]$, $s \epsilon \mathbb{N}$)
  Μια χαρακτηριστική συμβολοσειρά ω λέμε ότι ικανοποιεί $(\tau, s) - cg$, αν για κάθε fork $F \vdash \omega$, κάθε viable tine t του F, οποιοδήποτε τμήμα του t με s slots έχει τουλάχιστον τs κόμβους .

- **Existential Chain Quality** ($\exists$ cq: $s \epsilon \mathbb{N}$)
  Μια χαρακτηριστική συμβολοσειρά ω λέμε ότι ικανοποιεί $s - \exists cq$, εάν για κάθε fork $F \vdash \omega$ και κάθε viable tine t του F, οποιοδήποτε τμήμα του t με s slots έχει τουλάχιστον έναν ειλικρινή κόμβο.

Μαθηματικά Εργαλεία

Για να γίνουν οι αποδείξεις των παραπάνω ιδιοτήτων, χρειάζονται κάποια μαθηματικά εργαλεία που παρουσιάζονται στην συνέχεια.

Θεώρημα 7.1.1 (**Φράγμα Chernoff–Hoeffding**). *Έστω $X_1, ..., X_T$ ανεξάρτητες τυχαίες μεταβλητές με $Q_i \in [0,1]$. Έστω $Q = \sum_{i=1}^{T} X_i$ και $\mu = \mathbb{E}[Q]$. Τότε για κάθε $\delta \geq 0$ ισχύει :*

$$Pr[X \geq (1+\delta)\mu] \leq e^{-\frac{\delta^2}{2+\delta}\mu} \quad \textit{και} \quad Pr[X \leq (1-\delta)\mu] \leq e^{-\frac{\delta^2}{2}\mu}$$

Και επιπλέον για κάθε $\Lambda > 0$:

$$Pr[X \geq \mu + \Lambda] \leq e^{-\frac{2\Lambda^2}{T}} \quad \textit{και} \quad Pr[X \leq \mu - \Lambda] \leq e^{-\frac{2\Lambda^2}{T}}$$

Λήμμα 7.1.1. Έστω $X = (X_1, ..., X_n)$ μια οικογένεια τυχαίων μεταβλητών με τιμές στο {0,1} με ιδιότητα :

$$\forall i > 0, \quad \mathbb{E}[X_i | X_1, ..., X_{i-1}] \leq p$$

Έστω $B = (B_1, ..., B_n)$ μια οικογένεια ανεξάρτητων τυχαίων μεταβλητών με τιμές στο {0,1} για τις οποίες $\mathbb{E}[B_i = 1] = p$. Τότε $X \prec B$.

Chain Quality

Λήμμα 7.1.2. Έστω ένα χαρακτηριστικό string $\omega = \omega_1 ... \omega_L$ με $\omega \in \{0,1\}^L$ ώστε σε κάθε ω_i να ανατίθεται ανεξάρτητα η τιμή 1 με πιθανότητα $\frac{1}{2} - \varepsilon$ για κάποιο $\varepsilon \in (0, \frac{1}{2})$. Τότε για $s > 0$,

$$Pr[\text{η } \omega \text{ να μην ικανοποιεί } s - \exists cq] \leq (\frac{\varepsilon^{-2}+3}{2})\ L\ exp(-2e^2 s)$$

Chain Growth

Λήμμα 7.1.3. Έστω ένα χαρακτηριστικό string ω που ικανοποιεί την $\exists cq$ με παράμετρο $s_{\exists cq}$ και hcg με παραμέτρους τ_{hcg} s_{hcg}. Τότε ικανοποιεί cg με παραμέτρους :

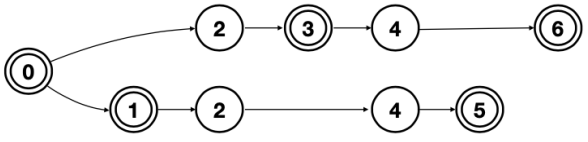
$$s = 2s_{\exists cq} + s_{hcg} \quad \text{και} \quad \tau = \tau_{hcg} \cdot (\frac{s_{hcg}}{2s_{\exists cq} + s_{hcg}})$$

Θεωρώντας ότι $s_{hcg} \geq 2s_{\exists cq}$, ικανοποιείται η cg με παράμετρο $\tau \geq \frac{\tau_{hcg}}{2}$.

Common Prefix

Ορισμός 7.1.17 (**Flat fork**). Έστω t_1 και t_2 δύο tines ενός fork F, τότε συμβολίζουμε με $t_1\ t_2$ εάν μοιράζονται μια ακμή. Ένα fork ονομάζεται $flat$ εάν έχει δύο tines $t_1 \nearrow t_2$ μήκους όσο το ύψος του fork F. Ένα χαρακτηριστικό string $\omega \in \{0,1\}^*$ είναι forkable εαν υπάρχει *flat fork* $F \vdash \omega$.

Άρα όταν αναφερόμαστε σε ένα forkable string αναφερόμαστε σε ένα fork που έχει συμβεί στην αλυσίδα το οποίο έχει δύο μονοπάτια tines τα οποία έχουν το ίδιο μήκος και καμία κοινή ακμή. Αυτό είναι το πιο επικίνδυνο fork που μπορεί να συμβεί διότι όπως έχει αναφερθεί οι χρήστες θα πρέπει να μπορούν να βρουν την μεγαλύτερη αλυσίδα για να την επεκτείνουν. Εάν υπάρχουν δύο τέτοιες αλυσίδες χωρίς την ιδιότητα του προθήματος το πρωτόκολλο έχει πρόβλημα. Αποδεικνύεται ότι η πιθανότητα να συμβεί αυτό είναι πολύ μικρή :



Σχήμα 7.1: Παράδειγμα forkable $\omega = 0010100$

Θεώρημα 7.1.2. *Αν ω είναι μια χαρακτηριστική συμβολοσειρά με $\omega \in \{0,1\}^*$, με πιθανότητα για κάθε $\omega_i = 1$: $\frac{1-\varepsilon}{2}$ με $\varepsilon \in (0,1)$. Έτσι έχουμε:*

$$Pr[\omega \text{ να είναι } forkable] = 2^{-\Omega(\sqrt{n})}$$

Για να μπορέσουν να μελετηθούν τα forkable strings θα πρέπει να μπορούν να συγκριθούν τα διάφορα tines μεταξύ τους, όχι μόνο ως προς το μήκος, αλλά και ως προς το πλεονέκτημα του αντιπάλου στα δύο tines, ώστε να ερευνηθούν οι πιθανές κινήσεις του. Για τους λόγους αυτούς ορίζουμε τις ποσότητες gap,reserve,reach και margin.

Ορισμός 7.1.18 (**Gap,Reserve,Reach**)**.** Έστω ένα closed fork και ενα tine $\hat{t}$ με το μεγαλύτερο μήκος από τα υπόλοιπα στο F. Τότε για οποιοδήποτε άλλο tine t διαφορετικό από το $\hat{t}$:
Ορίζουμε ως gap(t) την διαφορά του μήκους από το $\hat{t}$ στο t. Δηλαδή :

$$\text{gap(t)} = length(\hat{t}) - length(t)$$

Ορίζουμε ως reserve(t) τον αριθμό των μη έμπιστων slot που υπάρχουν στο ω μετά το τελευταίο block του $t = (r, \upsilon_1, ..., \upsilon_k)$. Δηλαδή :

$$\text{reserve(t)} = \mid i \mid \omega_i = 1 \text{ και } i > \boldsymbol{\ell}(\upsilon_k)\mid$$

Τέλος ορίζουμε ως reach(t) την διαφορά μεταξύ των δύο ποσοτήτων gap(t) και reserve(t) :

$$\text{reach(t)} = reserve(t) - gap(t)$$

Ορισμός 7.1.19 (**Margin**)**.** Έστω ένα closed fork με μέγιστη ποσότητα reach(t) όλων των συγκρίσεων tines του F, $\rho(F)$, δηλαδή :

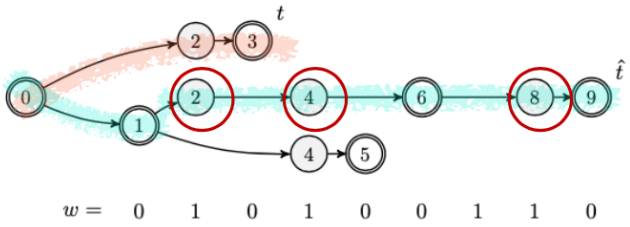
$$\rho(F) = max_t reach(t)$$

Το margin του fork F ορίζεται ως το προτελευταίο reach δύο tine χωρίς κοινές ακμές:

$$margin(F) = \mu(F) = max_{t_1 \not\sim t_2}(minreach(t_1), reach(t_2))$$

Παρατήρηση 7.1.3. Μπορούν εύκολα να κατανοηθούν οι έννοιες Gap και Reserve για ένα tine, αλλά τι είναι η διαφορά τους Reach ; Ουσιαστικά το reach(t) πρόκειται για το πλεονέκτημα του αντιπάλου να προσθέσει adversarial blocks στο tine t ώστε να φτάσει το μακρύτερο tine $\hat{t}$ και να δημιουργήσει δύο tines με το ίδιο μήκος που θα είναι και τα μεγαλύτερα στο fork. Άρα θα έχει μετατρέψει το ω σε ένα forkable string. Αυτό συμβαίνει εάν το προτελευταίο μακρύτερο tine συγκριθεί με το μακρύτερο και υπάρχει δυνατότητα reach ≥ 0, δηλαδή εάν margin ≥ 0.

Για καλύτερη κατανόηση των παραπάνω εννοιών θα καταφύγουμε στο παράδειγμα που δώθηκε προηγουμένως :



Το string ω = 010100110 έχει 3 tines $t, \hat{t}, t'$, εκ των οποίων αυτά που μοιράζονται τον κοινό κόμβο 1 είναι $\hat{t} \sim t'$, ενώ τα υπόλοιπα ανα δύο δεν μοιράζονται κοινές ακμές. Δηλαδή $\hat{t} \nsim t$ και $t' \nsim t$. Θα επικεντρωθούμε λοιπόν στις περιπτώσεις που δεν υπάρχουν κοινές ακμές. Από αυτά τα ζευγάρια μόνο το πρώτο μας ενδιαφέρει καθώς περιλαμβάνει το tine με το μεγαλύτερο μήκος $\hat{t}$. Για αυτά τα δύο ισχύει :

- gap(t) = length($\hat{t}$) - length(t) = 6 - 2 = 4
- reserve(t) = {2,4,8} = 3 adversarial slots
- reach(t)= 4 - 3 = 1

Πρόταση 7.1.1. *Μια χαρακτηριστική συμβολοσειρά ω είναι forkable αν και μόνο αν υπάρχει ένα closed fork $F \vdash \omega$ για το οποίο margin(F) ≥ 0.*

Απόδειξη

Ευθύ: για ένα forkable string $\omega \Rightarrow$έχει closed fork $F \vdash \omega$ και margin(F)$\geq$ 0.

- Εάν η χαρακτηριστική συμβολοσειρά ω αποτελείται από κακόβουλα slots, δηλαδή $\forall \omega_i \in \omega, i \neq 0$, τότε το closed fork F που θα μελετηθεί θα είναι ένα τετριμένο fork με έναν κόμβο, την ρίζα $\omega_0 = 0$ που είναι ειλικρινής εξ' ορισμού. Επίσης, ως δέντρο ενός κόμβου, η διαφορά μήκους gap(t) είναι 0 διότι δεν υπάρχουν άλλοι κόμβοι, και η ποσότητα reserve(t) επίσης είναι 0. Άρα reach(t) = reserve(t) - gap(t) = 0 και margin(F)= max{reach(t)} = 0 και ισχύει το margin(F) $\geq$ 0.

- Έστω ένα forkable string ω, με τουλάχιστον ένα ειλικρινές slot, εκτός της ρίζας ω_0, με $\hat{i}$ η μεγαλύτερη ετικέτα ειλικρινούς slot στο ω. Έστω το closed fork $\bar{F}$, που προκύπτει εαν αφαιρέσουμε τα μη ειλικρινά slots μετά το τελευταίο έμπιστο στο F. Τότε το tine $\hat{t}$ που περιέχει το $\hat{i}$ θα είναι το μεγαλύτερο στο $\bar{F}$. Επειδή το ω είναι forkable, θα έχει ένα flat fork και άρα δύο tines με $t_1 \nsim t_2$, μήκους όσο το ύψος του F. Άρα τα t_1, t_2 έχουν μήκος τουλάχιστον όσο το $\hat{t}$. Οπότε αν συγκρίνουμε το $\hat{t}$ με οποιοδήποτε από τα προθήματα των t_1, t_2 που

ανήκουν στο $\bar{F}$, θα συνειδητοποιήσουμε ότι gap(t)=0 σε όλες τις περιπτώσεις αφού έχουν το ίδιο μήκος. Άρα reach(t) = reserve(t) - gap(t) = reserve(t) ≥ 0. Δηλαδή reach(t)$\geq 0 \implies margin(\bar{F}) \geq 0$.

Αντίστροφο: για ένα closed fork $F \vdash \omega$ με margin$(F) \geq 0 \Rightarrow$ το string ω είναι forkable. Έστω ότι το ω έχει ένα closed fork F με margin$(F) \geq 0$.

Αφού το margin(F) είναι μη αρνητικό, σημαίνει ότι υπάρχουν δύο tines χωρίς κοινές ακμές $t_1 \nsim t_2$ με reach$(t_i) \geq 0$ για $i = 1, 2$. Το F επειδή είναι closed τελειώνει σε ειλικρινές slot. Μπορούμε να προσθέσουμε gap(t_i) adversarial slots στο tine $t_i, i = 1, 2$ που έχει το μικρότερο μήκος, ανάλογα με το reserve(t_i) ώστε τα μονοπάτια που θα προκύψουν να έχουν ίδιο μήκος. Τότε θα έχει δημιουργηθεί ένα flat fork στο ω και άρα το ω είναι forkable.

□

Στόχος είναι να δειχθεί ότι η πιθανότητα να προκύψουν forkable συμβολοσειρές είναι πολύ μικρή. Για αυτόν τον λόγο μελετάται η συμπεριφορά των reach και margin, στην περίπτωση της συμβολοσειράς με closed fork F και margin$(F) \geq 0$. Όπως φαίνεται παρακάτω οι ποσότητες $(\rho(\omega), \mu(\omega))$ εκτελούν τυχαίο περίπατο δύο διαστάσεων. Το ζευγάρι αυτό ονομάζεται $\mathbf{m}(\omega)$. Σε κάθε fork η ποσότητα $\rho(\omega)$ είναι πάντα μη αρνητική εξ'ορισμού ενώ το $\mu(\omega)$ μπορεί να πάρει και αρνητικές τιμές.

Λήμμα 7.1.4. Έστω ένα χαρακτηριστικό string ω με closed fork F και margin$(F) \geq 0$, Για τις ποσότητες $\rho(\omega) = max\ \rho(F)$ και $\mu(\omega) = max\ \mu(F)$ μιας χαρακτηριστικής συμβολοσειράς ω, με $\mathbf{m}(\omega) = (\rho(\omega), \mu(\omega))$ έχουμε:

$$\mathbf{m}(\varepsilon) = (0, 0)$$

$$\mathbf{m}(\omega_1) = (\rho(\omega) + 1, \mu(\omega) + 1)$$

$$\mathbf{m}(\omega_0) = \begin{cases} (\rho(\omega) + 1, 0), & \text{για } \rho(\omega) > \mu(\omega) = 0 \\ (0, \mu(\omega) - 1), & \text{για } \rho(\omega) = 0 \\ (\rho(\omega) - 1, \mu(\omega) - 1), & \text{αλλιώς} \end{cases}$$

Απόδειξη Θεωρήματος 4.1.2

Η πιθανότητα να ανατεθεί ένα slot είτε από ειλικρινή είτε από κακόβουλο χρήστη είναι ανεξάρτητη και ίση με $\frac{1+\varepsilon}{2}$ με $\varepsilon \in (0, 1)$. Δηλαδή :

$$Pr[\omega_i = 0 = \frac{1 + \varepsilon}{2} = 1 - Pr[\omega_i = 1]$$

Άρα το ω είναι διωνυμική τυχαία μεταβλητή. Έχουμε λοιπόν τις τυχαίες μεταβλητές :

$$R_t = \rho(\omega_1, ..., \omega_t) \text{ και } M_t = \mu(\omega_1, ..., \omega_t)$$

Από το παραπάνω λήμμα έχουμε :

$$R_t > 0 \Rightarrow \begin{cases} R_{t+1} = R_t + 1, & \text{για } \omega_{t+1} = 1 \\ R_{t+1} = R_t - 1, & \text{για } \omega_{t+1} = 0 \end{cases}$$

$$M_t < 0 \Rightarrow \begin{cases} M_{t+1} = M_t + 1, & \text{για } \omega_{t+1} = 1 \\ M_{t+1} = M_t - 1, & \text{για } \omega_{t+1} = 0 \end{cases}$$

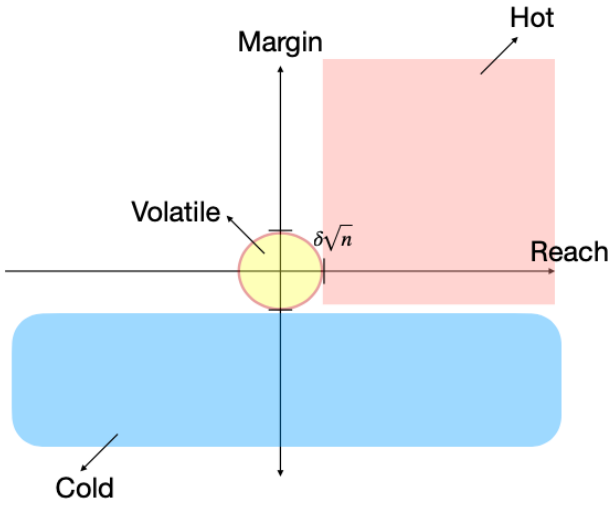
$$R_t = 0 \Rightarrow \begin{cases} R_{t+1} = 1, & \text{για } \omega_{t+1} = 1 \\ R_{t+1} = 0, & \text{για } \omega_{t+1} = 0 \\ M_{t+1} < 0, & \text{για } \omega_t = 0 \end{cases}$$

Δηλαδή όταν το reach είναι θετικό, εκτελεί απλό τυχαίο περίπατο, όταν το margin είναι αρνητικό επίσης εκτελεί απλό τυχαίο περίπατο.
Χωρίζουμε το χαρακτηριστικό string σε $\sqrt{n}$ κομμάτια, άρα θεωρούμε πως το σύστημα κινείται σε χρόνο από το 0 μέχρι το $\sqrt{n}$. Έστω μικρή σταθερά $\delta << \varepsilon$. Τότε διακρίνουμε τις καταστάσεις $\delta\sqrt{n}$ λόγω του φράγματος που Chernoff για τους απλούς τυχαίους περιπάτους :

Ορισμός 7.1.20 (**Φράγμα Chernoff**)**.** Έστω T βήματα ενός απλού τυχαίου περιπάτου μεταβλητής Y_t όταν βρίσκεται στην κατάσταση 0. Τότε η τελική τιμή είναι συγκεντρωμένη γύρω από την τιμή $-\epsilon T$. Ειδικότερα, $\mathbb{E}[Y_T] = -\epsilon T$ και η πιθανότητα ο τυχαίος περίπατος στο τελευταίο βήμα T να έχει τιμή μεγαλύτερη από το μισό της μέσης τιμής είναι :

$$Pr[Y_T > -\frac{\epsilon T}{2}] = 2^{-\Omega(T)}$$

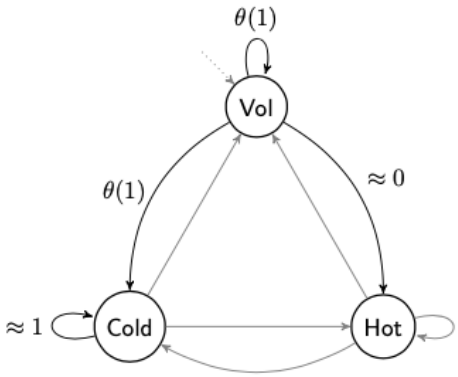
Επειδή η τιμή $\mathbb{E}[Y_T] = -\epsilon T$ της τυχαίας μεταβλητής στο βήμα T είναι κοντά στο $-\epsilon T$, τότε και $\mathbb{E}[R_{\sqrt{n}}] = -\delta\sqrt{n}$ και $\mathbb{E}[M_{\sqrt{n}}] = -\delta\sqrt{n}$ για τις τυχαίες μεταβλητές reach και margin. Έχουμε λοιπόν :



Έχουμε λοιπόν τις εξής περιπτώσεις που φαίνονται στο παραπάνω διάγραμμα :

- **Hot** : $R_{(t)} \geq \delta\sqrt{n}$ και $M_{(t)} \geq -\delta\sqrt{n}$
- **Volatile** : $-\delta\sqrt{n} \leq M_{(t)} \leq R_{(t)} < \delta\sqrt{n}$
- **Cold** : $M_{(t)} < -\delta\sqrt{n}$

Για να αποδειχθεί το common prefix θέλουμε να δείξουμε ότι η πιθανότητα να υπάρχουν forkable strings είναι πολύ μικρή, δηλαδή ότι η πιθανότητα $margin(\omega) \geq 0$ είναι πολύ μικρή. Με την ανάλυση που έγινε, αυτο μεταφράζεται στην κατάσταση **Hot** όπου $M_{(t)} \geq -\delta\sqrt{n}$. Στο πρωτόκολλο παρουσιάζεται μια απεικόνιση μετάβασης των καταστάσεων **Hot,Volatile,Cold** προκειμένου να ελεγχθούν οι πιθανότητες παραμονής ή οχι στις τρεις καταστάσεις.



Σχήμα 7.2: Η αλληλεπίδραση των καταστάσεων **Hot,Volatile,Cold** [1]

Αυτό μεταφράζεται ως :

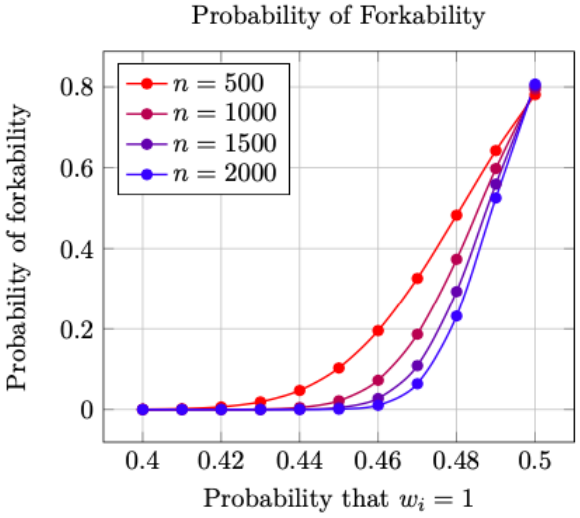
$$Pr[Cold_{t+1}|Cold_t] \geq 1 - 2^{-\Omega(\sqrt{n})}$$

$$Pr[Cold_{t+1}|Vol_t] \geq \Omega(\varepsilon)$$

$$Pr[Hot_{t+1}|Vol_t] \leq 2^{-\Omega(\sqrt{n})}$$

Δηλαδή η πιθανότητα το σύστημα να βρίσκεται στην κατάσταση **Cold**, και παραμένει σε αυτήν είναι σχεδόν 1. Εάν το margin πάρει τιμή μικρότερη από $-\delta\sqrt{n}$, τότε το σύστημα δεν υπάρχει περίπτωση να έχει forkable strings. Η δεύτερη περίπτωση περιγράφει την πιθανότητα, το σύστημα από την κατάσταση **Volatile** να μεταπηδήσει στην κατάσταση **Cold**. Αυτή η πιθανότητα είναι ίση με $\varepsilon \in (0, 1)$ και είναι σταθερή. Τέλος η επικίνδυνη πιθανότητα, εάν το σύστημα βρίσκεται στην κατάσταση **Volatile**, να μεταβεί στην κατάσταση **Cold** είναι πολύ μικρότερη από $2^{-\Omega(\sqrt{n})}$. Δηλαδή η πιθανότητα να υπάρξει forkable string ω είναι σχεδόν μηδενική. Το συμπέρασμα είναι ότι με την πάροδο του χρόνου το σύστημα θα καταλήξει στην κατάσταση **Cold** και άρα δύο tines (αλυσίδες με κοινή αρχή χωρίς κοινές ακμές) μέγιστου μήκους θα έχουν αναγκαστικά common prefix. □

Προς γραφική απόδειξη των ανωτέρω, έχει δημιουγηθεί ένα γράφημα στο πρωτόκολλο [1], το οποίο δείχνει την πιθανότητα ένα string ω να είναι forkable, σε σχέση με την πιθανότητα ανεξάρτητης ανάθεσης των τιμών $\omega_i = 1$ στο διάστημα $\{0.4, 0.41, ..., 0.5\}$ για χαρακτηριστικές συμβολοσειρές μήκους 500,1000,1500 και 2000 αντίστοιχα.



