Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Μηδενικής Γνώσης Επαληθευσιμότητα Κατανεμημένης Αποθήκευσης

Διπλωματικη Εργασια

του

ΝΙΚΟΛΑΟΥ Σ. ΣΤΑΜΑΤΕΛΟΠΟΥΛΟΥ

Επιβλέπων: Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Αθήνα, -Προσθήκη μήνα/έτους-

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Μηδενικής Γνώσης Επαληθευσιμότητα Κατανεμημένης Αποθήκευσης

Διπλωματικη Εργασια

του

ΝΙΚΟΛΑΟΥ Σ. ΣΤΑΜΑΤΕΛΟΠΟΥΛΟΥ

Επιβλέπων: Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την -Προσθήκη μήνα/έτους-.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Νεκτάριος Κοζύρης	-1ος Εξεταστής-	-1ος Εξεταστής-
Καθηγητής Ε.Μ.Π.	-προσθηκη τιτλου-	-προσθηκη τιτλου-

Αθήνα, -Προσθήκη μήνα/έτους-

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

(Υπογραφή)

.....
Νικόλαος
Σταματελόπουλος

-Προσθήκη μήνα/έτους-

Περίληψη

Στην εποχή της πληροφορίας, τα κατανεμημένα συστήματα αρχείων αποτελούν μια αποδοτική λύση στις διαρκώς αυξανόμενες ανάγκες αποθήκευσης μεγάλων όγκων δεδομένων. Η απομάκρυνση από το συγκεντρωτικό μοντέλο του απλού διακομιστή και ο διαμοιρασμός των δεδομένων σε μια οργανωμένη συστάδα πολλαπλών κόμβων ευνοεί την κλιμακωσιμότητα του συστήματος, όμως στερείται κεντρικής αρχής ελέγχου. Δεδομένου ότι οι κόμβοι αυτοί μπορεί να αποτελούν ξεχωριστές φυσικές οντότητες στις οποίες πιθανώς να έχουν φυσική ή απομακρυσμένη πρόσβαση τρίτα πρόσωπα, εγείρονται ερωτήματα σχετικά με την ακεραιότητα των δεδομένων που φέρουν.

Στόχος της παρούσας διπλωματικής είναι ο σχεδιασμός και η υλοποίηση ενός κατανεμημένου συστήματος αρχείων που θα προσφέρει μια λύση στο πρόβλημα της ακεραιότητας των δεδομένων του χρήστη. Για την επίτευξη αυτού, χρησιμοποιούμε την τεχνολογία του Ethereum, με σχεδιασμό κατάλληλων έξυπνων συμβολαίων, εισάγοντας στο hdfs μια τρίτη έμπιστη οντότητα, υπεύθυνη για τον παραπάνω έλεγχο. Επιπλέον εκμεταλλευόμαστε τη μέθοδο των zk-SNARKs, με τους επιμέρους κόμβους της συστάδας να κατασκευάζουν κατάλληλες αποδείξεις μηδενικής γνώσης, ικανές να μας πείσουν για την ακεραιότητα των δεδομένων που φέρουν, οι οποίες και θα ελέγχονται μέσω των έξυπνων συμβολαίων (on-chain). Κατά τον τρόπο αυτό καθιερώνουμε ένα σύστημα ελέγχου που βασίζεται στην δημόσια επαληθευσιμότητα του Ethereum χωρίς να θυσιάζει την εμπιστευτικότητα, αφού οι αποδείξεις που στέλνουν οι κόμβοι δεν προδίδουν καμία πληροφορία σχετικά με το περιεχόμενο των δεδομένων που αντιπροσωπεύουν. Έτσι κάθε χρήστης του συστήματος έχει τη δυνατότητα να επαληθεύσει πως τα αρχεία του δεν έχουν τροποποιηθεί ή διαγραφθεί από τρίτα πρόσωπα με πιθανές μεθόδους που δεν υπακούουν στο πρωτόκολλο του hdfs. Κατόπιν πειραματικής αξιολόγησης συμπεραίνουμε πως η ασφάλεια των δεδομένων σε ένα κατανεμημένο σύστημα αρχείων είναι επιτεύξιμη σύμφωνα με την παραπάνω προσέγγιση, με σχετικά μικρό κόστος ως προς τη διαθεσιμότητα του συστήματος, όμως περιορίζεται από τις υψηλές υπολογιστικές απαιτήσεις των zk-snarks.