Παρατηρείται στο διάγραμμα πως για πιθανότητες {0.4, 0.41, 0.42, 0.43} ο αντίπαλος $\mathcal{A}$ να έχει υπο τον έλεγχο του κάποιο slot, σε οποιοδήποτε μήκος του ω, η πιθανότητα για forkability είναι σχεδόν μηδενική. Για τις πιθανότητες {0.43, 0.44, 0.45} το ενδεχόμενο η ω να είναι forkable παραμένει μηδενικό για strings μήκους 1000,1500,2000, ενώ για συμβολοσειρά μήκους 500, η πιθανότητα για forkability αυξάνεται εκθετικά μέχρι το 0.2. Για τις υπόλοιπες ανεξάρτητες πιθανότητες $\omega_i = 1$ μέχρι το 0.5, για οποιαδήποτε συμβολοσειρά, η πιθανότητα για ω forkable αυξάνεται γραμμικά μέχρι το 0.8 .

Συνεπώς, προτιμούνται μεγάλες συμβολοσειρές μήκους τουλάχιστον 1000, και πιθανότητα ανάθεσης slot στον $\mathcal{A}$ μέχρι 45%, ώστε η υπόθεση χαρακτηριστικής συμβολοσειράς να είναι forkable να έχει μηδενική πιθανότητα να συμβεί.

7.1.6 Παραγωγή τυχαιότητας στο δυναμικό πρωτόκολλο

Στο δυναμικό πρωτόκολλο, πλέον η κατανομή των χρημάτων αλλάζει σε κάθε εποχή, και εκλέγονται νέοι slot leaders. Για την εκλογή τους αυτή θα πρέπει να υπάρχει μια τυχαιότητα ρ, η οποία βοηθάει τον αλγόριθμο επιλογής αρχηγού να επιλέξει τυχαία τους slot leaders της εποχής. Κατα την περιγραφή της διαδικασίας επιλογής, υποθέσαμε πως η τυχαιότητα παράγεται από μια κατανομή και υπάρχει στο genesis block.

Προκειμένου το πρωτόκολλο να είναι ασφαλές από επιθέσεις, θα πρέπει η τυχαιότητα - randomness ρ να ανανεώνεται σε κάθε εποχή τουλάχιστον. Η ιδέα πίσω από αυτό είναι ότι εάν ο αντίπαλος $\mathcal{A}$ γνωρίζει την τυχαιότητα, θα γνωρίζει και τα νομίσματα που επιλέγονται από τον αλγόριθμο εκλογής των αρχηγών,αφού είναι δημόσιος. Άρα θα μπορεί να αγοράσει τα νομίσματα αυτά πριν να μάθουν οι αντίστοιχοι χρήστες ότι είναι το τυχερό τους νόμισμα, και εν τέλει να είναι αυτός που θα παράξει το block. Για να μην συμβεί κάτι τέτοιο, χρησιμοποιείται η κατανομή χρημάτων, και άρα οι χρήστες της προηγούμενης εποχής, με την καινούρια τυχαιότητα. Οι χρήστες από την προηγούμενη εποχή δεν είναι δυνατό να γνωρίζουν την τυχαιότητα της τρέχουσας.

Για την παραγωγή της τυχαιότητας σε κάθε εποχή, το πρωτόκολλο δεν μπορεί να εμπιστευτεί κάποιο εξωτερικό μηχανισμό ή κάποια κατανομή που θα δώσει απλά μια τιμή σπόρου ρ. Για τον λόγο αυτό χρησιμοποιείται ένα πρωτόκολλο που ονομάζεται *Guaranteed Output Delivery Coin Tossing* ή αλλιώς *G. O. D. Coin Tossing.*

Κρυπτογραφικά Εργαλεία

Το πρώτο εργαλείο που χρησιμοποιείται είναι τα **commitments** ή αλλιώς *σχήματα δέσμευσης*. Τα commitments είναι ένα πρωτόκολλο που πραγματοποιείται ανάμεσα σε δύο διαφορετικές πλευρές, τον *Αποστολέα* και τον *Παραλήπτη*, και επιτρέπει στον *Αποστολέα* να δεσμευτεί ότι έχει στείλει μια συγκεκριμένη τιμή. Στόχος είναι ο *Αποστολέας* να πείσει τον *Παραλήπτη* για την τιμή αυτή, χωρίς να την αποκαλύψει. Τα σχήματα αυτά αποτελούνται από δύο φάσεις. Η πρώτη φάση ονομάζεται *commit* ή αλλιώς *δέσμευση*, στην οποία ο *Αποστολέας* στέλνει την τιμή κρυμμένη με τέτοιο τρόπο που ο *Παραλήπτης* δεν μπορεί να την μάθει. Στην συνέχεια στην δεύτερη φάση, ο *Αποστολέας* στέλνει ένα "κλειδί" στον *Παραλήπτη* , που θα αποκρύψει την κρυμμένη τιμή που δεσμεύτηκε ο *Αποστολέας*. Εάν αυτή η τιμή που βρήκε ο *Παραλήπτης* από την απόκρυψη με την χρήση του κλειδιού είναι ίδια με την τιμή στην οποία δεσμεύτηκε ο *Αποστολέας*, το πρωτόκολλο τελειώνει εδώ επιτυχώς.

Μια χρήσιμη εφαρμογή των σχημάτων δέσμευσης είναι το **coin tossing protocol**. Πρόκειται για μια ιδέα που παρουσιάστηκε από τον Manuel Blum το 1981 στην δημοσίευση Coin Flipping By Telephone [20]. Οι δύο πλευρές *Αποστολέας*, *Παραλήπτης* καλούνται να συμφωνήσουν σε μια κοινή τιμή, με την ρίψη ενός υποτιθέμενου νομίσματος. Έστω ότι οι δύο πλευρές ονομάζονται A και Π. Το πρωτόκολλο πάει ως εξής :

1. Ο A διαλέγει μια τυχαία τιμή από το $\{0,1\}^\lambda$. Έστω η τιμή αυτή m.

2. Ο A στέλνει στον Π μια δέσμευση για την τιμή m, για παράδειγμα την $ENC_{pk}(\rho, m)$ με σχήμα κρυπτογράφησης δημοσίου κλειδιού.

3. Στην συνέχεια, ο Π στέλνει στον A μια τιμή που έχει διαλέξει επίσης τυχαία από το $\{0,1\}^\lambda$. Έστω η τιμή αυτή m'.

4. Ο A στέλνει το δημόσιο κλειδί της κρυπτογράφησης, την τυχαιότητα ώστε ο Π να υπολογίσει την τιμή m.

5. Ταυτόχρονα, και ο A και ο Π υπολογίζουν την τιμή $\hat{m} = m \oplus m'$.

Παρατηρούμε πως με αυτόν τον τρόπο, και οι δύο μπορούν να συμφωνήσουν εν τέλει σε μια κοινή τιμή, εφόσον θα έχουν και οι δύο τις ίδιες τιμές m, m'. Αυτές οι τιμές δεν μπορούν να είναι διαφορετικές, διότι έχουν παραχθεί μέσω ενός σχήματος δέσμευσης που επιτρέπει σε οποιαδήποτε από τις δύο πλευρές A,Π να καταλάβει εάν κάποιος δεν έχει στείλει την τιμή στην οποία έχει δεσμευτεί. Το πρόβλημα είναι ότι ο χρήστης A έχει ένα προβάδισμα διότι γνωρίζει τις τιμές m, m' πριν από τον χρήστη Π. Πιο συγκεκριμένα, ήδη στο τρίτο βήμα της παραπάνω διαδικασίας ο A μπορεί να υπολογίσει το $\hat{m}$ ενώ ο Π οχι. Άρα αν ο A είναι κακόβουλος μπορεί να μην πραγματοποιήσει το τέταρτο βήμα και να αφήσει τον Π στην άγνοια, και άρα η διαδικασία να μην είναι δίκαιη. Στο [26], αναφέρονται σε διάφορες δημοσιεύσεις οι προσπάθειες για δημιουργία επιτυχούς και δίκαιου

πρωτοκόλλου coin flipping μεταξύ δύο πλευρών, αλλά αυτό φαίνεται ότι είναι αδύνατο να συμβεί σε μικρό αριθμό βημάτων.

Ο σκοπός είναι να φτιαχτεί ένα σχήμα **coin tossing** για το πρωτόκολλο του Ouroboros ώστε όλοι οι χρήστες να μπορούν να λάβουν και να συμφωνήσουν σε μια τιμή τυχαιότητας ρ. Λόγω του ότι θέλουμε αυτό το σχήμα να εφαρμοστεί σε ένα πρωτόκολλο blockchain, είναι σημαντικό να υλοποιείται σε λίγα βήματα ώστε οι χρήστες να μπορούν να βρίσκουν γρήγορα την τυχαιότητα, να εκλέγουν γρήγορα slot leaders και άρα να αυξάνουν γρήγορα την αλυσίδα. Επίσης είναι αναγκαίο το πρωτόκολλο να μπορεί να εφαρμοστεί για συμφωνία πολλαπλών χρηστών και οχι μόνο μεταξύ δύο πλευρών. Αυτό μπορεί να συμβεί με την χρήση σχημάτων διανομής απορρήτων (secret sharing).

Σε ένα σχήμα διανομής απορρήτων ή **secret sharing**, στόχος είναι να διαμοιραστεί ένα μυστικό s σε πολλαπλούς χρήστες. Για να μοιραστεί το μυστικό s, δίνεται ένα ξεχωριστό μέρος της πληροφορίας του s από τον *διανομέα* (dealer) σε κάθε χρήστη, έστω s_i, το οποίο δεν αρκεί για να ανακτηθεί το μυστικό. Για να αποκτήσουν όλοι οι χρήστες το μυστικό s, θα πρέπει να συνδυάσουν τις πληροφορίες τους για να το κατασκευάσουν. Ωστόσο, υπάρχει ένα φράγμα στο πόσα s_i πρέπει να συνδυαστούν ώστε να κατασκευαστεί ολόκληρο το s. Εάν υπάρχουν l χρήστες στο σύστημα, τότε για συνδυασμό t πληροφοριών (προφανώς $t \leq l$), το s μπορεί να ανακτηθεί, ενώ για συνδυασμό $t-1$ πληροφοριών οχι. Η πρώτη υλοποίηση ενός τέτοιου σχήματος έγινε από τον A.Shamir στο [16] που δημοσιεύτηκε το 1979, με την χρήση της πολυωνυμικής παρεμβολής μέσω των συντελεστών Lagrange.

1. Διαλέγεται ένα $\mathcal{P}$ πολυώνυμο βαθμού $t-1$ με τιμή $\mathcal{P}(0) = s$.
2. Ο *dealer* παράγει για τους l χρήστες ζευγάρια (x_i, y_i) και τα διαμοιράζει.
3. Από την θεωρία για την πολυωνυμική παρεμβολή, μόνο εάν υπάρχουν $\{(x_i, y_i)\}_{i=1}^{t}$ ζευγάρια μπορεί να βρεθεί το $\mathcal{P}$ με την βοήθεια των συντελεστών Lagrange μέσω του τύπου :

$$\mathcal{P}(x) = \sum_{i=1}^{t} y_i \prod_{i \neq j,\ j=1}^{t} \frac{x - x_j}{x_i - x_j}$$

Ένας ασφαλής τρόπος να γίνει αυτό είναι ο *dealer* να επιλέγει έναν πρώτο αριθμό p , και t συντελεστές $\{a_1, ..., a_t\} \epsilon \mathbb{Z}_p$. Το πολυώνυμο θα έχει την μορφή :

$$\mathcal{P}(x) = \{a_i x^i\}_{i=1}^{t} + s \ (mod p)$$

και ο σταθερός όρος θα είναι το μυστικό s. Θα μοιράσει λοιπόν τα ζευγάρια (x_i, y_i).

Το πρόβλημα με αυτό το σχήμα είναι ότι υποθέτει πως ο *διανομέας* και οι χρήστες είναι τίμιοι. Δηλαδή ότι ο διανομέας θα δώσει τα σωστά ζευγάρια (x_i, y_i) σε όλους και οι χρήστες αντίστοιχα θα μοιραστούν τα ζευγάρια σωστά ώστε όλοι να μπορούν να φτιάξουν το s. Για αυτό προτείνεται από το πρωτόκολλο [15] ο συνδυασμός του **secret sharing** με τα **commitments**. Το αποτέλεσμα ονομάζεται επαληθεύσιμη διανομή απορρήτων (**VSS**).

Προκειμένου να έχουμε ένα ασφαλές σχήμα διανομής απορρήτων με κακόβουλους χρήστες, εισάγεται το **Verifiable Secret Sharing (VSS)**. Θα πρέπει οι πληροφορίες του διανομέα να μπορούν να επαληθευτούν. Γι' αυτό τον λόγο, αντί να μοιράζονται τα ζευγάρια (x_i, y_i) μόνο, μοιράζεται και μια επιπλέον πληροφορία $\{c_i = g^{a_i}\}_{i=0}^{t}$, όπου g είναι ο γεννήτορας ομάδας με δύσκολο district logarithm problem (DLP) , και μπορεί αποδείξει ότι όντως το ζευγάρι (x_i, y_i) είναι έγκυρο αφού :

$$log_g \prod_{j=0}^{t} c_j^{x_i^j} = y_i$$

Στην συνέχεια οι χρήστες υπολογίζουν το s με τους συντελεστές Lagrange.

G.O.D. Coin Tossing

Το πρωτόκολλο που προκύπτει από τον συνδυασμό των παραπάνω κρυπτογραφικών σχημάτων, για τις ανάγκες του Ouroboros ονομάζεται **Guaranteed Output Delivery Coin Tossing** protocol. Περιγράφεται στην δημοσίευση [13] από τους Ignacio Cascudo και Bernardo David. Είναι ένα πρωτόκολλο coin tossing για ένα σύνολο με ειλικρινή πλειοψηφία χρηστών, που επιτρέπει σε οποιονδήποτε να ελέγξει ότι το αποτέλεσμα προέκυψε δίκαια και αποδοτικά ενώ παράλληλα όλοι όσοι συμμετείχαν θα λάβουν το αποτέλεσμα.

Το πρωτόκολλο πραγματοποιείται μεταξύ n χρηστών, $P_1, ..., P_n$, που έχουν από ένα ζευγάρι δημοσίου - ιδιωτικού κλειδιού pk_i, sk_i. Έστω ένας πρώτος αριθμός p, η ομάδα $\mathbb{Z}_p$ και δύο γεννήτορες της ομάδας h, g.

Κάθε χρήστης εκτελεί τα παρακάτω :

Commit

1. Χρησιμοποιείται ένα PVSS πρωτόκολλο με φράγμα t.
   i) Ως διανομέας επιλέγει ένα μυστικό $s \in \mathbb{Z}_p$ και το κρυπτογραφεί $S = h^s$.

ii) Στην συνέχεια επιλέγει τους συντελεστές $\{a_1, ..., a_{t-1}\} \in \mathbb{Z}_p$ για το πολυώνυμο

$$\mathcal{P}(x) = \{a_i x^i\}_{i=1}^{t-1} + s \ (mod p)$$

iii) Υπολογίζει τις εξατομικευμένες πληροφορίες (shares) s_i για $1 \leq i \leq n$, τις κρυπτογραφεί $\hat{s_i} = pk_i^{s_i}$

iv) Παράγει μια δέσμευση $c_i = g^{s_i}$ και μια απόδειξη $PROOF_d$ που επιτρέπει σε κάθε χρήστη να επαληθεύσει την αυθεντικότητα των shares $\hat{s_i}$.

v) Διαμοιράζει τα $\{\hat{s_i}\}_{i=1}^{n}$ και το $PROOF_d$.

2. Οι χρήστες επαληθεύουν τις πληροφορίες $\hat{s_i}$ με το $PROOF_d$.

3. Οι χρήστες μαθαίνουν τα shares τους $\hat{s_i}$, λαμβάνουν το s_i αποκρυπτογραφώντας με το sk_i, και διαμοιράζονται μια δέσμευση $com(s_i, r_i)$ και ένα $PROOF_i$. Το $r_i \in \mathbb{Z}_p$ είναι μια τυχαιότητα για το commitment.

Reveal

4. Οι χρήστες επαληθεύουν τις πληροφορίες $com(s_i, r_i)$ με το $PROOF_i$..

5. Αφού δουν t δεσμεύσεις της μορφής $com(s_i, r_i)$, γνωστοποιούν τα στοιχεία τους $open(s_i, r_i)$.

Recovery

6. Σε περίπτωση που λάβουν commitments από κάποιο χρήστη j αλλά όχι open, κοινοποιούν το $\hat{s_j}$ και το $PROOF_j$ τους που κατέχουν ως διανομείς ώστε να μαζευτούν t δεσμεύσεις και να ξεκινήσει το $open$.

7. Όταν υπάρχουν t αποκρυπτογραφημένες πληροφορίες της μορφής s_i, παράγεται το S. Η τελική τυχαιότητα είναι $\rho = \prod_i S_i$, όπου i είναι οι χρήστες που έστειλαν commitment.

Εφαρμογή στο Ouroboros

Το πρωτόκολλο **G. O. D. Coin Tossing** μπορεί να εφαρμοστεί μεταξύ των χρηστών του Ouroboros, προκειμένου να παραχθεί μια ομοιόμορφη τυχαιότητα για τον αλγόριθμο εκλογής αρχηγού. Το πρωτόκολλο αυτό είναι ιδανικό για αυτή την δουλειά καθώς είναι αποδοτικό, επιτρέπει

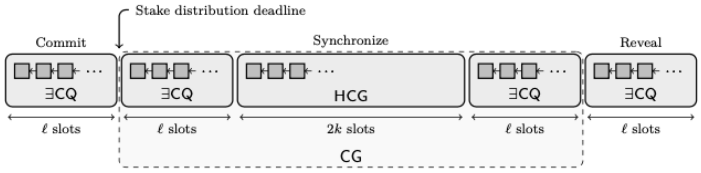
την ομοφωνία σε μια τυχαιότητα, λειτουργεί άρτια παρουσία κακόβουλων χρηστών αρκεί αυτοί να είναι λιγότεροι από ένα ποσοστό, και εγγυάται πως όλοι οι χρήστες θα λάβουν την τιμή της τυχαιότητας.

Επειδή το πρωτόκολλο λειτουργεί υπό την υπόθεση ειλικρινούς πλειοψηφίας των χρηστών, από τους n χρήστες που συμμετέχουν στο G. O. D., αρκούν οι $\frac{n}{2}$ να είναι ειλικρινείς. Οι χρήστες που συμμετέχουν είναι οι R χρήστες της επιτροπής ένας για το κάθε slot της εποχής. Άρα ορίζεται $t = \frac{R}{2}$. Επιπλέον, θέλουμε ο σπόρος τυχαιότητας ρ να είναι ήδη γνωστός στην αρχή κάθε εποχής ώστε να επιτρέπει την εκτέλεση του αλγορίθμου slot leader election. Για αυτόν τον λόγο το πρωτόκολλο G. O. D. εφαρμόζεται από τους κατάλληλους χρήστες της προηγούμενης εποχής. Το Ouroboros υλοποιεί ένα δημόσιο blockchain και είναι σημαντικό να επαληθεύεται από οποιονδήποτε χρήστη είτε είναι ενεργός είτε οχι. Η επαλήθευση αυτή πρέπει να γίνεται και στον αλγόριθμο επιλογής αρχηγού slot και άρα και στην τυχαιότητα. Το G. O. D. Coin Tossing επιτρέπει την επαλήθευση με την χρήση των αποδείξεων μηδενικής γνώσης κατά την εκτέλεση του $(PROOFs)$, αρκεί όλα τα μηνύματα να είναι δημόσια. Έτσι επιλέχθηκε το πρωτόκολλο εύρεσης τυχαιότητας να συμβαίνει παράλληλα και γραμμικά με την εκτέλεση μιας εποχής, και οι πληροφορίες-μηνύματα να αναρτώνται στο blockchain. Για αυτόν τον λόγο ορίζονται δύο παράμετροι k και l που θα είναι οι χρόνοι-slots σε κάθε εποχή όπου θα πραγματοποιούνται τα βήματα του G. O. D. Coin Tossing. Το τελικό πρωτόκολλο ονομάζεται π_{DLS}.

Ως coin tossing πρωτόκολλο το π_{DLS}, θα χωρίζεται σε δύο φάσεις, την φάση της δέσμευσης (commitment) όπου όλοι οι χρήστες θα στέλνουν τις δεσμεύσεις-commitments της μορφής $Com(s_i, r)$, και την φάση της αποκάλυψης (reveal), που οι χρήστες θα ανοίγουν τις δεσμεύσεις για να μπορεί να φτιαχτεί η τυχαιότητα. Το **commitment** διαρκεί για $2k + 3l$ slots. Όλοι οι χρήστες R που ανήκουν στην επιτροπή, ξεκινάνε να εκτελούν το π_{DLS}, όπως στο G. O. D. Coin Tossing, από το πρώτο βήμα στο οποίο όλοι αποτελούν διανομείς της δικής τους τυχαιότητας s^u, με φράγμα $t = \frac{n}{2}$ για το PVSS, και διαμοιράζουν τα κρυπτογραφημένα shares $\hat{s_i}$. Μετά τα $2k + 3l$ slots, οι χρήστες εκτελούν το τέταρτο βήμα του G. O. D. Coin Tossing όπου ελέγχουν με τα zero-knowledge proofs των χρηστών, και τα commitments $\hat{s_i}$. Έτσι το πρωτόκολλο έχει έρθει στην φάση του **reveal**, που οι χρήστες ανοίγουν τις δεσμεύσεις τους στα shares, σύμφωνα με το πέμπτο βήμα, και για όσους χρήστες δεν έχουν ανοίξει τις δεσμεύσεις εκτελούν το έκτο βήμα που εξαναγκάζουν το άνοιγμα. Τέλος μπορούν όλοι σύμφωνα με το τελευταίο βήμα του G. O. D. Coin Tossing να μάθουν την τυχαιότητα s^u του κάθε χρήστη u από τους R, με το να συνδυάσουν τις τιμές s_i^u με έναν αλγόριθμο reconstruct και μια συνάρτηση κατακερματισμού : $s^u = H(Rec(s_1^u, ..., s_R^u)) = \rho_u$. Ο σπόρος $\rho = \oplus_{u=1}^{R} \rho_u$.