Λέξεις Κλειδιά

Ακεραιότητα Δεδομένων, Κατανεμημένο Σύστημα Αρχείων, HDFS, Merkle Trees, zk-SNARK

Abstract

In the era of information, distributed file systems implement an efficient solution to the rapidly increasing demand in storage for large volumes of data. While withdrawing from the centralized model of the simple server and dividing the data in an organized cluster of multiple nodes favors scalability, it lacks on central audit authority. Considering that each system node can run as an individual physical entity, with the possibility of third parties having access to it either physically or remotely, arises the question of whether our data is actually secure.

The purpose of this thesis is to design and implement a distributed file system that provides a solution to the problem of the integrity of the stored user data. In order to accomplish that, we utilize the technology of Ethereum, by creating appropriate smart contracts, to introduce a third trusted party auditor to the existing HDFS impementation, that is responsible for verifying the integrity of the data. Additionally, we take advantage of the idea behind zk-SNARKs, with each individual node of the cluster providing appropriate zero knowledge proofs, capable of convincing about the integrity of the data in possession, that can then be verified through the smart contract (on-chain). Thus, we establish a auditing system based on the public verifiability of Ethereum without sacrificing confidentiality, since the proofs that each node sends, provide no information of the actual content of the data they refer to. Therefore each user of this system has the ability to verify that his files were not modified or deleted from a third party through a way that could possibly not comply to the hdfs protocol. Following experimental evaluation we conclude that providing data security in a distributed file system is achievable through the above approach, with little degradation in system availability, but is limited due to the high computational requirements of the zk-SNARKs.

Keywords

Data Integrity, Distributed File System, HDFS, Merkle Trees, zk-SNARK

Ευχαριστίες

Θα ήθελα καταρχάς να ευχαριστήσω τον καθηγητή Νεκτάριο Κοζύρη για την ευκαιρία που μου έδωσε να εκπονήσω την παρούσα διπλωματική στο εργαστήριο Υπολογιστικών Συστημάτων. Επίσης ευχαριστώ θερμά την κ. Αικατερίνη Δόκα για την εξαιρετική συνεργασία, την υπομονή και την πολύτιμη καθοδήγησή της. Τέλος θα ήθελα να ευχαριστήσω την οικογένειά μου για την ηθική συμπαράσταση που μου προσέφερε όλα αυτά τα χρόνια.

Αθήνα, -Προσθήκη μήνα/έτους-

Νικόλαος Σταματελόπουλος

Περιεχόμενα

Κατάλογος Σχημάτων

Κεφάλαιο 1

Εισαγωγή

Η ανάγκη για μεταφορά και κοινή πρόσβαση σε αρχεία μεταξύ απομακρυσμένων υπολογιστικών συστημάτων έγινε αισθητή πριν τη διάδοση και καθιέρωση των τοπικών δικτύων. Στις αρχές της δεκαετίας του 1980, όσοι χρήστες επιθυμούσαν να μοιραστούν ή να μεταφέρουν αρχεία, τα αντέγραφαν σε κάποιο φορητό αποθηκευτικό μέσο, όπως μαγνητικές ταινίες ή δισκέτες, το οποίο και μετέφεραν από τον έναν υπολογιστή στον άλλον. Η μέθοδος αυτή έλαβε ανεπίσημα τη χιουμοριστική ονομασία sneakernet και χρησιμοποιείται ακόμα και στη σημερινή εποχή με πιο σύγχρονα αποθηκευτικά μέσα. Είναι όμως ολοφάνερο πως μια τέτοια μέθοδος αδυνατεί να αντεπεξέλθει σε περιβάλλοντα με πολλούς χρήστες.

Με την εξέλιξη της τεχνολογίας των δικτύων υπολογιστών, καθιερώθηκαν νέες τεχνικές διαμοιρασμού αρχείων που εκμεταλλεύονταν τη μετάδοση δεδομένων σε ένα τοπικό δίκτυο (LAN), και αργότερα σε δίκτυα μεγαλύτερης κλίμακας, όπως το Πρωτόκολλο Μεταφοράς Αρχείων FTP. Αν και η μέθοδος απέφευγε τη χρονοβόρα φυσική μεταφορά αποθηκευτικών μέσων, τα αρχεία έπρεπε να αντιγραφούν αρχικά από τον υπολογιστή σε έναν διακομιστή, και από εκεί ξανά προς τον υπολογιστή προορισμό, που επιθυμούσε να τα ανακτήσει. Επιπλέον, οι χρήστες ήταν υποχρεωμένοι να γνωρίζουν τη φυσική διεύθυνση κάθε υπολογιστή που συμμετείχε στη διαδικασία διαμοιρασμού των αρχείων [1].