Σχηματικά η εξέλιξη του π_{DLS} γραμμικά με το blockchain γίνεται ως εξής

:



Σχήμα 7.3: Εξέλιξη του coin tossing παράλληλα με το blockchain

7.1.7 Περιγραφή του δυναμικού πρωτοκόλλου

Το πρωτόκολλο για δυναμικό stake, είναι παρόμοιο με το πρωτόκολλο στατικού stake σε μια εποχή, αλλά με διαφορές καθώς πλέον η κατανομή χρημάτων δεν είναι σταθερή, αλλά αλλάζει σε κάθε εποχή. Αυτό σημαίνει πως αλλάζει η κατανομή χρημάτων, η επιτροπή από slot leaders και η τυχαιότητα για τον αλγόριθμο επιλογής αρχηγών. Για αυτόν τον λόγο είναι ένα σύνθετο πρωτόκολλο που αποτελείται από δύο υπο-πρωτόκολλα, το π_{DPoS} και το π_{DLS}. Το π_{DPoS} είναι το βασικό πρωτόκολλο με το οποίο αλλάζουν οι εποχές και η κατανομή χρηνάτων, ενώ το π_{DLS} είναι το πρωτόκολλο παραγωγής τυχαιότητας. Το τελικό πρωτόκολλο $\Pi[\pi_{DPoS}, \pi_{DLS}]$, λειτουργεί όπως το π_{DPoS} αλλά εκτελείται παράλληλα και το π_{DLS}. Το π_{DLS} περιγράφτηκε στην προηγούμενη ενότητα, άρα θα γίνει περιγραφή του π_{DPoS}.

Αρχικοποίηση

Κατα την αρχικοποίηση του δυναμικού πρωτοκόλλου, ο κάθε χρήστης φροντίζει να αποκτήσει τις απαραίτητες πληροφορίες ώστε να προετοιμαστεί για το στάδιο επέκτασης της αλυσίδας. Πρώτα λαμβάνει από το περιβάλλον το ζευγάρι δημοσίων και ιδιωτικών κλειδιών του (pk, sk). Στην συνέχεια ενημερώνεται για το τρέχων slot που κάνει την αρχικοποίηση. Εάν είναι το πρώτο slot sl_1, λαμβάνει το genesis block $genblock$, την κατανομή των stake S_0, τον σπόρο τυχαιότητας ρ_0 και την συνάρτηση εκλογής leader F και κρατάει ένα τοπικό blockchain $C = B_0$. Αν το τρέχων slot δεν είναι το πρώτο, λαμβάνει την αλυσίδα από το σύστημα και φτιάχνει τοπικό blockchain C . Στην πρώτη περίπτωση ορίζει $st = H(B_0)$, αλλιώς $st = H(head(C))$.

Με λίγα λόγια κατα την αρχικοποίηση προσδιορίζονται τα εξής :

1. Ζευγάρι κλειδιών

2. Τρέχων slot και αντίστοιχες πληροφορίες
3. Τοπικό Blockchain
4. Κατάσταση state

Παραγωγή τυχαιότητας

Καλείται το π_{DLS} από τους slot leaders για τον προσδιορισμό της τυχαιότητας ρ_j της εποχής e_j όπως περιγράφεται εδώ. Το πρωτόκολλο αυτό τρέχει παράλληλα με την επέκταση της αλυσίδας και την παραγωγή συναλλαγών.

Επέκταση Αλυσίδας

Αφού ο χρήστης έχει δημιουργήσει μια τοπική αλυσίδα και έχει όλες τις απαραίτητες πληροφορίες, μπαίνει στην διαδικασία επέκτασης της αλυσίδας. Πρόκειται για μια επαναλαμβανόμενη διαδικασία που συμβαίνει R φορές, όσα και τα slot της εποχής, και πραγματοποιείται από κάθε χρήστη, είτε είναι slot leader είτε όχι.
Εάν πρόκειται για μια νέα εποχή e_j, $j \geq 2$, κάθε χρήστης διαβάζει το blockchain και λαμβάνει την κατανομή των χρημάτων $\mathbb{S}_j$ από το τελευταίο block $(j-1)R-2(k+l)-E$. Στην συνέχεια λαμβάνεται η τυχαιότητα ρ_j που παράχθηκε στην προηγούμενη εποχή e_{j-1}.
Ο χρήστης λαμβάνει τις συναλλαγές $d \in \{0,1\}^*$ που πρόκειται να βάλει σε κάποιο block του blockchain. Στην συνέχεια ενημερώνεται για όλες τις νέες αλυσίδες που υπάρχουν διαθέσιμες, και ορίζει ως νέα αλυσίδα αυτή που προκύπτει από την συνάρτηση maxvalid$(\mathbb{C}, C)$, εφόσον επαληθεύονται οι υπογραφές των block που περιέχει και οι slot leader επαληθεύονται από τον αλγόριθμο LeaderVerify$(\varrho, \mathbb{S}, pk, sl)$. Τότε $C' =$ maxvalid$(\mathbb{C}, C)$ και C' είναι η καινούρια τοπική αλυσίδα, και $st = H(head(C'))$. Στην συνέχεια, στην περίπτωση που ο αλγόριθμος LeaderSelection$(\varrho, \mathbb{S}, sl)$ έχει επιστρέψει το δημόσιο κλειδί του, φτιάχνει το $B = (st, d, sl, \sigma)$ όπου d οι συναλλαγές και σ η υπογραφή $\sigma = Sign_{sk}(st, d, sl)$. Δημιουργεί την καινούργια τοπική αλυσίδα $C'' = C'|B$ και την μεταδίδει στο σύστημα. Επίσης ορίζεται $st = H(head(C''))$.

Με λίγα λόγια κατα την επέκταση αλυσίδας έχουμε τα βήματα :

1. Καινούρια κατανομή χρημάτων
2. Λήψη τυχαιότητας
3. Για κάθε slot $\in 1, .., R$:
   (a) Δεδομένα συναλλαγών
   (b) Συλλογή έγκυρων αλυσιδών
   (c) Εφαρμογή maxvalid()
   (d) Ενημέρωση τοπικής αλυσίδας και κατάστασης
   (e) Έλεγχος LeaderSelection$(\varrho, \mathbb{S}, sl)$

(f) Για θετική απάντηση :
- Δημιουργία καινούριου block
- Δημιουργία καινούριας αλυσίδας με το block
- Μετάδοση καινούριας αλυσίδας
- Ενημέρωση τοπικής αλυσίδας και κατάστασης

Παραγωγή συναλλαγών

Σε αυτό το στάδιο του πρωτοκόλλου ο χρήστης μπορεί να παράξει συναλλαγές, σύμφωνα πάντα με ένα υπόδειγμα έγκυρης συναλλαγής. Για αυτόν τον λόγο δημιουργεί και μια υπογραφή μαζί με την συναλλαγή tx, και στην συνέχεια αποστέλνει το ζευγάρι (tx, σ) στο περιβάλλον.

Παρατήρηση 7.1.4. Το τελικό πρωτόκολλο $\Pi[\pi_{DPoS}, \pi_{DLS}]$ δέχεται σαν παραμέτρους τις $k, l, E, \in, \sigma, R, L$. Η παράμετρος k αφορά την ιδιότητα CP, common prefix, δηλαδή εάν σε μια αλυσίδα αφαιρέσουμε k slots, τότε θα προκύψει κοινό πρόθημα. Η παράμετρος l αφορά την ιδιότητα CQ και προσδιορίζει ότι σε l slot θα υπάρχει τουλάχιστον ένα ειλικρινές block. Το E είναι μια παράμετρος που ονομάζεται *lookahead* και αφορά τα slots στα οποία ο adversary μπορεί να έχει πρόσβαση στην παραγωγή της τυχαιότητας, πριν από τους ειλικρινείς χρήστες. Οι παράμετροι $\in, \sigma$ είναι ο περιορισμός του αντιπάλου στο ποσοστό των χρηστών-χρημάτων που μπορεί να διαφθείρει και το stake shift, η στατιστική διαφορά στις κατανομές χρημάτων δύο εποχών αντίστοιχα. Τέλος R είναι ο αριθμός των slot μιας εποχής και L είναι η διάρκεια ζωής του συστήματος σε slots.

Αποδεικνύεται ότι η εκτέλεση του πρωτοκόλλου $\Pi[\pi_{DPoS}, \pi_{DLS}]$ σε χρόνο L, για τις παραπάνω παραμέτρους με $R \geq 2k + 4l$, L πολλαπλάσιο του R, και αντίπαλο $\mathcal{A}$ *$(\frac{1}{2} - \varepsilon)$-αρχικά-περιορισμένο* με καθυστέρηση διαφθοράς $D = 2R$, οι ιδιότητες persistence και liveness με παραμέτρους k και $2(k + l)$ αντίστοιχα, θα ισχύουν και άρα η εκτέλεση του πρωτοκόλλου θα είναι ασφαλής.

7.1.8 Κίνητρα συμμετοχής

Ο $\mathcal{A}$ έχει στόχο να εμποδίσει καινούριες συναλλαγές να επικυρώνονται και άρα να εμποδίσει την αλυσίδα από το να συνεχίσει να αυξάνεται. Οι έμπιστοι χρήστες όμως τι κίνητρο έχουν να συμμετέχουν; Στόχος είναι να βρεθεί μια δίκαιη λύση ώστε οι έμπιστοι χρήστες να μην έχουν κίνητρο να παρεκκλίνουν από το πρωτόκολλο. Για αυτό εισάγεται ο ρόλος των *input*

endorsers και των χρηματικών κινήτρων, *incentives*.

Οι *input endorsers* εκλέγονται βάσει των χρημάτων που διαθέτουν όπως ακριβώς οι slot leaders. Εάν κάποιος χρήστης έχει εκλεχθεί ως slot leader δεν αποκλείεται να εκλεχθεί και ως input endorser και αντίστροφα. Δηλαδή ένας χρήστης μπορεί να έχει δύο ρόλους στο πρωτόκολλο. Ο ρόλος των input endorsers είναι να προσυπογράφουν τις συναλλαγές που θα προστεθούν στα καινούρια block και εκλέγεται ένας input endorser για κάθε slot (είναι εφικτό με κάποιες μετατροπές να εκλέγονται και περισσότεροι). Οι προτάσεις των χρηστών αυτών, εαν δεν έχουν χρησιμοποιηθεί σε κάποιο συγκεκριμένο block, παραμένουν διαθέσιμες για d slots, για οποιονδήποτε slot leader. Άρα οι προτάσεις των input endorsers διαμοιράζονται ξεχωριστά από τα blocks ώστε αν κάποιες δεν έχουν χρησιμοποιηθεί να είναι διαθέσιμες για d slot ακόμη. Οι input endorsers θα πρέπει να είναι ενεργοί, καθώς εάν δεν υπάρχουν διαθέσιμες προτάσεις από input endorsers για τους slot leaders, τότε αυτοί θα παράξουν ένα κενό block. Σε περίπτωση που μια συναλλαγή έχει μπει δύο φορές στην αλυσίδα, παραμένει μόνο η πρώτη χρονικά εμφάνιση.

Τα *incentives* θα πρέπει να μοιράζονται με τέτοιο τρόπο ώστε να κινητοποιούν τους χρήστες να συμμετέχουν πιο πολύ στο πρωτόκολλο, δηλαδή να ανταμοίβονται για την διαθεσιμότητα (availability) και την επαλήθευση συναλλαγών (transaction verification). Όσον αφορά την διαθεσιμότητα, υπάρχουν κάποιες περιπτωσεις που είναι αναγκαίο να είναι ενεργοί. Συγκεκριμένα :

- Ένα slot πριν να ενεργήσουν ως slot leaders για να αποκτήσουν την μακρύτερη αλυσίδα και τις συναλλαγές των input endorsers που θα βάλουν στο block.

- Κατα το slot στο οποίο είναι επιλεγμένοι ως slot leaders.

- Στο slot στο οποίο θα πρέπει να συμμετάσχουν στην παραγωγή τυχαιότητας με το να στείλουν την δέσμευση της δικής τους τυχαιότητας.

- Στο slot στο οποίο θα πρέπει να συμμετάσχουν στην παραγωγή τυχαιότητας με το να ανοίξουν την δέσμευση και να αποκρύψουν το μέρος της μυστικής πληροφορίας s_i που διαθέτουν.

- Να ελέγχουν συχνά εάν έχουν επιλεχθεί ως slot leaders ή ως input endorsers.

- Κατα το slot στο οποίο είναι επιλεγμένοι ως input endorsers για να μαζέψουν τις συναλλαγές, να τις επεξεργαστούν και να τις επικυρώσουν.

Τα χρηματικά κίνητρα που θα δίνονται στους διαθέσιμους και ενεργούς χρήστες μπορούν να προέλθουν από μια επιβάρυνση (fee) που θα έχουν όσοι πραγματοποιούν συναλλαγές και θα μαζεύονται σε ένα pool. Στο τέλος μιας συγκεκριμένης χρονικής περιόδου θα μοιράζονται στους input endorsers ανάλογα με το ποσοστό συμματοχής τους και στους slot leaders που συμμετείχαν στα slots αυτής της περιόδου. Πιο συγκεκριμένα, για τα blocks που αφοορύν την εποχή e_j, δημιουργείται μια ακολουθία από τις συναλλαγές που προτάθηκαν από τους input endorsers για αυτά, και από την ακολουθία αυτή εξάγονται οι χρηματικές επιβαρύνσεις (fees). Τα χρήματα αυτά αθροίζονται και συγκεντρώνονται σε ένα σύνολο που ονομάζεται $P_{all}^{(j)}$ και αφορά την εποχή e_j. Τότε κάθε χρήστης i μπορεί να διεκδικήσει μέρος από το $P_{all}^{(j)}$ με την φόρμουλα :

$$\left(\beta \cdot \frac{|\{j \mid E_j\}|}{r} \; + \; (1-\beta) \cdot \frac{|\{j \mid L_j\}|}{R}\right) \cdot P_{all}^{(j)}$$

όπου :

- $\beta \in [0,1]$ είναι μια παράμετρος του πρωτοκόλλου
- r είναι ο αριθμός των input endorsers της εποχής e_j
- E_j είναι οι χρήστες που έχουν εκλεχθεί ως input endorsers για $j \in \{1,2,...,r\}$ σε κάποιο slot της e_j
- R είναι τα slots της εποχής e_j
- L_j είναι οι χρήστες που έχουν εκλεχθεί ως slot leaders σε κάποιο από τα $j \in \{1,2,...,R\}$ slots της e_j

Οι χρήστες i μπορούν να διεκδικήσουν μέρος του $P_{all}^{(j)}$ σε οποιαδήποτε στιγμή μετά από $2(k+l)$ slots της επόμενης εποχής από την e_j. Παρατηρούμε πως οι input endorsers και οι slot leaders αμείβονται ανάλογα με το πόσες φορές έχουν εκλεχθεί στο πρωτόκολλο για αυτόν τον ρόλο, προς τον συνολικό αριθμό των εκλεγόμενων για αυτόν τον ρόλο. Άρα κάποιος χρήστης αμοίβεται μόνο για την επιλογή του και όχι για το πόσο έχει συνεισφέρει στο πρωτόκολλο, κάτι που αναφέρεται στο [1] ότι μπορεί να βελτιωθεί μελλοντικά. Το σύστημα με τον μηχανισμό απόδοσης χρηματικών αμοιβών στους χρήστες, αποδεικνύεται ότι είναι μια προσεγγιστικά *δ-Ισορροπία Νας*. Δηλαδή ένας χρήστης μέσω οποιασδήποτε στρατηγικής, δεν μπορεί να επωφεληθεί περισσότερο από δ χρήματα στις αμοιβές του.

Θεώρημα 7.1.3. *Κατα την εκτέλεση του πρωτοκόλλου* $\Pi[\pi_{DPoS}, \pi_{DLS}]$ *σε χρόνο* L*, για τις παραμέτρους* $k, l, E, \in, \sigma, R, L$ *με* $R \geq 2k + 4l$*,* L *πολλαπλάσιο του* R*, και αντίπαλο* $\mathcal{A}$ $(\frac{1}{2} - \varepsilon)$*-αρχικά-περιορισμένο με καθυστέρηση διαφθοράς* $D = 2R$*, η στρατηγική ένας χρήστης να φερθεί ειλικρινώς είναι προσεγγιστικά μια δ-Ισορροπία Νας, ενάντια σε*

οποιαδήποτε στρατηγική αντιπάλου $\mathcal{A}$. Οι συνολικές αμοιβές της εκτέλεσης του Π *είναι φραγμένες από μια παράμετρο* λ*, και* $\delta > 0$.

Απόδειξη
Για την απόδειξη θεωρούμε πως ο αριθμός των χρηστών είναι σταθερός, όπως και το stake.
Έστω ένα σύνολο V από κακόβουλους χρήστες, που εκτελούν το πρωτόκολλο μαζί με άλλους ειλικρινείς χρήστες για L slots. Σύμφωνα με το πρωτόκολλο, όποια και αν είναι η στρατηγική των κακόβουλων χρηστών, οι ιδιότητες persistence και liveness έχει αποδειχθεί ότι θα ισχύουν. Δηλαδή, σύμφωνα με την ιδιότητα liveness, τουλάχιστον ένα ειλικρινές block θα παράγεται σε l slot. Δηλαδή οι input endorsers θα μπορούν να συμμετάσχουν τουλάχιστον μια φορά στο διάστημα των l slot, και για όσες φορές έχουν εκλεχθεί. Αντίστοιχα και οι slot leaders θα μπορούν να παράξουν τα αναμενόμενα blocks.
Η πρώτη κίνηση ονομάζεται α_1. Άρα εάν το πρωτόκολλο λειτουργήσει σωστά με τις ιδιότητες persistence και liveness, η ωφελιμότητα των χρηστών V, U_V, αφού οι αμοιβές δίνονται από την ίδια συνάρτηση για όλους τους χρήστες, είναι :

$$U_V(\alpha_1) = \sum_{i=1}^{t} \left(\beta \cdot \frac{num_i(E_j)}{r} \; + \; (1-\beta) \cdot \frac{num_i(L_j)}{R} \right) \cdot P_{all}^{(j)}$$

Η δεύτερη κίνηση ονομάζεται α_2. Εάν το πρωτόκολλο δεν λειτουργεί σωστά και αποτύχει, τότε η ωφελιμότητα θα είναι οι χρήστες V να λάβουν όλες τις αμοιβές αφού το πρωτόκολλο θα αποτύχει και αυτές δεν θα μοιραστούν :

$$U_V(\alpha_2) = P_{all}^{(j)}$$

Στην συνέχεια υπολογίζεται η συνολική αναμενόμενη ωφελιμότητα που είναι η αναμενόμενη ωφελιμότητα κάθε κίνησης επί την πιθανότητα να συμβεί η κίνηση. Τελικά η αναμενόμενη ωφελιμότητα για την εκτέλεση του πρωτοκόλλου είναι :

$$E[U_V] = E[U_V(\alpha_1)] + E[U_V(\alpha_2)]$$
$$E[U_V] = E[U_V(\alpha_1)] + P_{all}^{(j)} \cdot \varepsilon$$

και επειδή οι συνολικές αμοιβές της εκτέλεσης του Π είναι φραγμένες από μια παράμετρο λ, ισχύει ότι :

$$\varepsilon P_{all}^{(j)} \leq \delta$$

Άρα :

$$E[U_V] = E[U_V(\alpha_1)] + \delta$$

Δηλαδή το πρωτόκολλο Π είναι προσεγγιστικά δ-*Nash Equilibrium.*
□

7.1.9 Αντιπρόσωποι

Ο τρόπος με τον οποίο το πρωτόκολλο είναι δομημένο έχει δύο προβλήματα για τους χρήστες. Οι χρήστες οι οποίοι πρέπει να συμμετέχουν ως slot leaders ή input endorsers θα πρέπει να είναι ενεργοί αρκετά συχνά για τις ανάγκες του πρωτοκόλλου. Από την άλλη υπάρχουν αρκετοί χρήστες που θα ήθελαν να συμμετέχουν αλλά δεν έχουν αρκετό stake ώστε να εκλεχθούν από τον αλγόριθμο επιλογής. Για τους λόγους αυτούς εισάγεται ένας ακόμη ρόλος για τους χρήστες του Ouroboros, οι χρήστες με αυτόν τον ρόλο ονομάζονται *Αντιπρόσωποι* ή *Delegates*.

Οι αντιπρόσωποι, θα μπορούν να δρουν στην θέση των εκλεγμένων slot leaders σε ορισμένες ενέργειες όπως στην διαδικασία παραγωγής τυχαιότητας. Πιο συγκεκριμένα, οι slot leaders θα μπορούν να δίνουν άδεια σε χρήστες που θα ονομάζονται αντιπρόσωποι, ώστε να εκτελούν για αυτούς το πρωτόκολλο π_{DLS}. Οι *delegates* για να μπορούν να συμμετέχουν θα πρέπει να εκπροσωπούν ένα καλό ποσοστό χρημάτων, ώστε να αποφεύγονται επιθέσεις όπου θα υπάρχουν τόσο πολλοί αντιπρόσωποι που η απόδοση του πρωτοκόλλου θα μειώνεται. Ο $\mathcal{A}$ θα μπορούσε δηλαδή να χωρίσει τα χρήματα του σε πολλούς λογαριασμούς ώστε οι αντιπρόσωποι να αποτελούν ένα μεγάλο σύνολο λογαριασμών που μπορεί το πρωτόκολλο να μην μπορεί να υποστηρίξει. Για αυτό ορίζεται το μικρότερο δυνατό ποσοστό χρημάτων για να γίνει κάποιος delegate ως 1%.

Μια υλοποίηση ενός σχήματος με αντιπροσώπους μπορεί να γίνει με την χρήση των proxy signatures. Οι proxy signatures είναι ένας ειδικός τύπος ηλεκτρονικών υπογραφών σύμφωνα με τον οποίο ένας χρήστης (original signer) μπορεί να μεταβιβάσει το δικαίωμα υπογραφής του σε έναν άλλο χρήστη (proxy signer) [14]. Το κρυπτογραφικό αυτό εργαλείο επιτρέπει σε οποιονδήποτε να πειστεί ότι ο delegate-proxy signer είναι εξουσιοδωτημένος να δράσει στην θέση του slot leader-original signer, αυτό μπορεί να γίνει με την χρήση πιστοποιητικών που θα παρέχουν τις απαραίτητες πληροφορίες για την επαλήθευση και θα δημοσιεύονται στο blockchain. Οι delegates μπορούν να είναι αντιπρόσωποι χρηστών για περιορισμένο χρονικό διάστημα, κάτι το οποίο μπορούν να ορίσουν οι ίδιοι οι slot leaders, για παράδειγμα οι proxy υπογραφές τους να είναι έγκυρες για ορισμένα slot. Από το πρωτόκολλο προτείνεται η χρήση των proxy signatures όπως παρουσιάζονται στο [14].

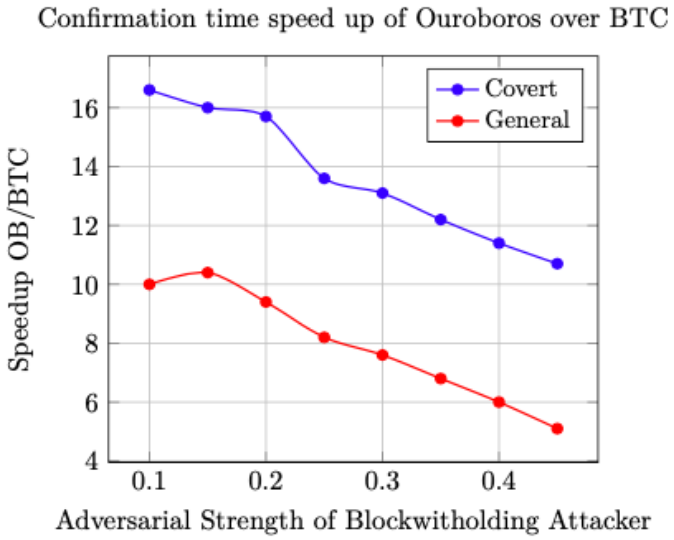
7.1.10 Πειράματα αποδοτικότητας

Εκτός από το ισχυρό θεωρητικό υπόβαθρο του πρωτοκόλλου [1], παρουσιάζονται και πολύ ενδιαφέροντα πειραματικά αποτελέσματα για την απόδοση του Ouroboros Classic. Πρώτα γίνεται ένα πείραμα σύγκρισης του χρόνου επιβεβαίωσης συναλλαγών του Ouroboros σε σχέση με το Bitcoin, όταν ένας αντίπαλος $\mathcal{A}$ προσπαθεί να εκτελέσει επίθεση double-spending και η δύναμή του αυξάνεται από 10% σε 45%, ώστε η πιθανότητα να επιτύχει η επίθεση να είναι 0.1%.

Adversary	BTC	OB Covert	OB General
0.10	50	3	5
0.15	80	5	8
0.20	110	7	12
0.25	150	11	18
0.30	240	18	31
0.35	410	34	60
0.40	890	78	148
0.45	3400	317	663

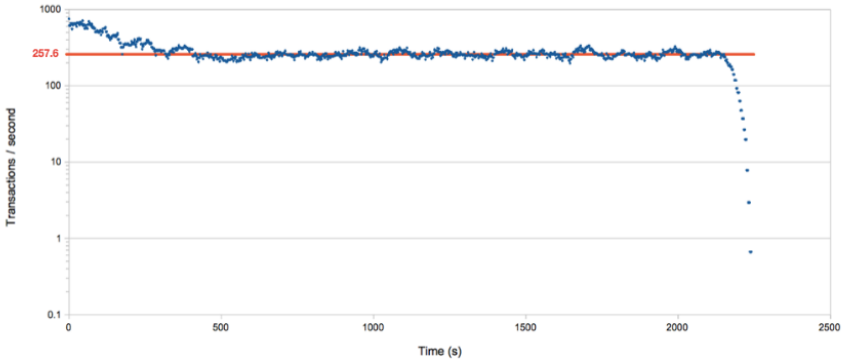
Σχήμα 7.4: Επιβεβαίωση συναλλαγών του Bitcoin και του Ouroboros σε λεπτά, σε σχέση με την δύναμη του αντιπάλου $\mathcal{A}$

Παρατηρούμε από τον πίνακα πως ακόμη και για μικρό ποσοστό ελέγχου του $\mathcal{A}$, στο Bitcoin έχουμε πολύ μεγαλύτερη αναμονή για τον verifier ώστε να επιβεβαιωθούν οι συναλλαγές. Βέβαια αυτό έχει να κάνει και με το ποσοστό παραγωγής block που στο Bitcoin είναι 10 λεπτά για ένα block, ενώ στο Ouroboros είναι 20 δευτερόλεπτα. Στην συνέχεια παρουσιάζεται για το ίδιο πείραμα, η επιτάχυνση του Ouroboros στην επιβεβαίωση συναλλαγών που για γενικό αντίπαλο η ταχύτητα είναι 10-5 φορές μεγαλύτερη από αυτή του Bitcoin.



Σχήμα 7.5: Ταχύτητα επιβεβαίωσης συναλλαγών του Bitcoin και του Ouroboros σε λεπτά, σε σχέση με την δύναμη του αντιπάλου $\mathcal{A}$

Στο τελευταίο σχήμα βλέπουμε από μια εφαρμογή του Ouroboros σε Amazon Elastic Compute Cloud (EC2), με 40 κόμβους να συμμετέχουν, και διάρκεια slot 5 δευτερολέπτων. Η μέση τιμή συναλλαγών που προστίθενται στην αλυσίδα είναι 257.6 συναλλαγές ανά δευτερόλεπτο.



Σχήμα 7.6: Αριθμός συναλαγών σε ένα δυτερόλεπτο για 40 κόμβους και slot διάρκειας 5 δευτερολέπτων

7.2 Ouroboros Praos

Το Ouroboros Praos [2] είναι μια εξελιγμένη μορφή του Ouroboros Classic που περιγράφτηκε στο προηγούμενο κεφάλαιο. Ενώ το Ouroboros Classic είναι ένα Proof-of-Stake blockchain με πολύ καλύ ανάλυση ασφάλειας, παρέχει ιδιότητα αυστηρού συγχρονισμού στην επικοινωνία, ενώ ο αντίπαλος $\mathcal{A}$ είναι περιορισμένος στο να διαφθείρει χρήστες με καθυστέρηση. Στο Ouroboros Praos επιτυγχάνεται η λειτουργία ενός πρωτοκόλλου με άμεση διαφθορά από τον $\mathcal{A}$ (fully adaptive corruption) και με ένα μερικώς συγχρονισμένο δίκτυο (semi-synchronous network). Αυτά είναι εφικτά μέσω νέων εργαλείων, των *forward secure digital signatures* και των *Verifiable Random Functions*.

7.2.1 Εισαγωγή

Το Ouroboros Praos αναλύεται όπως το Ouroboros Classic. Για αρχή περιγράφεται η στατική κατάσταση του πρωτοκόλλου, όπου δηλαδή τα χρήματα στο σύστημα παραμένουν σταθερά και στην συνέχεια περιγράφεται το πρωτόκολλο στην δυναμική του μορφή όπου η κατανομη χρησμάτων αλλάζει ανα χρονικές περιόδους. Αρχικά λοιπόν περιγράφεται ένα γενικό μοντέλο με συμβολισμούς και ορισμούς, στην συνέχεια περιγράφεται η διαδικασία εκλογής των αρχηγών και τα δύο πρωτόκολλα στατικό και δυναμικό με ανάλυση ασφάλειας.

7.2.2 Περιγραφή μοντέλου

Χρόνος

Ο χρόνος στο πρωτόκολλο Ouroboros Praos εξελίσσεται με τον ίδιο τρόπο όπως και στο Ouroboros Classic. Περιγραφή του χρόνου στου κλασικό πρωτόκολλο του Ouroboros εδώ. Η μόνη διαφορά είναι ότι σε χρόνο ενός slot δεν είναι απαραίτητο ότι τα μηνύματα μεταξύ ειλικρινών χρηστών θα παραδοθούν.

Βασική υπόθεση ειλικρινούς πλειοψηφίας

Η υπόθεση ειλικρινούς πλειοψηφίας παραμένει ίδια με το κλασικό πρωτόκολλο

του Ouroboros και περιγράφεται εδώ.

Μοντέλο αντιπάλου
Το μοντέλο αντιπάλου του Ouroboros Praos περιλαμβάνει έναν πολύ πιο

ισχυρό αντίπαλο σε σχέση με τον αντίπαλο του κλασικού πρωτοκόλλου. Ο $\mathcal{A}$ ακόμη μπορεί να διαφθείρει και να ελέγξει οποιονδήποτε χρήστη θέλει. Επίσης μπορεί να διαβάσει όλα τα μηνύματα που στέλνονται, να αλλάξει την σειρά αποστολής, καθώς και να καθυστερήσει μηνύματα. Η διαφορά είναι ότι σε αυτό το πρωτόκολλο μπορεί να επέμβει στην επικοινωνία μεταξύ ειλικρινών χρηστών, και να καθυστερήσει μηνύματα μέχρι το πολύ $\Delta \in \mathbb{N}$ slots. Επιπλέον δεν υπάρχει καθυστέρηση στην διαφθορά των χρηστών, όπως στο Ouroboros Classic που υπήρχε καθυστέρηση μετά από D slots. Το ποσοστό χρημάτων που ελέγχει ο $\mathcal{A}$, σύμφωνα με την υπόθεση ειλικρινούς πλειοψηφίας, σε κάθε slot θα πρέπει να είναι μικρότερο από 50%.

Ιδιότητες ενός blockchain συναλλαγών
Οι επιθυμητές ιδιότητες του Ouroboros Praos παραμένουν οι ίδιες :

- **Persistence** (με παράμετρο $k \epsilon \mathbb{N}$)
  Η ιδιότητα αυτή έχει να κάνει με την *σταθερότητα* μιας συναλλαγής. Εάν ένας κόμβος θεωρήσει μια συναλλαγή *σταθερή* ($stable$) τότε και οι υπόλοιποι κόμβοι όταν ερωτηθούν θα πρέπει να βλέπουν την συγκεκριμένη συναλλαγή σταθερή και συμφωνούν σε ό,τι προϋπάρχει πριν από αυτή. Η συναλλαγή θεωρείται $\sigma \tau \alpha \theta \epsilon \rho$ αν και μόνο αν η συναλλαγή βρίσκεται k blocks βαθιά στο blockchain.

- **Liveness** (με παράμετρο $u \epsilon \mathbb{N}$)
  Αν όλοι οι έμπιστοι χρήστες στο σύστημα συμφωνήσουν στο να συμπεριληφθεί μια καινούρια συναλλαγή στο blockchain, τότε μετά από χρόνο u slot όλοι οι κόμβοι που θα ερωτηθούν αν απαντήσουν ειλικρινώς θα κρίνουν την συναλλαγή αυτή ως *σταθερή*.

- **Common prefix** (CP: $k \epsilon \mathbb{N}$)
  Δύο αλυσίδες C_1, C_2 υιοθετημένες από δύο έμπιστες ομάδες χρηστών κατά την έναρξη των $sl_1 \leq sl_2$, είναι τέτοιες ώστε $C_1^{\lceil k} \preceq C_2$. Με τον όρο $C_1^{\lceil k}$ εννοούμε την αλυσίδα που προκύπτει εάν αφαιρέσουμε από την C_1 τα k τελευταία blocks. Η ένδειξη $\preceq$ συμβολίζει την σχέση προθήματος.

- **Honest Chain Growth** (HCG: $\tau \epsilon [0,1],\ s \epsilon \mathbb{N}$)
  Αν C είναι μια αλυσίδα υιοθετημένη από έμπιστη ομάδα χρηστών,

και sl_2 είναι το slot με το τελευταίο block της C, και sl_1 ένα προηγούμενο slot με έμπιστο block. Αν $sl_2 \geq sl_1 + s$, τότε ο αριθμός των block που υπάρχουν μετά το sl_1 στην C είναι τουλάχιστον $\tau \cdot s$. Το τ ονομάζεται συντελεστής ταχύτητας.

- **Existential Chain Quality** ($\exists$ CQ: $s \epsilon \mathbb{N}$)
  Έστω μια αλυσίδα C υιοθετημένη από μια έμπιστη ομάδα κατα την έναρξη ενός slot. Τότε οποιοδήποτε τμήμα της C επεκτείνει s προηγούμενα slot, περιέχει τουλάχιστον ένα έμπιστα δημιουργημένο block.

Υποθέσεις ασφάλειας
Το πρωτόκολλο του Ouroboros Praos είναι ασφαλές κάτω από τις εξής

υποθέσεις :

- Το πρωτόκολλο λειτουργεί υπό μερικώς συγχρονισμένο δίκτυο. Δηλαδή υπάρχει καθυστέρηση $\Delta \in \mathbb{N}$ slots.

- Σε κάθε slot ο $\mathcal{A}$ μπορεί να ελέγχει το πολύ 50% των συνολικών χρημάτων.

- Υπάρχει μέγιστος αριθμός slot που ένας χρήστης μπορεί να είναι ανενεργός. Με άλλα λόγια οι χρήστες δεν παραμένουν για πολλή ώρα εκτός σύνδεσης. Αυτό μπορεί να βελτιωθεί με την εφαρμογή του *lazy honesty*.

Βασικοί ορισμοί
Θα παρουσιαστούν κάποιοι βασικοί ορισμοί όπως αναγράφονται στο [2]

με σκοπό την εξοικείωση με τους συμβολισμούς του πρωτοκόλλου.

Ορισμός 7.2.1 (**Genesis Block**)**.** Το Genesis Block (B_0) περιλαμβάνει τους χρήστες με τα δημόσια κλειδιά των χρηστών για την εκλογή του αρχηγού μέσω των vrfs, και τα κλειδιά των υπογραφών προς αντιστοιχία με τα χρηματικά ποσά που διαθέτουν στο σύστημα :

$$\mathbb{S}_0 = \left((U_1, pk_1^{vrf}, pk_1^{kes}, pk_1^{dsig}, s_1), ..., (U_n, pk_n^{vrf}, pk_n^{kes}, pk_n^{dsig}, s_n)\right)$$

Επίσης περιλαμβάνει τον σπόρο η.

Ορισμός 7.2.2 (**State**)**.** Είναι ένα αλφαριθμητικό $st \in \{0,1\}^\lambda$ και προσδιορίζει μια κατάσταση της αλυσίδας.

Ορισμός 7.2.3 (**Block**)**.** Κάθε block $B = (sl_j, st, d, B_{\pi j}, \sigma_j)$ που δημιουργείται σε ένα $slot_j$ περιέχει την τρέχουσα κατάσταση της αλυσίδας st, τις πληροφορίες που θα μπούν στην αλυσίδα $d \in \{0,1\}^*$, την τιμή $B_{\pi j}$ που είναι η απόδειξη ότι το block B είναι έγκυρο ,τον χρόνο που δημιουργήθηκε $sl_j \in \{1, ..., R\}$ και την υπογραφή του χρήστη U_i που το παρήγαγε $\sigma_i = Sign_{sk}(st, d, sl_j, B_{\pi j})$.

Ορισμός 7.2.4 (**Blockchain**)**.** Μια αλυσίδα blockchain C με αρχή το genesis block B_0, είναι μια ακολουθία από blocks $B_1, ...B_n$ συσχετιζόμενα με μια αντίστοιχη ακολουθία από slots που είναι αυστηρά αυξανόμενη. Σε κάθε B_i, η κατάσταση προσδιορίζεται ως η τιμή κατακερματισμού του προηγούμενου block $H(B^{i-1})$. Το μήκος της αλυσίδας συμβολίζεται ως $len(C) = n$ και είναι το νούμερο n από blocks που περιέχει. Το τελευταίο block της C, B_n, ονομάζεται $head(C)$. Ένα κενό string το ορίζουμε ως $head(\varepsilon) = \varepsilon$.

Ορισμός 7.2.5 (**Πρόθημα**)**.** Σε μια αλυσίδα blockchain C μήκους n, εάν αφαιρέσουμε τα k δεξιότερα blocks, προκύπτει η $C^{\lceil k}$. Για $k \geq len(C)$, έχουμε $C^{\lceil k} = \varepsilon$. Για δύο αλυσίδες C_1, C_2, λέμε ότι η C_1 είναι πρόθημα της C_2 και συμβολίζουμε με $C_1 \preceq C_2$, εάν αφαιρώντας κάποια blocks από την C_2 προκύψει η C_1.

Ορισμός 7.2.6 (**Εποχή**)**.** Ως εποχή (epoch), ορίζεται ένα σύνολο από R διαδοχικά slots $S = \{sl_1, ..., sl_R\}$.

Ορισμός 7.2.7 (**Ποσοστό χρημάτων αντιπάλου**)**.** Έστω U_A το σύνολο των χρηστών που ελέγχονται από τον αντίπαλο $\mathcal{A}$. Τότε το ποσοστό των χρημάτων του αντιπάλου στο σύστημα n χρηστών είναι :

$$\alpha = \frac{\sum_{j \in U_A} s_j}{\sum_{i=1}^{n} s_i}$$

Ορισμός 7.2.8 (**Έγκυρη συναλλαγή**)**.** Μια συναλλαγή (tx, σ) θεωρείται έγκυρη εάν:

- Έχει μορφή : "Ο χρήστης pk_1 μεταφέρει στον χρήστη pk_2 ένα ποσό x νομισμάτων με σειριακό νούμερο συναλλαγής sn".

- Η υπογραφή σ που έχει η συναλλαγή επαληθεύεται δωσμένου του αντίστοιχου δημοσίου κλειδιού.

- Δεν υπάρχουν δύο συναλλαγές από τον ίδιο χρήστη με το ίδιο νούμερο συναλλαγής sn.

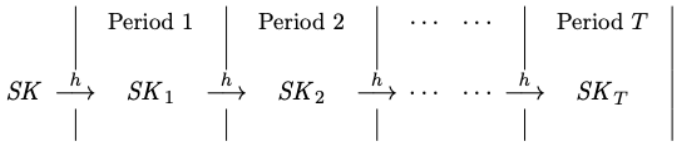
Ορισμός 7.2.9 (**Έγκυρη αλυσίδα**)**.** Μια αλυσίδα C θεωρείται έγκυρη αλυσίδα blockchain εάν όλες οι συναλλαγές που έχουν προστεθεί στα block της είναι έγκυρες και τα block έχουν παραχθεί με έγκυρο τρόπο σύμφωνα με την απόδειξη $B_{\pi j}$.

Ορισμός 7.2.10 (**Function maxvalid**)**.** Η συνάρτηση *maxvalid* είναι μια βασική συνάρτηση του πρωτοκόλλου που επιτρέπει στους χρήστες να επιλέξουν μια από τις διαθέσιμες αλυσίδες που έχουν δημιουργηθεί από τους χρήστες. Η συνάρτηση αυτή δέχεται σαν ορίσματα την τρέχουσα αλυσίδα του συστήματος C και το σύνολο των έγκυρων αλυσιδών (valid chains) $\mathbb{C}$. Ο κανόνας σύμφωνα με τον οποίο λειτουργεί είναι ότι ο χρήστης διαλέγει την μεγαλύτερη σε μήκος αλυσίδα από τις $\mathbb{C} \cup \{C\}$.

7.2.3 Forward secure signatures

Σε ένα πρωτόκολλο όπως το Ouroboros Praos με έναν αντίπαλο τόσο ισχυρό, είναι δυνατόν οποιαδήποτε στιγμή να διαφθείρει χρήστες. Αυτό σημαίνει πως θα μπορεί να παράξει υπογραφές μηνυμάτων με τα ιδιωτικά κλειδιά τους, αλλά αυτό δεν είναι το μόνο πρόβλημα. Τα μηνύματα που μπορεί να υπογράψει δεν είναι μόνο τα τρέχοντα ή τα μελλοντικά αλλά θα μπορούσε να δημιουργήσει υπογραφές και για μηνύματα του παρελθόντος. Η λύση για αυτό είναι να δημιουργηθούν υπογραφές που θα μπορούν να πιστοποιηθούν σε ποια χρονική στιγμή παράχθηκαν.

Όπως περιγράφεται και στο [17], η ιδέα για τα σχήματα υπογραφών με forward security είναι ότι για έναν χρήστη με δημόσιο κλειδί pk και ιδιωτικό κλειδί sk_0, ορίζεται ένα χρονικό διάστημα από T γύρους στο οποίο το δημόσιο κλειδί παραμένει σταθερό. Το ιδιωτικό κλειδί όμως εξελίσσεται σε κάθε γύρο, με sk_1 ενεργό για τον γύρο 1, ..., sk_T ενεργό για τον γύρο T. Η παραγωγή των νέων ιδιωτικών κλειδιών μπορεί να γίνει με εφαρμογή μιας μονόδρομης συνάρτησης F στο κλειδί του προηγούμενου γύρου. Δηλαδή $sk_t \leftarrow F(sk_{t-1})$. Στην συνέχεια τα ιδιωτικά κλειδιά ενός γύρου που έχει περάσει διαγράφονται. Σχηματικά η διαδικασία παρουσιάζεται από το [17] ως εξής :



Σχήμα 7.7: Εξέλιξη ιδιωτικού κλειδιού σε T περιόδους με εφαρμογή μιας μονόδρομης συνάρτησης h

Το αποτέλεσμα είναι ο $\mathcal{A}$ ακόμη και αν έχει διαφθείρει έναν χρήστη και μπορεί να υπογράψει στην θέση του, να μην μπορεί να παραβιάσει τις προηγούμενες υπογραφές, μόνο τις τρέχουσες και τις μελλοντικές. Ο χρήστης U_i όμως υπό τον έλεγχο του $\mathcal{A}$, σημειώνεται από το σύστημα ως διεφθαρμένος άρα όλοι γνωρίζουν πως τα μηνύματα και τα blocks είναι adversarial. Στο Praos, το σχήμα αυτό των υπογραφών ονομάζεται π_{kes} και δημιουργείται με βάση τις ιδανικές λειτουργίες που προτείνει το πλαίσιο καθολικής συνθεσιμότητας, Universal Composability Framework (UC) [8].

Στο π_{kes} υπάρχουν τέσσερεις αλγόριθμοι για τους χρήστες $U_i, i \in \{1, .., n\}$ που το χρησιμοποιούν :

- **Παραγωγή κλειδιών**: Τα ιδιωτικά κλειδιά παράγονται από τον αλγόριθμο παραγωγής κλειδιών πάντα προς αντιστοιχία με το pk_i και με τον χρόνο k. Άρα για $k = 1$ παράγεται το sk_1 που επαληθεύεται από το pk_i του χρήστη U_i.

- **Υπογραφή και ανανέωση**: Κάθε φορά που απαιτείται μια υπογραφή σε χρόνο k, παράγεται η σ_k, τρέχεται ο αλγόριθμος ανανέωσης για το sk_k, παράγεται το sk_{k+1} και στην συνέχεια το sk_k διαγράφεται και αυξάνεται το k.

- **Επαλήθευση**: Η επαλήθευση για μια υπογραφή σ_k γίνεται μόνο με το δημόσιο κλειδί pk_i του χρήστη U_i, και με τον χρόνο k του μηνύματος που έχει υπογραφτεί.

Η διαδικασία του π_{kes} για αριθμό ανανεώσεων T των υπογραφών και τρέχων χρόνο $1 \leq j \leq T$ όταν ένας χρήστη θέλει να παράξει υπογραφή μπορεί να περιγραφτεί ως εξής :

- Παραγωγή ζεύγους κλειδιών pk, sk_j

- Υπογραφή μηνύματος m κατά τον χρόνο j : $sign_{sk_j}(m) = \sigma_j$

- Ανανέωση sk_j σε sk_{j+1}
- Διαγραφή του sk_j
- Διαμοιρασμός σ_j

Η διαδικασία του π_{kes} για αριθμό ανανεώσεων T των υπογραφών και τρέχων χρόνο $1 \leq j \leq T$ όταν ένας χρήστης θέλει να επαληθεύσει μια υπογραφή μπορεί να περιγραφτεί ως εξής :

Για μια σ_j υπογραφή μηνύματος m από χρήστη με δημόσιο κλειδί pk κατά τον χρόνο j:

- Εάν το pk είναι καταγεγραμμένο από το σύστημα για την σ_j, και η σ_j είναι όντως η υπογραφή του μηνύματος m, επιστρέφεται 1 από την επαλήθευση και ο αλγόριθμος επιτυγχάνει.
- Εάν το pk είναι καταγεγραμμένο από το σύστημα για την σ_j, ο χρήστης με pk δεν είναι διεφθαρμένος αλλά το μήνυμα m δεν αντιστοιχίζεται στην σ_j, επιστρέφεται 0 από την επαλήθευση και ο αλγόριθμος αποτυγχάνει.
- Εάν το pk είναι καταγεγραμμένο από το σύστημα για την σ_j, αλλά $j < k$ όπου k είναι το τρέχων χρόνος, τότε επιστρέφεται 0.