Καθώς κατεβλήθηκαν προσπάθειες για την αντιμετώπιση των παραπάνω προβλημάτων, αναπτύχθηκαν καινούργια μοντέλα αποθήκευσης και διαμοιρασμού αρχείων. Μεταξύ αυτών είναι και τα Κατανεμημένα Συστήματα Αρχείων, η χρήση των οποίων έχει καθιερωθεί σε πολλές μεγάλες επιχειρήσεις και ερευνητικές ομάδες, για τις δυνατότητες και τα οφέλη που προσφέρουν. Το κύριο χαρακτηριστικό τους είναι ο διαμοιρασμός των δεδομένων των χρηστών στους διαφορετικούς υπολογιστές που συγκροτούν το σύστημα. Όμως η κατανεμημένη φύση του συστήματος συνοδεύεται από εγγενής δυσκολίες και περιορισμούς ως προς τον έλεγχο της ορθής λειτουργίας του.

Έχουν αναπτυχθεί αρκετές υλοποιήσεις κατανεμημένων συστημάτων αρχείων, καθεμία με τα δικά της χαρακτηριστικά. Το HDFS είναι μία αρκετά διαδεδομένη υλοποίηση ανοικτού λογισμικού, σχεδιασμένη για να τρέχει σε υπολογιστές εμπορίου ενώ παρουσιάζει μεγάλη ανοχή σε σφάλματα. Αποτελεί το βασικό συστατικό του Apache Hadoop project, μιας πλατφόρμας σχεδιασμένης για την αποθήκευση και επεξεργασία μεγάλων όγκων δεδομένων.

Τα αρχεία που αποθηκεύονται στο σύστημα δεν βρίσκονται υπό την άμεση κατοχή του χρήστη, αλλά τοποθετούνται σε κάποιον ίσως απομακρυσμένο υπολογιστή. Έτσι, σε περίπτωση που κάποιοι από τους επιμέρους υπολογιστές που συγκροτούν το σύστημα, δεν

αποτελούν έμπιστες οντότητες ή δε συμμορφώνονται στο πρωτόκολλο λειτουργίας του συστήματος, υπονομεύεται η εγγύηση για ασφάλεια και ακεραιότητα των δεδομένων.

1.1 Αντικείμενο της διπλωματικής

Σκοπός της διπλωματικής αυτής είναι η ενίσχυση του HDFS, εισάγοντας μία έμπιστη τρίτη οντότητα, βασισμένη στο Ethereum blockchain, που θα λειτουργεί ως αρχή ελέγχου και θα εγγυάται την ακεραιότητα των δεδομένων που αποθηκεύονται στο σύστημα. Όλοι οι κόμβοι του συστήματος είναι υποχρεωμένοι να κατασκευάζουν και να υποβάλλουν αποδείξεις για την ακεραιότητα των δεδομένων που διαθέτουν ανά τακτά χρονικά διαστήματα, οι οποίες επαληθεύονται μέσω ενός έξυπνου συμβολαίου. Τα αποτελέσματα των ελέγχων είναι διαθέσιμα προς τους χρήστες, ώστε να μπορούν να αποφανθούν σχετικά με τυχόν αλλοιώσεις ή τροποποιήσεις στα δεδομένα τους. Λόγω της δημόσιας φύσης και διαφάνειας των συναλλαγών στο blockchain, αποφεύγεται η αποστολή του περιεχομένου των αρχείων. Αντί αυτού, οι αποδείξεις κατασκευάζονται και επαληθεύονται βάση της μεθόδου των zk-SNARKs, διατηρώντας έτσι την ιδιωτικότητα των δεδομένων.

1.2 Οργάνωση του τόμου

Η παρούσα διπλωματική διακρίνεται σε δύο κύρια μέρη. Το πρώτο, θεωρητικό μέρος περιλαμβάνει το θεωρητικό υπόβαθρο και τα χαρακτηριστικά των εργαλείων και τεχνολογιών που χρησιμοποιήθηκαν. Πιο συγκεκριμένα στο κεφάλαιο 2 αναφέρονται τα χαρακτηριστικά των κατανεμημένων συστημάτων αρχείων, καθώς και η αρχιτεκτονική και οι βασικές λειτουργίες του HDFS. Στο κεφάλαιο 3 παρουσιάζεται η δομή των Merkle trees, ενώ στο κεφάλαιο 4 τα βασικά γνωρίσματα του Ethereum blockchain, των έξυπνων συμβολαίων και της μεθόδου αλληλεπίδρασης ενός χρήστη με αυτά. Στο κεφάλαιο 5 αναφέρεται συνοπτικά η ιδέα των zk-SNARKs, καθώς και μία επισκόπηση του τρόπου λειτουργίας τους. Το δεύτερο, πρακτικό μέρος συγκροτείται από δύο κεφάλαια. Στο κεφάλαιο 6 παρουσιάζεται η μέθοδος που ακολουθήσαμε για την υλοποίηση του συστήματος μέσω της τροποποίησης του HDFS, μαζί με τις σχεδιαστικές αποφάσεις που τη συνοδεύουν. Τέλος, στο κεφάλαιο 7 παρουσιάζεται η μεθοδολογία και τα αποτελέσματα της πειραματικής αξιολόγησης του συστήματος.