Στο πρωτόκολλο πέρα από τις forward secure υπογραφές με κλειδιά pk^{kes}, sk^{kes}, υπάρχει και ένα σχήμα απλών ηλεκτρονικών υπογραφών που ονομάζεται π_{dsig} με αντίστοιχα κλειδιά pk^{dsig}, sk^{dsig}. Οι forward secure υπογραφές χρησιμοποιούνται στα μηνύματα που είναι κρίσιμο να μην μπορούν να πλαστογραφηθούν, όπως είναι τα μηνύματα δημιουργίας ενός block. Αντιθέτως, τα μηνύματα συναλλαγών μπορούν να υπογράφονται με το απλό σχήμα, με κλειδιά pk^{dsig}, sk^{dsig}, καθώς εφόσον μπουν σε κάποιο block αυτό προστατεύει τις πληροφορίες του με το σχήμα π_{kes}.

7.2.4 Δημιουργία επιτροπής

Η δημιουργία επιτροπής του Ouroboros Classic γινόταν με την βοήθεια μιας συνάρτησης $F(.)$ και του *Follow The Satoshi* αλγόριθμου ώστε να επιλέγονται οι χρήστες που θα είναι slot leaders. Η τυχαιότητα παραγόταν με μια ξεχωριστή διαδικασία από τους slot leaders οι οποίοι έτρεχαν ένα coin tossing πρωτόκολλο με guaranteed output delivery. Στο Ouroboros Praos υπάρχουν τρεις βασικές διαφορές με το κλασικό πρωτόκολλο

:

- Σε κάθε slot έχουν δικαίωμα να εκλεχθούν πάνω από ένας χρήστης.
- Υπάρχει περίπτωση να προκύψει slot χωρίς slot leader.
- Οι slot leaders γνωστοποιούν την ιδιότητά τους μόνο όταν δημοσιεύσουν το block στο slot που έχουν εκλεχθεί.

Το πρωτόκολλο αντί να τρέχει έναν αλγόριθμο παραγωγής τυχαιότητας, χρησιμοποιεί τις κρυπτογραφικές συναρτήσεις Verifiable Random Functions. Το εργαλείο αυτό μπορεί να εκτελεστεί ιδιωτικά και να παράξει ένα ψευδοτυχαίο αλλά επαληθευόμενο αποτέλεσμα για οποιονδήποτε στο πρωτόκολλο. Η επαλήθευση γίνεται με την γνώση του δημοσίου κλειδιού του χρήστη που παρήγαγε την τυχαιότητα και με την απόδειξη π, που παράγεται από τις VRFs για αυτόν τον σκοπό. Επιπλέον το κρυπτογραφικό αυτό εργαλείο παράγει ένα ψευδοτυχαίο αποτέλεσμα που είναι ανθεκτικό ακόμη και όταν ο αντίπαλος $\mathcal{A}$ έχει παράξει τα κλειδιά που χρησιμοποιούνται για την παραγωγή τυχαιότητας (pk^{vrf}, sk^{vrf}). Περισσότερες λεπτομέρειες για τις Verifiable Random Functions υπάρχουν σε αυτό το κεφάλαιο.

Δεδομένης όμως της τυχαιότητας, χρειάζεται και μια συνάρτηση που θα την αξιοποιεί και θα εκλέγει τους slot leaders βάσει του stake τους όπως ορίζεται στα proof-of-stake πρωτόκολλα. Πιο συγκεκριμένα θα πρέπει η πιθανότητα επιλογής p_i, κάθε χρήστη U_i, να είναι ανάλογη του stake που έχει αυτός στην δεδομένη εποχή. Το stake αυτό στην πραγματικότητα είναι το $\alpha_i = \frac{s_i}{\sum_{i=1}^{n} s_i}$, δηλαδή το ποσοστό των χρημάτων του στα συνολικά χρήματα της εποχής του πρωτοκόλλου. Η συνάρτηση αυτή είναι η εξής :

$$p_i = \varphi_f(\alpha_i) = 1 - (1 - f)^{\alpha_i}$$

Ουσιαστικά πρόκειται για μια συνάρτηση που έχει σαν είσοδο το relative stake α_i του χρήστη U_i, και μια παράμετρο f που ονομάζεται συντελεστής ενεργών slot. Ο συντελεστής ενεργών slot συνδέει την πιθανότητα να επιλεχθεί κάποιος χρήστης p_i με το ποσοστό χρημάτων του α_i. Η έξοδος της συνάρτησης είναι η πιθανότητα p_i που καθορίζει την επιλογή του U_i ως slot leader.

Διερεύνηση συνάρτησης $\varphi_f(\alpha_i)$

Η συνάρτηση $\varphi_f(\alpha_i)$ για τιμή $\alpha_i = 1$ δίνει :

$$\varphi_f(1) = 1 - (1 - f)^1 = 1 - 1 + f = f$$

Η τιμή $\alpha_i = 1$ σημαίνει πως ο χρήστης U_i έχει στην κατοχή του όλα τα χρήματα αυτής της εποχής. Άρα η πιθανότητα επιλογής ως slot leader ενός χρήστη που έχει relative stake 1 είναι ακριβώς ο συντελεστής ενεργών slot f.

Επιπλέον μας ενδιαφέρει η συνάρτηση $\varphi_f(\alpha_i)$ να μην επιτρέπει στον αντίπαλο $\mathcal{A}$ να μπορεί να επωφεληθεί από τον διαμοιρασμό χρημάτων σε πολλούς λογαριασμούς με σκοπό την αύξηση της πιθανότητας επιλογής του. Αυτό μαθηματικά εκφράζεται με την ιδιότητα ανεξάρτητου αθροίσματος (aggregation property) :

$$\varphi_f\left(\sum \alpha_i\right) = \sum \varphi_f(\alpha_i)$$

Απόδειξη

Έστω ότι ο U_i έχει χωρίσει τα χρήματα του σε n λογαριασμούς, τότε :

$$1 - \varphi_f\left(\sum_n \alpha_i\right) = (1-f)^{\sum_n \alpha_i} = (1-f)^{\alpha_1 + \ldots + \alpha_n} =$$

$$= (1-f)^{\alpha_1} \cdot \ldots \cdot (1-f)^{\alpha_n} = \prod_n (1-f)^{\alpha_i} = \prod_n 1 - \varphi_f(\alpha_i)$$

□

Άρα ένας χρήστης είτε έχει τα χρήματα του σε έναν λογαριασμό α, είτε τα έχει διαμοιράσει σε 2 λογαριασμούς $\alpha_1 + \alpha_2 = \alpha$, η πιθανότητα να εκλεγεί είναι ίδια σε οποιαδήποτε περίπτωση, σύμφωνα με την παραπάνω σχέση :

$$1 - \varphi_f(\alpha) = \big(1 - \varphi_f(\alpha_1)\big) \cdot \big(1 - \varphi_f(\alpha_2)\big)$$

Τυχαιότητα VRFs και $\varphi_f(\alpha_i)$

Για να συνδυαστούν η τυχαιότητα των VRFs y_i για τον χρήστη U_i και η συνάρτηση εκλογής $\varphi_f(\alpha_i)$, ορίζεται ένα φράγμα στην y_i που περιέχει την $\varphi_f(\alpha_i)$. Έστω το φράγμα αυτό T_i.

Τότε στην περίπτωση που $y_i < T_i$, ο χρήστης U_i είναι slot leader. Η τιμή όμως y_i είναι ομοιόμορφα κατανεμημένη στο διάστημα $[0, 2^{y_i^{length}} - 1)$, ενώ η τιμή $\varphi_f(\alpha_i)$ που πρέπει να την συγκρίνουμε είναι μια πιθανότητα και άρα $\varphi_f(\alpha_i) \in [0,1]$. Συνεπώς κανονικοποιούμε την τιμή y_i ως $\frac{y_i}{2^{y_i^{length}}}$, και τώρα $\frac{y_i}{2^{y_i^{length}}} \in [0,1]$. Εν τέλει θέλουμε :

$$\frac{y_i}{2^{y_i^{length}}} < \varphi_f(\alpha_i) \Rightarrow y_i < 2^{y_i^{length}} \cdot \varphi_f(\alpha_i)$$

Δηλαδή το φράγμα είναι $T_i = 2^{y_i^{length}} \cdot \varphi_f(\alpha_i)$.

7.2.5 Περιγραφή του πρωτοκόλλου σε μια εποχή

Η πιο απλή μορφή του πρωτοκόλλου του Ouroboros Praos είναι η ανάλυση των βημάτων μέσα σε μια εποχή με R slots, όπου η κατανομή των χρημάτων $\mathbb{S}_0$ παραμένει σταθερή. Θεωρείται δηλαδή ότι μέσα στην εποχή δεν υπάρχει αλλαγή στο stake distribution, και δεν υπάρχει stake shift. Οι πληροφορίες για την κατανομή των χρημάτων είναι αποθηκευμένες μέσα στο blockchain, στο πρώτο slot της εποχής. Εκεί είναι αποθηκευμένη και η αρχική τυχαιότητα nonce η, που θα χρησιμοποιηθεί από τις Verifiable Random Functions για την εκλογή των αρχηγών.

Αρχικοποίηση

Κατα την αρχικοποίηση του βασικού πρωτοκόλλου, ο κάθε χρήστης φροντίζει να αποκτήσει τις απαραίτητες πληροφορίες ώστε να προετοιμαστεί για το στάδιο επέκτασης της αλυσίδας. Πρώτα λαμβάνει από το περιβάλλον τα ζευγάρια δημοσίων και ιδιωτικών κλειδιών του για τις VRFs (pk^{vrf}, sk^{vrf}), για τις forward security υπογραφές (pk^{kes}, sk^{kes}), και για τις κανονικές υπογραφές (pk^{dsig}, sk^{dsig}). Στην συνέχεια ενημερώνεται για το τρέχων slot που κάνει την αρχικοποίηση. Εάν είναι το πρώτο slot sl_1, λαμβάνει το genesis block $genblock$, την κατανομή των stake $\mathbb{S}_0$, τον σπόρο τυχαιότητας η και κρατάει ένα τοπικό blockchain $C = B_0 = \{\mathbb{S}_0, \eta\}$. Αν το τρέχων slot δεν είναι το πρώτο, λαμβάνει την αλυσίδα από το σύστημα και φτιάχνει τοπικό blockchain C . Στην πρώτη περίπτωση ορίζει $st = H(B_0)$, αλλιώς $st = H(head(C))$.
Με λίγα λόγια κατα την αρχικοποίηση προσδιορίζονται τα εξής :

1. Ζευγάρια κλειδιών VRF και των υπογραφών
2. Τρέχων slot, κατανομή χρημάτων και τυχαιότητα
3. Τοπικό Blockchain
4. Κατάσταση state

Επέκταση Αλυσίδας

Αφού ο χρήστης έχει δημιουργήσει μια τοπική αλυσίδα και έχει όλες τις απαραίτητες πληροφορίες, μπαίνει στην διαδικασία επέκτασης της αλυσίδας. Πρόκειται για μια επαναλαμβανόμενη διαδικασία που συμβαίνει R φορές, όσα και τα slot sl_j της εποχής, και πραγματοποιείται από κάθε ενεργό χρήστη, είτε είναι slot leader είτε όχι.
Ο χρήστης λαμβάνει τις συναλλαγές $d \in \{0,1\}^*$ που πρόκειται να βάλει σε κάποιο block του blockchain. Στην συνέχεια ενημερώνεται για όλες τις νέες αλυσίδες που υπάρχουν διαθέσιμες, και ορίζει ως νέα αλυσίδα αυτή που προκύπτει από την συνάρτηση maxvalid$(\mathbb{C}, C)$, εφόσον επαληθεύονται οι forward security υπογραφές των block που περιέχει και οι slot leader από τις VRFs με το pk^{vrf} και την απόδειξη π, και ότι

$vrf_{output} = y_i < T_i$. Τότε C'= maxvalid$(\mathbb{C}, C)$ και C' είναι η καινούρια τοπική αλυσίδα, και $st = H(head(C'))$. Στην συνέχεια, στην περίπτωση που οι VRFs με το $\eta||sl_j$ επιστρέψουν μια τιμή y_i που είναι μικρότερη του φράγματος $T_i = 2^{y_i^{length}} \cdot \varphi_f(\alpha_i)$ του χρήστη U_i , φτιάχνει το $B = (st, d, sl_j, B_\pi, \sigma)$ όπου d οι συναλλαγές, $B_\pi = (U_i, y, \pi)$ οι πληροφορίες για την επαλήθευση και σ η υπογραφή $\sigma = Sign_{sk^{kes}}(st, d, sl_j, B_\pi)$. Δημιουργεί την καινούργια τοπική αλυσίδα $C'' = C'|B$ και την μεταδίδει στο σύστημα. Επίσης ορίζεται $st = H(head(C''))$.

Με λίγα λόγια κατα την επέκταση αλυσίδας έχουμε τα βήματα :

1. Δεδομένα συναλλαγών
2. Συλλογή έγκυρων αλυσιδών (σωστές forward security υπογραφές, σωστά στοιχεία από τις VRFs, $y_i < T_i$)
3. Εφαρμογή maxvalid()
4. Ενημέρωση τοπικής αλυσίδας και κατάστασης
5. Έλεγχος VRFs
6. Για θετική απάντηση :
   - Δημιουργία καινούριου block
   - Δημιουργία καινούριας αλυσίδας με το block
   - Μετάδοση καινούριας αλυσίδας
   - Ενημέρωση τοπικής αλυσίδας και κατάστασης

Παραγωγή συναλλαγών

Σε αυτό το στάδιο του πρωτοκόλλου ο χρήστης μπορεί να παράξει συναλλαγές, σύμφωνα πάντα με ένα υπόδειγμα έγκυρης συναλλαγής. Για αυτόν τον λόγο δημιουργεί και μια απλή υπογραφή με το pk^{dsig} μαζί με την συναλλαγή tx, και στην συνέχεια αποστέλνει το ζευγάρι (tx, σ) στο περιβάλλον.

7.2.6 Ανάλυση ασφάλειας σε μια εποχή

Το Ouroboros Praos διαφέρει σε δύο βασικά κομμάτια με το Ouroboros. Πρώτα από όλα, έχει άλλο μηχανισμό για επιλογή slot leader, που επιτρέπει σε ένα slot να ανατεθεί ένας αρχηγός, πολλαπλοί, ή και κανένας. Αυτό συμβαίνει διότι η εκλογή δεν βασίζεται σε μια συνάρτηση $F(.)$ που διαλέγει ένα pk και το επιστρέφει, αλλά εκτελείται ιδιωτικά από τον κάθε χρήστη και η τυχαία τιμή των *VRFs* ελέγχεται αν είναι μικρότερη από μια τιμή T_i. Συνεπώς υπάρχει περίπτωση πάνω από ενας χρήστης να επιλεχθεί για κάποιο slot. Επιπλέον, το πρωτόκολλο έχει υπόθεση μερικού συγχρονισμού με παράμετρο Δ, άρα ο $\mathcal{A}$ μπορεί να καθυστερήσει την επικοινωνία

ειλικρινών χρηστών μέχρι Δ slots. Αυτές οι αλλαγές πρέπει να εκφράζονται στην ανάλυση της ασφάλειας των ιδιοτήτων του Ouroboros Praos blockchain.

Σύμφωνα με τα παραπάνω, αφού είναι φυσικό να υπάρξουν πάνω από ένας slot leader για ένα sl_j, έχουμε δύο πιθανές περιπτώσεις δημιουργίας fork. Είτε από χρήστη διεφθαρμένο που ο αντίπαλος δρα στην θέση του και δημιουργεί πολλαπλά blocks σκοπίμως, είτε από ειλικρινείς χρήστες που απλά έτυχε να εκλεχθούν ως slot leaders σε ίδιο slot και όλοι ήταν ενεργοί ώστε να προτείνουν κάποιο block. Αυτό θα πρέπει να φαίνεται στο χαρακτηριστικό string της αλυσίδας, γι'αυτό τον λόγο δίνεται ένας καινούριος ορισμός σε σχέση με το Ouroboros Classic.

Ορισμός 7.2.11 (**Χαρακτηριστική συμβολοσειρά**)**.** Έστω μια αλυσίδα C του συστήματος, με genesis block B_0, και ένας αντίπαλος $\mathcal{A}$. Έστω $S = \{sl_1, ..., sl_R\}$ μια ακολουθία από slot μήκους $length|S| = R$. Έστω $P(j)$ το σύνολο των χρηστών που εκλέχθηκαν από το πρωτόκολλο ως slot leaders για το slot sl_j. Η χαρακτηριστική συμβολοσειρά $\omega \in \{0, 1, \perp\}^R$ του S είναι η τυχαία μεταβλητή που περιγράφεται ως:

$$\omega_j = \begin{cases} \perp, & \text{εάν } P(j) = \varnothing \\ 0, & \text{εάν } |P(j)| = 1 \text{ και οι slot leaders είναι ειλικρινείς} \\ 1, & \text{εάν } |P(j)| > 1 \text{ ή έστω ένας slot leader στο} P(j) \text{είναι διεφθαρμένος} \end{cases}$$

Για μια τέτοια χαρακτηριστική συμβολοσειρά $\omega \in \{0, 1, \perp\}^*$, το $\omega_j = 0$ έχει ειλικρινές περιεχόμενο, το $\omega_j = 1$ είναι διεφθαρμένο, ενώ το $\omega_j = \perp$ δεν έχει περιεχόμενο, είναι κενό. Επιπλέον, εάν $\omega_j = \{0, 1\}$ τότε το slot θεωρείται ενεργό, ενώ εάν $\omega_j = \perp$ θεωρείται ανενεργό (δεν υπήρχε ενεργός χρήστης να αναθέσει block).
Η κατανομή της τυχαίας μεταβλητής $\omega = \omega_1, ..., \omega_R$ συμβολίζεται $D^f_{\mathcal{Z},\mathcal{A}}$, για περιβάλλον $\mathcal{Z}$, αντίπαλο $\mathcal{A}$, και συντελεστή ενεργών slot f.

Επιπλέον, η χαλάρωση του συγχρονισμού του δικτύου, από συγχρονισμένο με καθυστέρηση διαφθοράς και εγγυημένη επικοινωνία μεταξύ ειλικρινών χρηστών, σε μερικώς συγχρονισμένο με άμεση διαφθορά και καθυστέρηση επικοινωνία μεταξύ ειλικρινών χρηστών Δ, δημιουργεί καινούριου είδους απαιτήσεις για την αλυσίδα. Αναλυτικότερα, θα θέλαμε το βάθος στην αλυσίδα κάποιου block που έχει φτιαχτεί από ειλικρνές χρήστη, να ξεπερνάει το βάθος οποιωνδήποτε ειλικρινών block τουλάχιστον Δ στο παρελθόν. Αυτό συμβαίνει διότι ένας ειλικρινής χρήστης, θα έχει μέγιστη καθυστέρηση Δ στα μηνύματα που δέχεται, και άρα Δ slots πριν από ένα ειλικρινές block σίγουρα υπήρχε κάποιο άλλο ειλικρινές block.

Ορισμός 7.2.12 (**Δ-fork**). Έστω $\omega \in \{0,1,\perp\}^k$ και Δ ένας μη αρνητικός ακέραιος. Τότε έστω $A = \{i|\omega_i \neq \perp\}$ το σύνολο των ενεργών block, $H = \{i|\omega_i = 0\}$ το σύνολο των ειλικρινών block. Ενα Δ-fork μιας χαρακτηριστικής συμβολοσειράς ω, είναι ένα κατευθυνόμενο δέντρο με ρίζα $F = (V, E)$ με ετικέτες $\ell : V \to \{0\} \cup A$ ώστε :

1. Η ρίζα $r \in V$ έχει ετικέτα $\ell(r) = 0$

2. Κάθε ακμή του δέντρου F κατευθύνεται μακρυά από την ρίζα

3. Οι ετικέτες κατα μήκος ενός κατευθυνόμενου μονοπατιού αυξάνονται αυστηρά

4. Κάθε μοναδικό ειλικρινές block $i \in H$ είναι η ετικέτα ακριβώς ενός κόμβου του F

5. Η συνάρτηση $\mathbf{d} : H \to \{1, ..., k\}$ που ορίζεται ως το βάθος ενός κόμβου v στο F με ετικέτα $\ell(v) = i$ ικανοποιεί την ιδιότητα *Δ-μονοτονίας* : εάν $i, j \in H$ και $i + \Delta < j$ τότε $\mathbf{d}(i) < \mathbf{d}(j)$)

Συμβολίζουμε ως $F \vdash_\Delta \omega$ το F που είναι ένα Δ-fork για την συμβολοσειρά ω. Το Δ-fork θα αναφέρεται ως fork.

Ορισμός 7.2.13 (**Tine, length, viability**). Ένα μονοπάτι μιας διακλάδωσης (fork) που ξεκινά από την ρίζα του F ονομάζεται *tine*. Το μήκος ενός *tine* t είναι οι ακμές που έχει το μονοπάτι αυτό και συμβολίζεται ως $length(t)$. Βάθος ενός κόμβου v ονομάζεται το μήκος του μοναδικού *tine* που καταλήγει σε αυτόν τον κόμβο και συμβολίζεται ως $depth(v)$. Η χρήση της ετικέτας ℓ μπορεί να χρησιμοποιηθεί και για τα tines, και $\ell(t) = \ell(v)$, δηλαδή η ετικέτα ενός tine είναι το τελευταίο του block v. Ένα tine t του F λέγεται $\Delta - viable$ εάν $\ell(t) \geq max_{h+\Delta \leq \ell(t)} \mathbf{d}(h)$ για όλα τα ειλικρινά block που εμφανίζονται Δ ή περισσότερα slot πριν το $\ell(t)$.

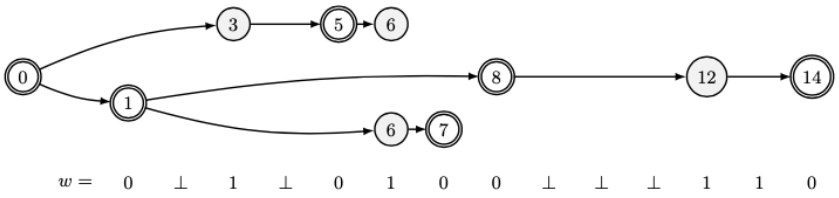
Ορισμός 7.2.14 (**Divergence**). Έστω F ένα Δ-fork μιας συμβολοσειράς $\omega \in \{0,1,\perp\}^*$. Για δύο $\Delta - viable$ tines t_1, t_2, το *divergence* μεταξύ τους είναι :

$$div(t_1, t_2) = min\{length(t_1), length(t_2)\} - length(t_1 \cap t_2)$$

όπου $(t_1 \cap t_2)$ είναι το κοινό πρόθημα των t_1, t_2. Για το *divergence* ενός fork ισχύει : $div_\Delta(F) = max_{t_1,t_2} div(t_1, t_2)$ για κάθε ζευγάρι t_1, t_2 του F. Για το *divergence* ενός χαρακτηριστικού string ω ισχύει : $div_\Delta(\omega) = max_{F \vdash_\Delta \omega} div_\Delta(F)$.

Ουσιαστικά το *divergence* είναι η ελάχιστη διαφορά στο μήκος δύο μονοπατιών μετά το κοινό τους πρόθημα. Για να ισχύει η ιδιότητα common prefix με παράμετρο k, θα πρέπει μεταξύ δύο μονοπατιών $t, \hat{t}$ με $length(t) <$

$length(\hat{t})$ να έχουμε $\hat{t}^{\lceil k} \preceq t$. Δηλαδή το t να είναι πρόθημα του $\hat{t}$. Αυτό σημαίνει πως $div(t, \hat{t}) = k$ για αυτή την περίπτωση, και γενικά για να ισχύει η ιδιότητα $k - CP$ θα πρέπει $div(\omega) \leq k$. Άρα για να έχουμε την ιδιότητα common prefix θα πρέπει να έχουμε και μικρό divergence για το χαρακτηριστικό string ω.



Στην παραπάνω εικόνα υπάρχει ένα fork σε ένα μερικώς συγχρονισμένο δίκτυο του Ouroboros Praos. Το χαρακτηριστικό string του fork είναι $\omega = 0 \perp 1 \perp 0100 \perp\perp\perp 110$. Τα ειλικρινή slot είναι τα {1,5,7,8,14}, τα μη ειλικρινή είναι τα {3,6,12} και τα κενά είναι {2,4,9,10,11}. Για να βρούμε την καθυστέρηση Δ παίρνουμε το βάθος των ειλικρινών slot και συγκρίνουμε. Θα πρέπει εάν $i, j \in H$ και $i + \Delta < j$ τότε $\mathbf{d}(i) < \mathbf{d}(j)$). Έχουμε $\mathbf{d}(1) = 1, \mathbf{d}(5) = 2, \mathbf{d}(7) = 3, \mathbf{d}(8) = 2, \mathbf{d}(14) = 4$. Για το $1 + 2 < 5$ έχουμε $\mathbf{d}(1) < \mathbf{d}(5)$. Για το $5 + 2 < 8$ έχουμε $\mathbf{d}(5) = \mathbf{d}(8)$. Άρα το $\Delta \neq 2$. Για το $1+3 < 5$ έχουμε $\mathbf{d}(1) < \mathbf{d}(5)$. Για το $8+3 < 14$ έχουμε $\mathbf{d}(8) < \mathbf{d}(14)$. Άρα έχουμε ένα $3 - fork$, δηλαδή $\Delta = 3$.

Θεώρημα 7.2.1 (**Συγχρονισμένη Περίπτωση**). *Έστω $k, \ell \in \mathbb{N}$ και $\epsilon \in (0, 1)$. Έστω $\omega \in \{0, 1\}^{\ell}$ που έχει προκύψει από την διωνυμική κατανομή με $Pr[\omega_i = 1] = \frac{(1-\epsilon)}{2}$. Τότε $Pr[div_0(\omega) \geq k] \leq exp(ln\ell - \Omega(k))$.*

Εκμεταλλευόμενοι το γεγονός ότι η ιδιότητα common prefix έχει αποδειχθεί για το συγχρονισμένο πρωτόκολλο του Ouroboros Classic [1], δηλαδή στην περίπτωση που δεν έχουμε καθυστέρηση παράδοσης μηνυμάτων και άρα $\Delta = 0$, θα γίνει μια αναγωγή απεικόνισης (reduction mapping) από τις χαρακτηριστικές συμβολοσειρές του Ouroboros Praos σε αυτές του Ouroboros Classic.

Ορισμός 7.2.15 (**Αναγωγή απεικόνισης**). Έστω $\Delta \in \mathbb{N}$. Τότε ορίζεται η συνάρτηση $\rho_{\Delta} : \{0, 1, \perp\}^* \rightarrow \{0, 1\}^*$ ως :

$$\rho_{\Delta}(\varepsilon) = \varepsilon$$

$$\rho_{\Delta}(\perp ||\omega') = \rho_{\Delta}(\omega')$$

$$\rho_\Delta(1||\omega') = 1||\rho_\Delta(\omega')$$

$$\rho_\Delta(0||\omega') = \begin{cases} 0||\rho_\Delta(\omega'), & \text{εάν } \omega' \in \perp^{\Delta-1} ||\{0,1,\perp\}^* \\ 1||\rho_\Delta(\omega'), & \text{αλλιώς} \end{cases}$$

Η συνάρτηση ρ_Δ ονομάζεται η αναγωγή απεικόνισης για παράμετρο καθυστέρησης Δ.

Λήμμα 7.2.1. Έστω $\omega \in \{0,1,\perp\}^*$, τότε $div_\Delta(\omega) \leq div_0(\rho_\Delta(\omega))$.

Θέλουμε να προσδιορίσουμε την κατανομή $\mathcal{D}^f_{\mathcal{Z},\mathcal{A}}$ των χαρακτηριστικών string $\omega \in \{0,1,\perp\}^R$, η οποία επηρεάζεται από τον αντίπαλο (A), το περιβάλλον του πρωτοκόλλου $\mathcal{Z}$, και τον συντελεστή ενεργών slot f. Έστω ένας αντίπαλος (A) με stake $\alpha_\mathcal{A}$, που δρα σε μια εποχή. Τότε η κατανομή των χαρακτηριστικών string επηρεάζεται από τον $\mathcal{A}$ καθώς αυτός αυξάνει την συμβολοσειρά προσθέτοντας block με τις εξής πιθανότητες :

$$p^\mathcal{A}_\perp = Pr[\omega_i = \perp] = \prod_{i\in\mathcal{P}}(1-\varphi(\alpha_i)) = \prod_{i\in\mathcal{P}}(1-f)^{\alpha_i} = (1-f)$$

$$p^\mathcal{A}_0 = Pr[\omega_i = 0] = \sum_{h\in\mathcal{H}}(1-(1-f)^{\alpha_h})\cdot(1-f)^{1-\alpha_i}$$

$$p^\mathcal{A}_1 = Pr[\omega_i = 1] = 1 - p^\mathcal{A}_\perp - p^\mathcal{A}_0$$

όπου $\mathcal{P}$ είναι το σύνολο των slot leaders και $\mathcal{H}$ είναι το σύνολο των ειλικρινών slot leaders.

Εφαρμόζοντας τις ιδιότητες της συνάρτησης $\varphi(\alpha_i)$ έχουμε :

$$p^\mathcal{A}_0 \geq \sum_{h\in\mathcal{H}}\varphi(\alpha_h)\cdot\prod_{i\in\mathcal{P}}(1-\varphi(\alpha_i)) \geq \varphi(\alpha_\mathcal{H})\cdot p^\mathcal{A}_\perp = \varphi(\alpha_\mathcal{H})\cdot(1-f)$$

Άρα η πιθανότητα να προστεθεί ένα ειλικρινές block στην συμβολοσειρά ω παρουσία αντιπάλου $\mathcal{A}$, έχει ελάχιστη τιμή $\varphi(\alpha_\mathcal{H})\cdot(1-f)$. Άρα η κατανομή με αυτή την τιμή για το $p^\mathcal{A}_0$ και τις υπόλοιπες πιθανότητες ίδιες :

$$p^\mathcal{A}_\perp = (1-f)$$

$$p_0^{\mathcal{A}} = Pr[\omega_i = 0] = \varphi(\alpha_{\mathcal{H}}) \cdot (1 - f)$$

$$p_1^{\mathcal{A}} = Pr[\omega_i = 1] = 1 - p_{\perp}^{\mathcal{A}} - p_0^{\mathcal{A}}$$

θα δειχθεί ότι εκφράζει καλύτερα την επιρροή του αντιπάλου στην δημιουργία των χαρακτηριστικών συμβολοσειρών.

Ορισμός 7.2.16 (**Κατανομή** $\mathcal{D}_{\alpha}^{f}$)**.** Έστω οι παράμετροι f και α, τότε η κατανομή $\mathcal{D}_{\alpha}^{f}$ που αντιπροσωπεύει την προσθήκη κάθε ω_i στα χαρακτηριστικά string $\omega \in \{0, 1, \perp\}^R$ περιγράφεται ως $p_{\perp} = (1 - f)$, $p_0 = Pr[\omega_i = 0] = \varphi(\alpha) \cdot (1 - f)$ και $p_1 = Pr[\omega_i = 1] = 1 - p_{\perp} - p_0$.

Λήμμα 7.2.2. Έστω x, y δύο χαρακτηριστικές συμβολοσειρές στο$\{0, 1, \perp\}^R$ ώστε $x \preceq y$. Τότε :

1. Για κάθε fork F, $F \vdash_{\Delta} x \Rightarrow F \vdash_{\Delta} y$.
2. Για κάθε Δ, $div_{\Delta}(x) \leq div_{\Delta}(y)$.

Ορισμός 7.2.17 (**Στοχαστική κυριαρχία**)**.** Ένα υποσύνολο $E \subseteq \{0, 1, \perp\}^R$ είναι μονότονο εάν για $x \in E$ και $x \preceq y$ ισχύει και $y \in E$. Έστω $\mathcal{D}$ και $\mathcal{D}'$ δύο κατανομές στο σύνολο των χαρακτηριστικών string $\{0, 1, \perp\}^R$. Τότε λέμε ότι η $\mathcal{D}'$ κυριαρχεί την $\mathcal{D}$, δηλαδή $\mathcal{D} \preceq \mathcal{D}'$, εάν $Pr_{\mathcal{D}}[E] \leq Pr_{\mathcal{D}'}[E]$ για κάθε μονότονο σύνολο E. Ένας αντίπαλος $\mathcal{A}$ λέγεται α − *dominated* εάν για την κατανομή $\mathcal{D}_{\mathcal{Z},\mathcal{A}}^{f}$ ισχύει $\mathcal{D}_{\mathcal{Z},\mathcal{A}}^{f} \preceq \mathcal{D}_{\alpha}^{f}$.

Τελικά στα υποσύνολα που θέλουμε να μελετήσουμε, με τις επικίνδυνες συμβολοσειρές x ώστε: $\mathcal{D}_{\cdot} = \{ x | div_{\Delta}(x) \geq k \}$, είναι μονότονα σύμφωνα με το παραπάνω λήμμα. Άρα η $\mathcal{D}_{\alpha}^{f}$ είναι κυρίαρχη κατανομή της $\mathcal{D}_{\mathcal{Z},\mathcal{A}}^{f}$ και $Pr_{\mathcal{D}_{\mathcal{Z},\mathcal{A}}^{f}}[div_{\Delta}(\omega) \geq k] \leq Pr_{\mathcal{D}_{\alpha}^{f}}[div_{\Delta}(\omega) \geq k]$. Έτσι μπορεί να διερευνηθεί η $\mathcal{D}_{\alpha}^{f}$ αντί για την $\mathcal{D}_{\mathcal{Z},\mathcal{A}}^{f}$. Η κυρίαρχη αυτή κατανομή με την ρ_{Δ} φτιάχνει μια οικογένεια μεταβλητών $\rho_{\Delta}(\omega) = x_1 ... x_{\ell} \in \{0, 1\}^*$ και $\omega \in \{0, 1, \perp\}^R$, με ℓ να είναι ο αριθμός των ενεργών block στο .

Στόχος είναι να εφαρμοστεί ένα άνω όριο στην πιθανότητα ένα string ω της κυρίαρχης κατανομής $\mathcal{D}_{\alpha}^{f}$ να έχει μεγάλο $\Delta - divergence$. Θα μπορούσαμε να εφαρμόσουμε το αποτέλεσμα της συγχρονισμένης περίπτωσης του Ouroboros Classic, αλλά η κυρίαρχη κατανομή δεν είναι απλή διωνυμική, οπότε δεν μπορεί να εφαρμοστεί κατευθείαν. Στο επόμενο λήμμα που έχει αποδειχθεί στο [2], δείχνεται ότι η αναγωγή απεικόνισης ρ_{Δ} με συμβολοσειρές της κατανομής $\mathcal{D}_{\alpha}^{f}$ έχει κατάλληλη δομή ώστε να χρησιμοποιηθεί από το θεώρημα Θεώρημα 7.2.1 συγχρονισμένης περίπτωσης και το παραπάνω Λήμμα 7.2.1 για την ζητούμενη απόδειξη ασφάλειας.

Λήμμα 7.2.3. Έστω $\rho_\Delta(\omega) = x_1...x_\ell$ με $\omega \in \{0,1,\perp\}^R$ κατανομής $\mathcal{D}_\alpha^f$. Τότε υπάρχει μια ακολουθία ανεξάρτητων τυχαίων κατανομών $z_1, z_2, ...$ με $z_i \in \{0,1\}$ ώστε :

$$Pr[z_i = 0] = \left(\frac{p_0}{p_0 + p_1}\right) p_\perp^{\Delta-1} \geq \alpha \cdot (1-f)^\Delta$$

και η ακολουθία $x_1...x_{\ell-\Delta} = \rho_\Delta(\omega_1, ..., \omega_R)^{\vdash\Delta}$ να είναι ένα πρόθημα της $z_1, z_2, ...$

Θεώρημα 7.2.2. *Έστω $f \in (0,1], \Delta \geq 1$ και α ώστε $\alpha(1-f)^\Delta = \frac{1+\varepsilon}{2}$ για κάποιο $\varepsilon > 0$. Έστω $\omega \in \{0,1,\perp\}^R$ σύμφωνα με την $\mathcal{D}_\alpha^f$. Άρα $Pr[div_\Delta(\omega) \geq k + \Delta] = 2^{-\Omega(k)+logR}$.*

Απόδειξη
Η $div_0(.)$ είναι μονότονη αφού για μια συμβολοσειρά y' που είναι πρόθημα της y, τότε $div_0(y') \leq div_0(y)$ διότι κάποιο fork της- y' μπορεί να γίνει fork της y, με όλα τα μονοπάτια της y'. Αντίστοιχα ένα fork της y μπορεί να μετατραπεί σε ένα fork της y' με την αφαίρεση κάποιων κόμβων, έστω s. Άρα ισχύει η ιδιότητα για $|y| \leq |y'| + s \Rightarrow div_0(y) \leq div_0(y') + s$. Σύμφωνα με το Λήμμα 7.2.1 έχουμε :

$$div_\Delta(\omega) \leq div_0(\rho_\Delta(\omega)) \leq div_0(\rho_\Delta(\omega)^{\vdash\Delta}) + \Delta \leq div_0(z_1, ..., z_R) + \Delta$$

Επειδή οι τυχαίες μεταβλητές $z_1, ..., z_R$ είναι διωνυμικές με $Pr[z_i = 0] \geq \alpha \cdot (1-f)^\Delta$, και από υπόθεση $\alpha(1-f)^\Delta = \frac{1+\varepsilon}{2} \Rightarrow Pr[z_i = 0] \geq \frac{1+\varepsilon}{2} \Rightarrow Pr[z_i = 1] \leq \frac{1-\varepsilon}{2}$. Άρα από Θεώρημα 7.2.1 έχουμε ότι $Pr[div_0(\omega) \geq k] \leq exp(ln\ell - \Omega(k)) \Rightarrow Pr[div_\Delta(\omega) \geq k + \Delta] \leq 2^{logR - \Omega(k)}$.

□

Τα παραπάνω βοηθάνε στην απόδειξη των τριών βασικών ιδιοτήτων της αλυσίδας του Ouroboros Praos, που περιγράφονται παρακάτω. Οι αποδείξεις υπάρχουν αναλυτικά στο white paper του Praos [2].

Θεώρημα 7.2.3 (**Common Prefx**). *Έστω $k, R, \Delta \in \mathbb{N}$ και $\varepsilon \in (0,1)$. Έστω $\mathcal{A}$ ένας α – dominated αντίπαλος κατά του πρωτοκόλλου του Ouroboros Praos για κάποιο α με $\alpha(1-f)^\Delta \geq \frac{1+\varepsilon}{2}$. Τότε η πιθανότητα ο $\mathcal{A}$ σε ένα Δ-μερικώς συγχρονισμένο περιβάλλον να παραβιάσει την ιδιότητα κοινού προθήματος με παράμετρο k σε μια περίοδο από R slot, δεν ξεπερνά το $exp(lnR + \Delta - \Omega(k))$.*

Θεώρημα 7.2.4 (**Chain Growth**). *Έστω $k, R, \Delta \in \mathbb{N}$ και $\varepsilon \in (0,1)$. Έστω $\mathcal{A}$ ένας α – dominated αντίπαλος κατά του πρωτοκόλλου του Ouroboros Praos για κάποιο $\alpha > 0$ με $\alpha(1-f)^\Delta \geq \frac{1+\varepsilon}{2}$. Τότε η πιθανότητα ο $\mathcal{A}$ σε ένα Δ-μερικώς συγχρονισμένο περιβάλλον να παραβιάσει την ιδιότητα αύξησης αλυσίδας με παραμέτρους $s \geq 4\Delta$ και $\tau = \frac{c\alpha}{4}$ κατα την περίοδο R slot, δεν ξεπερνά το $exp\frac{-csa}{20\Delta} + lnR\Delta + O(1)$ όπου $c = c(f, \Delta) = f(1-f)^\Delta$.*

Θεώρημα 7.2.5 (**Chain Quality**). *Έστω* $k, R, \Delta \in \mathbb{N}$ *και* $\varepsilon \in (0,1)$. *Έστω* $\mathcal{A}$ *ένας* α *– dominated αντίπαλος κατά του πρωτοκόλλου του Ouroboros Praos για κάποιο* $\alpha > 0$ *με* $\alpha(1-f)^{\Delta} \geq \frac{1+\varepsilon}{2}$. *Τότε η πιθανότητα ο* $\mathcal{A}$ *σε ένα* Δ*-μερικώς συγχρονισμένο περιβάλλον να παραβιάσει την ιδιότητα ποιότητας αλυσίδας με παραμέτρους* k *και* $\mu = \frac{1}{k}$ *σε μια περίοδο από* R *slot, δεν ξεπερνά το* $exp(lnR - \Omega(k))$.

7.2.7 Μετάβαση στο δυναμικό πρωτόκολλο

Στην προηγούμενη ενότητα ανάλυσης ασφάλειας στο στατικό πρωτόκολλο μιας εποχής, αποδείχθηκε πως αυτό είναι ασφαλές σύμφωνα με τις ιδιότητες common prefix, chain growth και chain quality, εάν ο στατικός αντίπαλος είναι α *– dominated* για μια τιμή $\alpha > 0$. Δηλαδή εάν ο αντίπαλος $\mathcal{A}$ έχει διαφθείρει χρήστες που κατέχουν το πολύ κλάσμα $(1-\alpha)$ των συνολικών χρημάτων. Αυτό θα πρέπει να ισχύει και για το δυναμικό πρωτόκολλο.

Θεώρημα 7.2.6. *Κάθε αντίπαλος* $\mathcal{A}$ *που επιτίθεται στο δυναμικό πρωτόκολλο και διαφθείρει το πολύ κλάσμα* $(1-\alpha)$ *των συνολικών χρημάτων κατα την συνολική εκτέλεση, είναι ένας* α *– dominated αντίπαλος.*

Απόδειξη
Έστω η οικογένεια ανεξάρτητων τυχαίων μεταβλητών $c_{i,t}$ με $1 \leq i \leq C$ και $1 \leq t \leq R$. Οι μεταβλητές αυτές περιγράφονται ως :

$$c_{i,t} = \begin{cases} 1 & \text{με} \quad \varphi_f(1/C) = 1-(1-f)^{1/C} \\ 0 & \text{αλλιώς} \end{cases}$$

Κάθε $c_{i,t}$ ουσιαστικά είναι ένα νόμισμα από τα C που αντιστοιχίζεται σε ένα slot κατα την διάρκεια του slot leader election. Κάθε χρήστης που εκλέγεται για slot leader έχει $c_{i,t} = 1$. Λόγω της ιδιότητας ανεξάρτητου αθροίσματος της $\varphi_f(.)$, η διαδικασία slot leader election μπορεί να γίνει με την ανεξάρτητη δειγματοληψία μιας τυχαίας μεταβλητής $c_{i,t}$.
Έστω ότι ο αντίπαλος δεν διαφθείρει χρήστες αλλά νομίσματα σύμφωνα με την παραπάνω διαδικασία, και ονομάζεται $\mathcal{A}_1$. Αφού η τυχαία δειγματοληψία ενός $c_{i,t}$ είναι το ίδιο με την τυχαία δειγματοληψία ενός χρήστη με χρήματα α_i με πιθανότητα επιτυχίας $\varphi_f(\alpha_i) = 1-(1-f)^{\alpha_i}$, η κατανομή των χαρακτηριστικών string σε μια εποχή δεν επηρεάζεται από αυτόν τον αντίπαλο και άρα $\mathcal{D}^f_{\mathcal{Z},\mathcal{A}} = \mathcal{D}^f_{\mathcal{Z},\mathcal{A}_1}$. Όταν ο $\mathcal{A}_1$ διαφθείρει ένα νόμισμα $c_{i,t}$ τότε μαθαίνει και εάν αυτό έχει επιλεχθεί σε κάποιο άλλο slot $\in \{1,..,R\}$.
Έστω ότι υπάρχει ένας άλλος αντίπαλος που διαφθείρει νομίσματα και ονομάζεται $\mathcal{A}_2$. Τότε αυτοί μπορούν να διαφθείρουν ταυτόχρονα αλλά με την προϋπόθεση τα νομίσματα που διαφθείρει ο $\mathcal{A}_1$ να μην είναι αυτά που διαφθείρει ο $\mathcal{A}_2$ και αντίστροφα. Το μόνο που αλλάζει είναι ότι οι

αντίπαλοι πλέον δεν γνωρίζουν εάν τα νομίσματα που έχουν διαφθείρει έχουν επιλεχθεί σε κάποιο άλλο slot $\in \{1, .., R\}$. Άρα δεν αλλάζει κάτι από την πλευρά του περιβάλλοντος ή της παραμέτρου f και άρα $\mathcal{D}^f_{\mathcal{Z},\mathcal{A}_1} = \mathcal{D}^f_{\mathcal{Z},\mathcal{A}_2}$.
Έστω ένας τρίτος αντίπαλος $\mathcal{A}_3$ που διαφθείρει τα πρώτα $(1-\alpha)\cdot C$ νομίσματα και λειτουργεί όπως ο $\mathcal{A}_2$. Τότε η μόνη διαφορά με πριν είναι ότι τα νομίσματα που έχει διαφθείρει ο $\mathcal{A}_3$ μπορεί να είναι μέρος των νομισμάτων που θα διαφθείρει ο $\mathcal{A}_2$. Άρα $Pr[\omega_2 \preceq \omega_3] = 1$, όπου ω_i είναι το χαρακτηριστικό string που έχει προκύψει από την εκτέλεση του πωρτοκόλλου για τον αντίπαλο $\mathcal{A}_i$. Άρα $\mathcal{D}^f_{\mathcal{Z},\mathcal{A}_2} \preceq \mathcal{D}^f_{\mathcal{Z},\mathcal{A}_3}$ [18]
Άρα πλέον έχουμε έναν στατικό αντίπαλο $\mathcal{A}_3$ που διαφθείρει $1-\alpha$ κλάσμα των χρημάτων και δρα ταυτόχρονα με κάποιον άλλο μέσα στο πρωτόκολλο αλλά ισχύει ότι η κατανομή του είναι κυρίαρχη. Τότε από τα θεωρήματα της προηγούμενος ενότητας για την ανάλυση του στατικού πρωτοκόλλου οι ιδιότητες common prefix, chain growth και chain quality ισχύουν. □

Τυχαιότητα

Η τυχαιότητα σε κάθε εποχή παράγεται από το πρωτόκολλο π_{RLB} συνδέοντας τις εξόδους των VRFs που μετέδωσαν οι χρήστες στην αλυσίδα blockchain, και το αποτέλεσμα υποβάλλεται σε μια συνάρτηση σύνοψης (hash function H) που υλοποιείται ως random oracle. Για λόγους ασφάλειας, οι πληροφορίες αυτές προέρχονται από blocks που δημοσιεύτηκαν δύο εποχές πριν.
Πιο συγκεκριμένα, για να δημιουργηθεί ο σπόρος η_j για την εποχή e_j :
Έστω C η αλυσίδα της εποχής e_{j-2}. Λαμβάνεται από κάθε $B = (st, d, sl, \sigma, B_\pi, \rho) \in C$ η τυχαιότητα των VRFs ρ, μέχρι και το block του slot που έχει timestamp $(j-2)R + \frac{16k}{1+\varepsilon}$. Όλες αυτές οι τυχαιότητες ενώνονται με τον σπόρο της τρέχουσας εποχής η_{j-1}, και υποβάλλονται σε μια hash function H :

$$\eta_j = H(\ \eta_{j-1} \ ||\ j ||\ \rho_i \ ||\ ...\ ||\ \rho_i'\ )$$

Παρατήρηση 7.2.1. Για την δημιουργία του τυχαίου σπόρου καλούνται ξεχωριστά οι VRFs, από την εκλογή αρχηγού. Πιο συγκεκριμένα, για να εκλεχθεί αρχηγός καλεί τις VRFs με τυχαιότητα $\eta_j \ ||\ sl \ ||\ TEST$ και λαμβάνεται το αποτέλεσμα (y, π), ενώ για να δημιουργηθεί ο σπόρος καλούνται οι VRFs με τυχαιότητα $\eta_j \ ||\ sl \ ||\ NONCE$ και λαμβάνεται το αποτέλεσμα (ρ_y, ρ_π) που χρησιμοποιείται όπως παραπάνω.

7.2.8 Περιγραφή του δυναμικού πρωτοκόλλου

Στο πρωτόκολλο για δυναμικό stake, η κατανομή χρημάτων δεν είναι σταθερή, αλλά αλλάζει σε κάθε εποχή. Η τυχαιότητα πλέον δεν λαμβάνεται από το genesis block όπως στο στατικό πρωτόκολλο, αλλά παράγεται μέσω του ίδιου του blockchain από τις VRFs σύμφωνα με το πρωτόκολλο π_{RLB}.

Αρχικοποίηση

Κατα την αρχικοποίηση του βασικού πρωτοκόλλου, ο κάθε χρήστης φροντίζει να αποκτήσει τις απαραίτητες πληροφορίες ώστε να προετοιμαστεί για το στάδιο επέκτασης της αλυσίδας. Πρώτα λαμβάνει από το περιβάλλον τα ζευγάρια δημοσίων και ιδιωτικών κλειδιών του για τις VRFs (pk^{vrf}, sk^{vrf}), για τις forward security υπογραφές (pk^{kes}, sk^{kes}), και για τις κανονικές υπογραφές (pk^{dsig}, sk^{dsig}). Στην συνέχεια ενημερώνεται για το τρέχων slot που κάνει την αρχικοποίηση. Εάν είναι το πρώτο slot sl_1, λαμβάνει το genesis block $genblock$, την κατανομή των stake $\mathbb{S}_0$, τον σπόρο τυχαιότητας η και κρατάει ένα τοπικό blockchain $C = B_0 = \mathbb{S}_0, \eta$. Αν το τρέχων slot δεν είναι το πρώτο, λαμβάνει την αλυσίδα από το σύστημα και φτιάχνει τοπικό blockchain C . Στην πρώτη περίπτωση ορίζει $st = H(B_0)$, αλλιώς $st = H(head(C))$. Εάν το πρωτόκολλο βρίσκεται στην δεύτερη ή σε μεταγενέστερη εποχή e_j, η τυχαιότητα λαμβάνεται μέσω του π_{RLB}. Αλλιώς, για την πρώτη εποχή βρίσκεται στο genesis block.

Με λίγα λόγια κατα την αρχικοποίηση προσδιορίζονται τα εξής :

1. Ζευγάρια κλειδιών VRF και των υπογραφών
2. Τρέχων slot, κατανομή χρημάτων και τυχαιότητα
3. Τοπικό Blockchain
4. Κατάσταση state

Επέκταση Αλυσίδας

Αφού ο χρήστης έχει δημιουργήσει μια τοπική αλυσίδα και έχει όλες τις απαραίτητες πληροφορίες, μπαίνει στην διαδικασία επέκτασης της αλυσίδας. Πρόκειται για μια επαναλαμβανόμενη διαδικασία που συμβαίνει R φορές, όσα και τα slot της εποχής, και πραγματοποιείται από κάθε χρήστη, είτε είναι slot leader είτε όχι.
Εάν πρόκειται για μια νέα εποχή e_j, $j \geq 2$, κάθε χρήστης διαβάζει το blockchain και λαμβάνει την κατανομή των χρημάτων $\mathbb{S}_j$ από το τελευταίο block $(j-2)R$. Στην συνέχεια λαμβάνεται η τυχαιότητα η_j που παράχθηκε στην προηγούμενη εποχή e_{j-1} από το π_{RLB}.

Ο χρήστης λαμβάνει τις συναλλαγές $d \in \{0,1\}^*$ που πρόκειται να βάλει σε κάποιο block του blockchain. Στην συνέχεια ενημερώνεται για όλες τις νέες αλυσίδες που υπάρχουν διαθέσιμες, και ελέγχει την εγκυρότητα του περιεχομένου τους. Δηλαδή επαληθεύονται οι forward security υπογραφές των block που περιέχει, ελέγχονται οι slot leader από τις VRFs με το pk^{vrf} και την απόδειξη π' και τυχαιότητα $\eta' \mid\mid sl' \mid\mid TEST$ και το φράγμα $y' < T'$, και τέλος ελέγχεται η τυχαιότητα η' από τις VRFs με το pk^{vrf} και την απόδειξη ρ'_π και τυχαιότητα $\eta' \mid\mid sl' \mid\mid NONCE$. Εάν όλα αυτά είναι ορθά, ορίζει ως νέα αλυσίδα αυτή που προκύπτει από την συνάρτηση maxvalid$(\mathbb{C}, C)$. Τότε $C' =$ maxvalid$(\mathbb{C}, C)$ και C' είναι η καινούρια τοπική αλυσίδα, και $st = H(head(C'))$. Καλούνται οι VRFs με το $\eta_j \mid\mid sl \mid\mid NONCE$ και επιστρέφουν τις τιμές ρ_y, ρ_π. Στην συνέχεια, στην περίπτωση που οι VRFs με το $\eta_j \mid\mid sl \mid\mid TEST$ επιστρέψουν μια τιμή y_i που είναι μικρότερη του φράγματος $T_i = 2^{y_i^{length}} \cdot \varphi_f(\alpha_i)$ του χρήστη U_i , φτιάχνει το $B = (st, d, sl, B_\pi, \rho, \sigma)$ όπου d οι συναλλαγές, $B_\pi = (U_i, y, \pi)$ οι πληροφορίες για την επαλήθευση του slot leader, $\rho = (\rho_y, \rho_\pi)$ οι πληροφορίες για την παραγωγή και επαλήθευση της μελλοντικής τυχαιότητας, και σ η υπογραφή $\sigma = Sign_{sk^{kes}}(st, d, sl, B_\pi, \rho)$. Δημιουργεί την καινούργια τοπική αλυσίδα $C'' = C'|B$ και την μεταδίδει στο σύστημα. Επίσης ορίζεται $st = H(head(C''))$.

Με λίγα λόγια κατα την επέκταση αλυσίδας έχουμε τα βήματα :

1. Δεδομένα συναλλαγών
2. Συλλογή έγκυρων αλυσιδών (σωστές forward security υπογραφές, σωστά στοιχεία από τις VRFs, $y' < T'$)
3. Εφαρμογή maxvalid()
4. Ενημέρωση τοπικής αλυσίδας και κατάστασης
5. Παραγωγή τυχαιότητας ρ με τις VRFs
6. Έλεγχος VRFs και φράγματος
7. Για θετική απάντηση :
   - Δημιουργία καινούριου block
   - Δημιουργία καινούριας αλυσίδας με το block
   - Μετάδοση καινούριας αλυσίδας
   - Ενημέρωση τοπικής αλυσίδας και κατάστασης

Παραγωγή συναλλαγών

Σε αυτό το στάδιο του πρωτοκόλλου ο χρήστης μπορεί να παράξει συναλλαγές, σύμφωνα πάντα με ένα υπόδειγμα έγκυρης συναλλαγής. Για αυτόν τον λόγο δημιουργεί και μια απλή υπογραφή με το pk^{dsig} μαζί με την συναλλαγή tx, και στην συνέχεια αποστέλνει το ζευγάρι (tx, σ) στο περιβάλλον.

Παρατήρηση 7.2.2 (**Αφάλεια δυναμικού πρωτοκόλλου**)**.** Έστω οι παράμετροι $k, R, \Delta, L \in \mathbb{N}, \varepsilon, \sigma \in (0,1)$. Αποδεικνύεται ότι η εκτέλεση του δυναμικού

πρωτοκόλλου του Ouroboros Praos με το π_{RLB} σε *Δ-μερικώς συγχρονισμένο δίκτυο*, με χρόνο L πολλαπλάσιο του R, $R = \frac{24k}{(1+\varepsilon)}$, και $\alpha_{\mathcal{H}}(1-f)^{\Delta} \geq \frac{(1+\varepsilon)}{2} + \sigma$, οι ιδιότητες persistence και liveness με παραμέτρους k και $u = \frac{8k}{(1+\varepsilon)}$ αντίστοιχα, θα ισχύουν με πιθανότητα $1-exp(lnL+\Delta-\Omega(k-logtkq)$ και άρα η εκτέλεση του πρωτοκόλλου θα είναι ασφαλής. Η παράμετρος t είναι ο αριθμός των διεφθαρμένων χρηστών και q είναι το φράγμα στις ερωτήσεις που μπορεί να κάνει στο random oracle.

Κεφάλαιο 8

Σύγκριση Πρωτοκόλλων

8.1 Εισαγωγή

Τα πρωτόκολλα Algorand και Ouroboros Classic, Ouroboros Praos που αναλύθηκαν παραπάνω, είναι όλα πρωτόκολλα κρυπτονομισμάτων που υλοποιούνται με την κατανεμημένη τεχνολογία blockchain, και εξελίσονται χρησιμοποιώντας κοινό αλγόριθμο ομοφωνίας Proof-of-Stake. Παρόλα αυτά μπορεί να έχουν άλλες διαφορές στο μοντέλο επικοινωνίας, στα κρυπτογραφικά εργαλεία που χρησιμοποιούν, στις υποθέσεις ασφάλειας, στην διαδικασία ομοφωνίας, στα κίνητρα που διαμοιράζονται ή και στις εφαρμογές τους. Σκοπός αυτού του κεφαλαίου είναι η σύγκριση των δύο πρωτοκόλλων ως προς τον τρόπο δομής και λειτουργίας τους.

8.2 Επίπεδο Δεδομένων

Τα πρωτόκολλα Ouroboros και Algorand διαθέτουν την ίδια δομή block. Πιο συγκεκριμένα, διαθέτουν τον αριθμό του γύρου του πρωτοκόλλου για τον οποίο παράγεται το block, μια τυχαία τιμή (σπόρος/seed) που χρησιμοποιείται για να την εκλογή των χρηστών που επιλέγονται να δράσουν στο πρωτόκολλο, την απόδειξη για την σωστή παραγωγή του seed, την τιμή κατακερματισμού (hash value) του προηγούμενου block για την ασφάλεια του blockchain, τα στοιχεία του χρήστη που το παργήγαγε, την απόδειξη ορθότητας επιλογής του, και τις συναλλαγές που θα προστεθούν με τα αντίστοιχα δεδομένα τους.

Η βασική διαφορά στην δομή των blockchain των πρωτοκόλλων Ouroboros Classic, Praos και Algorand είναι ότι στα πρωτόκολλα Praos και Algorand, μέσα στο κάθε block υπάρχουν οι αποδείξεις ορθού σπόρου τυχαιότητας και σωστά επιλεγμένου χρήστη που παρήγαγε το συγκεκριμένο block. Αυτό συμβαίνει διότι χρησιμοποιούν τις VRFs οι οποίες παρέχουν απόδειξη για κάθε ψευδοτυχαία τιμή που παράγουν. Αντιθέτως, το Ouroboros Classic λειτουργεί με ένα coin-tossing πρωτόκολλο για την παραγωγή της τυχαιότητας, και με follow-the-satoshi αλγόριθμο για εκλογή των χρηστών που παράγουν block. Αυτές οι πληροφορίες είναι αποθηκευμένες στο blockchain και οποιοσδήποτε θέλει να επαληθεύσει είτε τον σπόρο είτε την σωστή επιλογή του χρήστη θα πρέπει να ανατρέξει στην κατανομή χρημάτων από το blockchain και να πραγματοποιήσει μόνος του τις επαληθεύσεις.

Το προτέρημα λοιπόν της χρήσης των VRFs είναι ξεκάθαρο, καθώς παρέχεται άμεση αλλά και εύκολη επαλήθευση των ψευδοτυχαίων πληροφοριών. Το μόνο που έχουν να κάνουν οι χρήστες είναι να τρέξουν τον απλό αλγόριθμο επαλήθευσης. Αντιθέτως, η πρόταση του Ouroboros Classic περιλαμβάνει την συλλογή της κατανομής χρημάτων από την αλυσίδα, συλλογή όλων των μηνυμάτων του coin-tossing για την ψευδοτυχαιότητα και ξεχωριστή επαλήθευση κάθε μηνύματος.

8.3 Υποθέσεις ασφάλειας

8.3.1 Ποσοστό ειλικρινών χρηστών

Η βασική διαφορά στην υπόθεση ασφάλειας των πρωτοκόλλων Ouroboros και Algorand έχει να κάνει με τον αλγόριθμο ομοφωνίας που χρησιμοποιούν. Όπως προαναφέρθηκε το Algorand χρησιμοποιεί σαν αλγόριθμο ομοφωνίας έναν συνδυασμό Proof-of-Stake και Practical Byzantine Fault Tolerance. Επειδή τα καινούργια block προκύπτουν από βυζαντινού τύπου ομοφωνίες, για την ασφάλεια του πρωτοκόλλου γίνεται η υπόθεση ότι το κλάσμα των ειλικρινών χρηστών είναι ίσο ή μεγαλύτερο από $\frac{2}{3}$. Αντιθέτως, τα πρωτόκολλα Ouroboros υποθέτουν ειλικρινή πλειοψηφία χρηστών, δηλαδή περισσότερο από το 50% των χρηστών να είναι ειλικρινείς.

8.3.2 Κρυπτογραφικές υποθέσεις

Και στα τρία πρωτόκολλα γίνονται οι βασικές κρυπτογραφικές υποθέσεις ύπαρξης συναρτήσεων κατακερματισμού (hash functions) και ηλεκτρονικών υπογραφών. Ο αντίπαλος δεν έχει απεριόριστη υπολογιστική δύναμη και άρα όλα τα κρυπτογραφικά σχήματα που υποθέτουν την ύπαρξη hash functions όπως οι Verifiable Random Functions, είναι ασφαλή.

8.4 Επίπεδο Δικτύου

8.4.1 Συγχρονισμός

Το πρωτόκολλο Ouroboros Classic λειτουργεί σε ένα αυστηρά συγχρονισμένο δίκτυο. Όλοι οι ειλικρινείς χρήστες επικοινωνούν μεταξύ τους μέσα σε ένα καθορισμένο χρονικό διάστημα, μιας μονάδας χρόνου του πρωτοκόλλου. Δηλαδή μέσα σε μια μονάδα ενός slot όλοι οι ειλικρινείς χρήστες που έχουν στείλει κάποιο μήνυμα σε άλλο ειλικρινή χρήστη, αυτό έχει παραδοθεί. Όμως ο αντίπαλος μπορεί να διαβάσει τα μηνύματα και να αλλάξει την σειρά τους μέσα στο διάστημα αυτό.

Αντιθέτως, στο πρωτόκολλο του Algorand όπως και στο Ouroboros Praos έχουμε ένα μερικώς συγχρονισμένο δίκτυο. Στην περίπτωση του Algorand, ο χρόνος χωρίζεται σε τελείως ασύγχρονες περιόδους b που ακολουθούνται πάντα από πολύ πιο μικρές αλλά αυστηρού συγχρονισμού περιόδους s. Κατά την ασύγχρονη περίοδο b ο αντίπαλος μπορεί να πράξει όπως θέλει στο δίκτυο, αλλά το γεγονός ότι ακολουθεί μια περίοδος s εξασφαλίζει την ασφάλεια του πρωτοκόλλου καθώς τότε οι ειλικρινείς χρήστες έρχονται γρήγορα σε ομοφωνία. Το Ouroboros Praos υποθέτει ότι συνολοκά το δίκτυο είναι μερικώς συγχρονισμένο, υπό την έννοια ότι ο adversary έχει το δικαίωμα να καθυστερήσει οποιοδήποτε μήνυμα ειλικρινούς χρήστη το πολύ μέχρι χρόνο D, που μετράται στις μονάδες χρόνου του Ouroboros, δηλαδή μέχρι D slots.

8.4.2 Αρχιτεκτονική

Το δίκτυο του Algorand και του Ouroboros είναι αποκεντρωμένο και αποτελείται από κόμβους, οι οποίοι προκειμένου να επικοινωνήσουν μεταξύ τους, είναι οργανωμένοι σε ένα γράφημα όπου κάθε κόμβος συνδέεται τυχαία με άλλους γειτονικούς και ανταλλάσσουν μηνύματα. Η δομή αυτή αποτελεί ένα peer-to-peer δίκτυο. Όλοι οι κόμβοι έχουν γείτονες απλά

κάποιοι μπορεί να είναι πιο απομακρυσμένοι και άρα να λαμβάνουν με μια μικρή καθυστέρηση τα μηνύματα σε σχέση με τους άλλους.

Στο Algorand δεν υπάρχει κάποια διάκριση μεταξύ ενεργών και μη ενεργών κόμβων, το πρωτόκολλο που περιγράφεται κυρίως για ενεργούς χρήστες που μπορούν να λάβουν, να επιβεβαιώσουν και να διαμοιράσουν τα μηνύματα στο ομότιμο δίκτυο. Στην υλοποίηση του Algorand αναφέρεται πως είναι δυνατό να σημειωθεί ένας λογαριασμός ως ανενεργός (offline). Αντιθέτως, στο Ouroboros γίνεται διαχωρισμός των χρηστών σε ενεργούς και μη ενεργούς. Πιο συγκεκριμένα, στα πρωτόκολλα Ouroboros υποθέτουμε ότι οι χρήστες είναι ενεργοί την περισσότεροι ώρα, αλλιώς μπορούν να αναθέσουν τα καθήκοντά τους σε άλλους μέσω των Αντιπροσώπων. Στην περίπτωση που είναι ανενεργοί και τους έχει ζητηθεί να φτιάξουν κάποιο block, θα παραχθεί ένα κενό block.

Αναφέρεται στα [27],[28] πως το δίκτυο του Algorand αποτελείται από δύο είδη κόμβων, που ονομάζονται *relay nodes* και *participation nodes*. Οι κόμβοι τύπου *relay* συνδέονται με τους κόμβους τύπου *participation* ώστε να βοηθήσουν στην γρήγορη και αξιόπιστη μετάδοση των μηνυμάτων στο ομότιμο δίκτυο. Πιο συγκεκριμένα, ο ρόλος τους είναι να λαμβάνουν μηνύματα από *participation nodes* και άλλα *relay nodes*, και να ελέγχουν την εγκυρότητα των υπογραφών και τις προτεραιότητες και να διαμοιράζονται μόνο όσα είναι έγκυρα. Από την άλλη πλευρά, οι κόμβοι τύπου *participation* συμμετέχουν ενεργά στο πρωτόκολλο ομοφωνίας του Algorand και επικοινωνούν μεταξύ τους με την βοήθεια των *relay nodes*. Είναι οι πιο βασικοί κόμβοι και όσο συμμετέχουν ενεργά και ειλικρινώς στην ομοφωνία το Algorand είναι ασφαλές από διακλαδώσεις αλυσίδας (forks). Και οι δύο τύποι κόμβων μπορούν να συμμετάσχουν στο πρωτόκολλο ομοφωνίας, όμως προτιμάται να συμμετέχουν κυρίως οι κόμβοι *participation*. Οι κόμβοι *relay* αποθηκεύουν πάντα όλη την αλυσίδα, ενώ οι *participation nodes* αν θέλουν μπορούν να αποθηκεύσουν τοπικά περιορισμένο αριθμό κόμβων.

Στην υλοποίηση του Ouroboros για το Cardano, χρησιμοποιούνται τρία είδη κόμβων [19]: *core, relay και edge nodes*. Οι κόμβοι *core* είναι οι βασικότεροι κόμβοι καθώς συμμετέχουν στην ομοφωνία. Διαθέτουν χρήματα στον λογαριαμό τους και άρα μπορούν να παράξουν block. Επικοινωνούν με το υπόλοιπο δίκτυο μέσω των *relay nodes*, τα οποία δεν έχουν χρήματα ούτε κλειδιά, απλά επιτρέπουν την επικοινωνία των *core nodes* με το υπόλοιπο δίκτυο. Τέλος τα *edge nodes* είνα κόμβοι επίσης με μηδενικά χρήματα, που μπορούν όμως να εκδίδουν συναλλαγές και επικοινωνούν μόνο με τους κόμβους *relay*.

Παρατηρούμε λοιπόν πως τα δύο πρωτόκολλα λειτουργούν με βάση ένα peer-to-peer δίκτυο, και έχουν παρόμοια τοπολογία, με κόμβους που έχουν δικαίωμα να συμμετέχουν στην ομοφωνία και με κόμβους που βο-

ηθούν στην επικοινωνία των προαναφερθέντων με το υπόλοιπο δίκτυο. Ωστόσο, ίσως ο προσδιορισμός ενός peer-to-peer δικτύου που στη σύνδεση των κόμβων με τους γειτόνους θα λαμβανόταν υπόψιν το stake τους, να έκανε ακόμη πιο ασφαλή τα PoS πρωτόκολλα.

8.5 Επίπεδο Ομοφωνίας

8.5.1 Αλγόριθμος Ομοφωνίας

Τα πρωτόκολλα Ouroboros, όπως και το Algorand αποτελούν Proof-of-Stake πρωτόκολλα. Οι χρήστες που έχουν ενεργό ρόλο στο πρωτόκολλο (επικύρωση συναλλαγών, δημιουργία block, επιτροπές λήψης αποφάσεων) επιλέγονται με βάση το ποσό χρημάτων που διαθέτουν στον λογαριασμό τους. Υπάρχει όμως μια βασική διαφορά στον αλγόριθμο ομοφωνίας των πρωτοκόλλων Ouroboros και του Algorand, το Algorand χρησιμοποιεί ένα υβριδικό πρωτόκολλο ομοφωνίας Proof-of-Stake και Practical Byzantine Fault Tolerance, ενώ τα Ouroboros χρησιμοποιούν καθαρά Proof-of-Stake.

Πιο συγκεκριμένα, στο Algorand για να έρθουν οι χρήστες σε ομοφωνία, σχηματίζουν επιτροπές οι οποίες ψηφίζουν συνεχώς για μια τιμή μέχρι να μαζευτούν αρκετές ψήφοι για αυτή. Η λογική είναι ότι στο σύστημα υπάρχουν χρήστες ειλικρινείς αλλά και χρήστες μη-ειλικρινείς, και το πρόβλημα αυτό αντιμετωπίζεται με μια υπόθεση ασφάλειας $\frac{2}{3}$ των χρηστών να είναι ειλικρινείς. Όσο μαζεύονται ψήφοι αριθμού ίδιου ή μεγαλύτερου από τα $\frac{2}{3}$ των χρηστών, το πρωτόκολλο θα έρχεται σε ομοφωνία. Αυτή η λογική που περιγράφτηκε αποτελεί μια Byzantine Fault Tolerance ομοφωνία. Η ομοφωνία Proof-of-Stake αξιοποιείται έτσι ώστε οι χρήστες των επιτροπών που σχηματίζονται στο Algorand, να επιλέγονται με βάση τα χρήματα που διαθέτουν στο σύστημα. Εν τέλει το καινούργιο block στην αλυσίδα του Algorand προκύπτει απο την Βυζαντινή Συμφωνία επιτροπής χρηστών διαλεγμένων ανάλογα με τα χρήματα που διαθέτουν στο σύστημα.

Τα πρωτόκολλα Ouroboros από την άλλη, έχουν φτιαχτεί ώστε οι χρήστες που θα προσθέσουν το καινούριο block στην αλυσίδα να επιλέγονται κατευθείαν με βάση τα χρήματά τους. Αυτό σημαίνει πως δεν υλοποιείται κάποια ψηφοφορία για την εκλογή των χρηστών ή την προσθήκη ενός καινούργιου block. Εάν κάποιος χρήστης έχει επιλεχθεί για αυτή την ενέργεια, στέλνει απλά το μήνυμα προσθήκης καινούργιου block. Για αυτόν το λόγο τα πρωτόκολλα Ouroboros έχουν μια άλλου είδους ανάλυση ασφάλειας από αυτό του Algorand, που επιτυγχάνει την ασ-

φάλεια μέσα από τις συνεχείς ψηφοφορίες βυζαντινού τύπου.

8.5.2 Παραγωγή block

Για τα πρωτόκολλα Ouroboros έχουμε άμεση παραγωγή block από την στιγμή που έχει ανατεθεί σε κάποιο χρήστη αυτή η δουλειά. Δηλαδή σχηματίζεται μια επιτροπή με το σύνολο των χρηστών που θα παράξουν τα block σε μια εποχή, και κάθε χρήστης είναι υπεύθυνος για ένα συγκεκριμένο block. Είτε ο χρήστης αυτός είναι ειλικρινής ή κακόβουλος, εάν τύχει να επιλεχθεί αυτός θα το παράξει. Στην συνέχεια μέσω του κανόνα *maxvalid* επιλέγονται τα block που επεκλείνουν την μεγαλύτερη αλυσίδα. Αντιθέτως, το Algorand λειτουργεί με τελείως διαφορετικό τρόπο. Για να προστεθεί το κάθε block στην αλυσίδα θα πρέπει να περάσει από αρκετές ψηφοφορίες, να ελεγχθεί και στην συνέχεια να προστεθεί. Για να συμβεί αυτό δημιουργούνται διάφορες επιτροπές, η πρώτη είναι για να προταθούν κάποια block, η δεύτερη για να αναχθεί η επιλογή σε ένα block, και οι επόμενες για να δημιουργηθεί ασφαλής ομοφωνία σε αυτό. Άρα, εάν κάποιος ανήκει στην επιτροπή πρότασης block, δεν σημαίνει πως θα προσθέσει αναγκαστικά το block του στην αλυσίδα, αυτό θα γίνει μόνο εάν έχει την μεγαλύτερη προτεραιότητα ανάμεσα στις προτάσεις και το σύστημα μπορεί να έρθει σε ασφαλή ομοφωνία. Στο Ouroboros λοιπόν έχουμε άμεση παραγωγή block, αντιθέτως στο Algorand θα πρέπει ένα μεγάλο ποσοστό των χρηστών να το ελέγξει και να συμφωνήσει σε αυτό.

Συνεπώς στο Ouroboros επιτρέπονται οι διακλαδώσεις (forks) αφού η παραγωγή των block γίνεται άμεσα και στην συνέχεια διαλέγεται η μεγαλύτερη αλυσίδα με τον κανόνα *maxvalid*. Στο Algorand επειδή γίνονται πολλές ψηφοφορίες ώστε όλο το σύστημα να έρθει σε ομοφωνία για μια και μόνο τιμή block, δεν μπορεί να υπάρξει fork, εκτός και αν το σύστημα είναι ασύγχρονο για πάρα πολλή ώρα. Ακόμα και τότε όμως υπάρχει το recovery protocol στο οποίο οι χρήστες συνεχώς προσπαθούν να έρθουν σε ομοφωνία για ένα fork. Η πιθανότητα να συμβεί όμως αυτό είναι πολύ μικρή.

Εν τέλει, από τα παραπάνω συμπεραίνουμε πως το Ouroboros λύνει το πρόβλημα εμφάνισης διακλαδώσεων στην αλυσίδα (forks) με τον κανόνα επιλογής μεγαλύτερης αλυσίδας, αντιθέτως, στο Algorand οι διακλαδώσεις λύνονται με το πρωτόκολλο βυζαντινής συμφωνίας (BA★).

8.5.3 Σχηματισμός Επιτροπής

Όλα τα πρωτόκολλα έχουν διαφορετικό τρόπο σχηματισμού επιτροπής.
Όσον αφορά το πρωτόκολλο του Ouroboros Classic, ο σχηματισμός επιτροπής

γίνεται βάσει του Follow The Satoshi αλγόριθμου, και για την εφαρμογή του θα πρέπει όλα τα χρήματα στους λογαριασμούς των χρηστών να βρίσκονται στην μικρότερη ακέραια μονάδα νομίσματος. Υπο αυτή την υπόθεση, κάθε νόμισμα αποκτά έναν δείκτη από το 1 έως τον αριθμό των συνολικών νομισμάτων. Σε όλα τα νομίσματα αντιστοιχίζεται ένας ξεχωριστός αριθμός. Με βάση μια τυχαιότητα που αρχικοποιεί τον αλγόριθμο, επιλέγεται τυχαία ένα εκ των συνολικών νομισμάτων και ο ιδιοκτήτης του είναι ο χρήστης που θα παράξει το καινούργιο block. Έτσι σχηματίζεται η επιτροπή που θα παράξει τα blocks σε μια καθορισμένη χρονική περίοδο, που το πρωτόκολλο ονομάζει εποχή. Για κάθε block είναι υπεύθυνος ένας και μόνο χρήστης. Τα αρνητικό αυτού του σχήματος είναι ότι οι χρήστες επιλέγονται δημόσια και έτσι μπορούν να στοχοποιηθούν πολύ εύκολα πριν δράσουν στο πρωτόκολλο.

Στο πρωτόκολλο Ouroboros Praos, η επιτροπή σχηματίζεται με την χρήση των Verifiable Random Functions και πλέον γίνεται με ιδιωτικό τρόπο σε σχέση με την διαδικασία που ακολουθεί το Ouroboros Classic. Πιο συγκεκριμένα, κάθε χρήστης ιδιωτικά υπολογίζει την ψευδοτυχαία του τιμή των VRFs, και εάν αυτή είναι μικρότερη από ένα φράγμα T τότε επιλέγεται ως μέλος της επιτροπής παραγωγής block σε μια εποχή. Το φράγμα T είναι ξεχωριστό για κάθε χρήστη και φτιάχνεται από μια συνάρτηση κατάλληλα επιλεγμένη, που εξάγει έναν αριθμό από το 0 έως το 1 και ο αριθμός αυτός είναι η πιθανότητα που έχει ο χρήστης να επιλεχθεί με βάση τα νομίσματα που έχει στο σύστημα. Αυτό σημαίνει ότι για ένα block μπορεί να ανατεθούν ένας, περισσότεροι ή και κανένας χρήστης. Ο σχηματισμός επιτροπής για το Ouroboros Praos είναι πολύ καλύτερος από αυτόν του Ouroboros Classic, καθώς το σύνολο των χρηστών του Ouroboros μαθαίνει ότι ένας χρήστης έχει επιλεχθεί να παράξει block μόνο όταν αυτός δράσει, και επιπλέον οι υπόλοιποι χρήστες μπορούν πολύ ευκολα να επιβεβαιώσουν την επιλογή αυτή με ελάχιστες πράξεις επαλήθευσης των VRFs, της τυχαιότητας της εποχής και των υπογραφών.

Στο πρωτόκολλο Algorand, όλες οι επιτροπές, είτε είναι για πρόταση block είτε για ψηφοφορία, σχηματίζονται με τον αλγόριθμο Cryptographic Sortition. Η πιθανότητα να επιλεχθεί ένας χρήστης στην επιτροπή είναι ανάλογη με τα νομίσματα που έχει στον λογαριασμό του, και ο αλγόριθμος δημιουργείται με δύο βασικά στοιχεία, την διωνυμική κατανομή και τις Verifiable Random Functions. Η διωνυμική κατανομή χρησιμοποιείται ώστε να ελεγχθεί ο αριθμός των χρηστών που θα επιλεχθούν στην επιτροπή. Αυτό είναι αναγκαίο διότι δημιουργούνται δύο ειδών επιτροπές και κάθε μια έχει ξεχωριστό ρόλο και άρα απαιτείται διαφορετική πληθικότητα σε κάθε μια. Για παράδειγμα μια επιτροπή πρότασης block χρειάζεται 25 άτομα ώστε το σύστημα να είναι αποδοτικό, ενώ η επιτροπή ψηφοφορίας χρειάζεται 1000 άτομα ώστε το σύστημα να έρθει σε ασφαλή ομοφωνία. Το διάστημα [0,1) χωρίζεται σε υποδιαστήματα μήκους όσο η πιθανότητα να επιλεχθούν κανένα, ένα ή περισσότερα νομίσματα ενός χρήστη. Στην συνέχεια ελέγχεται σε ποιό διάστημα πέφτει η ψευδοτυχαία τιμή των Verifiable Random Functions. Το διάστημα αυτό υποδεικνύει τον αριθμό

των προτάσεων που έχει δικαίωμα να κάνει ο χρήστης, ή τον αριθμό των ψήφων που έχει σε κάποιο βήμα.

Εν τέλει, ο σχηματισμός επιτροπής στο Ouroboros Classic διαφέρει αρκετά σε σχέση με αυτόν στα Ouroboros Praos και Algorand. Ο πρώτος πραγματοποιείται στην αρχή κάθε εποχής δημόσια και οι χρήστες που θα παράξουν block είναι γνωστοί εκ των προτέρων. Αντιθέτως τόσο στο Ouroboros Praos όσο και στο Algorand, η διαδικασία είναι ιδιωτική για κάθε χρήστη και τα μέλη της επιτροπής γίνονται γνωστά μόνο όταν αυτά πράξουν. Αυτή είναι μια σημαντική ιδιότητα καθώς προστατεύει τα πρωτόκολλα από επιθέσεις τύπου denial-of-service attacks. Τέλος τα πρωτόκολλα Ouroboros Praos και Algorand χρησιμοποιούν το ίδιο κρυπτογραφικό εργαλείο: Verifiable Random Functions.

8.5.4 Σπόρος Τυχαιότητας

Στο Ouroboros Classic, προκειμένου να παραχθεί ο σπόρος τυχαιότητας που χρησιμοποιείται για ολόκληρη την εποχή, εκτελείται ένα πρωτόκολλο G.O.D. Coin Tossing. Πιο συγκεκριμένα, δεν συμμετέχουν όλοι οι χρήστες σε αυτό το πρωτόκολλο αλλά μόνο όσοι έχουν επιλεχθεί στην επιτροπή για να παράξουν κάποιο block. Όλοι οι χρήστες της επιτροπής, διαλέγουν μια τυχαία τιμή και με το σχήμα PVSS την διαμοιράζονται στο blockchain. Οι παραλήπτες κάνουν commit σε αυτή την τιμή που έλαβαν μέσω του PVSS και αφού δουν αρκετά έγκυρα commitments, κάνουν reveal της τιμής, κάθε χρήστης συγκεντρώνει τις τυχαίες τιμές που προέκυψαν και η τελική τυχαιότητα προκύπτει από το ευθύ άθροισμα όλων των τυχαίων τιμών. Το μειονέκτημα αυτού του πρωτοκόλλου G.O.D. Coin Tossing είναι η επαλήθευση καθώς όλοι οι υπόλοιποι χρήστες θα πρέπει να ανατρέξουν στο blockchain και να συλλέξουν τα μηνύματα της διαδικασίας και να την επαναλάβουν ελέγχοντας τις υπογραφές και τις αποδείξεις.

Για τον λόγο αυτό, στο βελτιωμένο πρωτόκολλο Ouroboros Praos, για την παραγωγή τυχαιότητας χρησιμοποιούνται οι Verifiable Random Functions με τις οποίες παράγεται ένα ψευδοτυχαίο αποτέλεσμα που είναι πολύ εύκολο να επαληθευτεί από οποιονδήποτε. Η διαδικασία που ακολουθείται σε κάθε εποχή είναι: όλα τα μέλη της επιτροπής συλλέγουν τον σπόρο τυχαιότητας, και μαζί με τον αριθμό του slot (μικρότερη μονάδα χρόνου στο Ouroboros) και το string "NONCE", φτιάχνουν καινούργιο σπόρο από τις VRFs και τον κοινοποιούν με την απόδειξη του. Ο τελικός σπόρος προκύπτει από την συνένωση του σπόρου του προηγούμενου γύρου μαζί με τις τυχαιότητες που έχουν φτιαχτεί από τους αρχηγούς των slot.

Στο Algorand επίσης χρησιμοποιούνται οι Verifiable Random Functions

για την παραγωγή του σπόρου. Παράγεται από τον προηγούμενο γύρο και προστίθεται μέσα στο block, και για την παραγωγή το χρησιμοποιείται ο σπόρος του προηγούμενου γύρου μαζί με τον αριθμό του γύρου. Αυτά μαζί με το ιδιωτικό κλειδί του χρήστη, είναι τα δεδομένα εισόδου των VRFs, και εξάγεται μια ψευδοτυχαία τιμή που είναι ο καινούριος σπόρος μαζί με την απόδειξη του.

Παρατηρείται πως το Ouroboros Praos όπως και το Algorand έχουν κοινό τρόπου παραγωγής σπόρου τυχαιότητας χρησιμοποιώντας το εργαλείο Verifiable Random Functions, παρ' όλα αυτά λόγω της διαφορετικής δομής τους, στο Ouroboros Praos ο σπόρος της καινούριας εποχής δημιουργείται με την εφαρμογή των VRFs *στην συνένωση των σπόρων της προηγούμενης*, ενώ στο Algorand ο σπόρος της καινούριας εποχής δημιουργείται με την εφαρμογή των VRFs *μόνο στον σπόρο της προηγούμενης*.

8.6 Επίπεδο Αμοιβών

Στο επίπεδο των αμοιβών, το Algorand δεν αναφέρει κάποιο μηχανισμό απόδοσης αμοιβών για τους χρήστες που συμμετέχουν στην ομοφωνία. Από την άλλη πλευρά, το Ouroboros διαθέτει έναν αποδεδειγμένα ασφαλή με θεωρία παιγνίων μηχανισμό. Η περιγραφή του μηχανισμού γίνεται στο Ouroboros Classic αλλά αποδεικνύεται πως μπορεί να χρησιμοποιηθεί και στο Ouroboros Praos [2]. Ο μηχανισμός αυτός αποδίδει αμοιβές σε όσους παράγουν block και σε όσους επικυρώνουν συναλλαγές και τις προσθέτουν στα blocks. Δηλαδή στους slot leaders και στους input endorsers. Οι αμοιβές αυτές συλλέγονται από φόρους που υπάρχουν σε όσους πραγματοποιούν συναλλαγές στο σύστημα. Οι φόροι αυτοί συλλέγονται στο τέλος κάθε εποχής και διαμοιράζονται στους χρήστες που συμμετείχαν στην συγκεκριμένη εποχή με τους ρόλους που αναφέρθηκαν. Αποδεικνύεται πως αυτός ο μηχανισμός είναι προσεγγιστικά μια δ-Ισορροπία Νας και άρα με καμία στρατηγική δεν μπορεί να επωφεληθεί κάποιος χρήστης περισσότερο από την απόδοση που του αντιστοιχεί μέσω του μηχανισμού.

8.7 Αναπαράσταση σύγκρισης

	Ouroboros Classic [1]	Ouroboros Praos [2]	Algorand [3]
Μηχανισμός Ομοφωνίας	PoS	PoS	PoS και PBFT
Ανοχή Σφαλμάτων	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{3}$
Συγχρονισμός Δικτύου	Συγχρονισμένο	Μερικώς Συγχρονισμένο	Μερικώς Συγχρονισμένο
Κρυπτογραφικά εργαλεία	PVSS	VRFs	VRFs
Εκλογή Αρχηγών	Follow The Satoshi	VRFs	VRFs
Επίλυση Fork	maxvalid(C)	maxvalid(C)	BA★
Απόδοση Αμοιβών	Ναι	Ναι	Οχι

Κεφάλαιο 9

Συμπεράσματα

Στόχος αυτής της διπλωματικής ήταν να γίνει μια ανάλυση στα πρωτόκολλα αλυσιδών συναλλαγών του Algorand και του Ouroboros, με σκοπό να γίνουν πιο κατανοητοί και εύπεμπτοι οι μηχανισμοί και ο τρόπος λειτουργίας τους. Για τον λόγο αυτό, η δομή της εργασίας είναι τέτοια που προωθεί την μελέτη τους σε επίπεδα. Είναι αξιοσημείωτο πως τα δύο αυτά πρωτόκολλα, αν και έχουν την ίδια εφαρμογή (υλοποιούν κρυπτονομίσματα), αλλά και παρόμοιους μηχανισμούς (κοινός μηχανισμός ομοφωνίας Proof-of-Stake, χρήση παρόμοιων κρυπτογραφικών εργαλείων), ταυτόχρονα έχουν τις δικές τους ξεχωριστές καινοτομίες. Στο Algorand παρατηρείται μια ιδιαίτερα προσεγμένη δομή, μια καινοτόμα εφαρμογή των VRFs για την δημιουργία επιτροπών, και με λεπτομερείς και πολυβηματικές διαδικασίες ψηφοφοριών, που όμως διατητούν την αποδοτικότητα και ασφάλεια του συστήματος. Το Ouroboros είναι επίσης μια ιδιαίτερη περίπτωση καθώς αποτελεί ένα proof-of-stake πρωτόκολλο με ισχυρή ασφάλεια και μηχανισμούς πανομοιότυπους με το Bitcoin, που όμως υλοποιεί τελείως διαφορετικό αλγόριθμο ομοφωνίας. Επιπλέον, παρέχει πολύ χρήσιμα στοιχεία, όπως αποδοτικό μηχανισμό αμοιβών και δυνατότητα μετάβασης αρμοδιοτήτων σε κατάλληλους αντιπροσώπους. Φυσικά υπάρχουν και άλλα πρωτόκολλα με εξίσου καινοτόμες ιδέες, που θα μπορούσαν να μελετηθούν σε κάποια μελλοντική έρευνα ώστε να υπάρχει μια καλύτερη εικόνα για πιθανές βελτιώσεις σε blockchain που υλοποιούν αλγόριθμο ομοφωνίας proof-of-stake.

Βιβλιογραφία

[1] A. Kiayias, I. Konstantinou, A. Russell, B. David, and R. Oliynykov. Ouroboros: A provably secure proof-of- stake blockchain protocol. (2016)

[2] B. David, P. Gazi, A. Kiayias and A. Russell. Ouroboros Praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. (2017)

[3] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, N. Zeldovich. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. (2017)

[4] J. Chen, S. Micali. ALGORAND*. (2017)

[5] J. Chen, S. Micali. Algorand: A secure and efficient distributed ledger. (2019)

[6] S. Micali, M. Rabiny, S. Vadhan: Verifiable Random Functions. (1999)

[7] L. Lamport, R. Shostak, and M. Pease: The Byzantine Generals Problem. (1982)

[8] R. Canetti: Universally Composable Security. (2000)

[9] Ε. Ζάχος, Α. Παγουρτζής, Π. Γροντάς: Υπολογιστική Κρυπτογραφία. (2015)

[10] A. Shamir: On the Generation of Cryptographically Strong Pseudo-random Sequences. (1981)

[11] O Goldreich, S Goldwasser, S Micali: How to construct random functions. (1986)

[12] RL. Rivest: Electronic Lottery Tickets as Micropayments (1997)

[13] I. Cascudo and B. David: SCRAPE: Scalable Randomness Attested by Public Entities (2017)

[14] A. Boldyreva, A. Palacio, B. Warinschi: Secure Proxy Signature Schemes for Delegation of Signing Rights (2003)

[15] B. Schoenmakers: A Simple Publicly Verifiable Secret Sharing Scheme and its Application to Electronic Voting (1999)

[16] A. Shamir: How to Share a Secret (1979)

[17] M. Bellare, SK. Miner: A Forward-Secure Digital Signature Scheme (1999)

[18] V. Strassen: Existence Of Probability Measures With Given Marginals (1965)

[19] K. Aydinli: Performance Assessment of Cardano. (2019)

[20] M. Blum: Coin Flipping by Telephone a protocol for solving impossible problems. (1983)

[21] S. Micali, RL. Rivest: Micropayments Revisited (2002)

[22] O. Goldreich, LA. Levin: A Hard-Core Predicate for all One-Way Functions. (1989)

[23] C. Cachin, S. Micali, M. Stadler: Computationally private information retrieval with polylogarithmic communication. (1999)

[24] MO. Rabin: Probabilistic algorithms for testing primality. (1980)

[25] R. Solovay and V. Strassen: A fast Monte-Carlo test for primality. (1977)

[26] I. Haitner, N. Makriyannis and E. Omri: On the Complexity of Fair Coin Flipping. (2018)

[27] Algorand Foundation: Pure-Proof-of-Stake Protocol. https://algorand.foundation/

[28] Algorand Developer Portal. https://developer.algorand.org/

[29] J. Katz and Y. Lindell: Introduction To Modern Cryptography, Second Edition. (2015)

[30] O. Goldreich: Foundations of Cryptography - Teaching Notes. (2001)

[31] S. Nakamoto: A Peer-to-Peer Electronic Cash System (2008)

[32] DL. Chaum: Computer Systems Established, Maintained, and Trusted by Mutually Suspicious Groups. (1982)

[33] S. Haber, WS. Stornetta: How to Time-Stamp a Digital Document. (1991)

[34] Z. Zheng, S. Xie, HN. Dai, X. Chen, H. Wang: Blockchain challenges and opportunities: a survey. (2018)

[35] A. Baliga: Understanding Blockchain Consensus Models. (2017)

[36] V. Buterin: Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform. (2014)

[37] P. Daian, R. Pass, and E. Shi.: Snow white: Provably secure proofs of stake. (2016)

[38] Ethereum.org. https://ethereum.org/

[39] M. Castro, B. Liskov: Practical Byzantine Fault Tolerance. (1999)

[40] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, SW. Cocco, J. Yellick :Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains (2018)

[41] AD. Kshemkalyani, M. Singhal: Distributed Computing: Principles Algorithms and Systems.

[42] M. Singhal, N. Shivaratri: Advanced Concepts in Operating Systems. (1994)

[43] R. Schollmeier: A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. (2001)

[44] RL. Rivest, A. Shamir, και L. Adleman: A method for obtaining digital signatures and public-key cryptosystems. (1978)

[45] W.Diffie , M.Hellman: New directions in cryptography. (1976)

[46] IB. Damgård: A design principle for hash functions. (1989)

[47] LA. Levin: One way functions and pseudorandom generators. (1987)

[48] M. Naor, O. Reingold: From unpredictability to indistinguishability: A simple construction of pseudorandom functions from macs (extended abstract). (1998)

[49] D.Boneh, MK.Franklin: Identity-based encryption (2001).

[50] N. Döttling, S. Garg: Identity-based encryption from the Diffie-Hellman assumption (2017).