Μέρος Ι

Θεωρητικό Μέρος

Κεφάλαιο 2

Κατανεμημένο Σύστημα Αρχείων

2.1 Σύστημα Αρχείων

2.1.1 Τι είναι ένα σύστημα αρχείων

Στην επιστήμη της πληροφορική, ως σύστημα αρχείων ορίζεται η μέθοδος αποθήκευσης και οργάνωσης των δεδομένων των αρχείων που χρησιμοποιούνται σε έναν ηλεκτρονικό υπολογιστή, καθώς και οι λογικοί κανόνες που τη διέπουν [2]. Χώρις αυτό, τα αποθηκευμένα αρχεία θα αποτελούσαν ένα μεγάλο αδόμητο κομμάτι δεδομένων, χωρίς κανένα τρόπο να τα ξεχωρήσουμε ή να προσδιορίσουμε την ακριβή τους τοποθεσία. Διαιρώντας τα δεδομένα σε μικρότερα τμήματα και αποδίδοντας τους ξεχωριστά αναγνωριστικά ονόματα, επιτυγχάνεται η εύκολη απομόνωση και αναγνώρισή τους από το σύνολο.

Τα συστήματα αρχείων χρησιμοποιούνται σε πολλές διαφορετικές συσκευές αποθήκευσης δεδομένων, όπως οι σκληροί δίσκοι ή τα CD-ROM για να παρακολουθείται η φυσική θέση των αρχείων πάνω στο μέσο. Σε ορισμένες περιπτώσεις, όπως στο tmpfs, μπορεί και η κύρια μνήμη του υπολογιστή (μνήμη τυχαίας προσπέλασης, RAM) να χρησιμοποιηθεί για τη δημιουργία ενός προσωρινού συστήματος αρχείων για βραχυπρόθεσμη χρήση.

Εκτός από τα συστήματα αρχείων που χρησιμοποιούνται σε μια τοπική συσκευή αποθήκευσης, υπάρχουν και αυτά που παρέχουν πρόσβαση σε δεδομένα σε ένα διακομιστή αρχείων μέσω ενός πρωτοκόλλου δικτύου, όπως το NFS ή το SMB. Επιπλέον, ένα σύστημα αρχείων μπορεί είναι και εικονικό, δηλαδή τα αρχεία στα οποία παρέχει πρόσβαση δε βρίσκονται αποθηκευμένα σε κάποιο μέσο, αλλά υπολογίζονται κατά την αίτηση για ανάκτησή τους (procfs, sysfs).

Τελικά, ένα σύστημα αρχείων διαφέρει από μία απλή υπηρεσία καταλόγου ή μητρώου, ενώ ορίζεται ως η δομημένη αναπαράσταση και διαχείρηση ενός συνόλους δεδομένων, μαζί με τα μεταδεδομένα που το χαρακτηρίζουν.

2.1.2 Ιδιότητες ενός συστήματος αρχείων

Ένα σύστημα αρχείων, ως μια διεπαφή για αλληλεπίδραση των προγραμμάτων χρήστη με ένα αποθηκευτικό μέσο, έχει πολλές ιδιότητες που το χαρακτηρίζουν. Μερικές από τις πιο σημαντικές ιδιότητες είναι οι εξής:

- **Διαχείρηση αποθηκευτικού χώρου**

Τα περισσότερα συστήματα αρχείων κάνουν χρήση μιας συσκευής αποθήκευσης δεδομένων η οποία προσφέρει πρόσβαση σε έναν πίνακα φυσικών τομέων ορισμένου μεγέθους, κατά κανόνα μια δύναμη του 2 σε μέγεθος (512 bytes ή 1, 2, ή 4 KiB είναι οι πιο συχνές). Το σύστημα αρχείων είναι υπεύθυνο για να οργανώσει τους τομείς αυτούς σε αρχεία και καταλόγους, καθώς και για την παρακολούθηση του ποιοι τομείς ανήκουν σε ποιο αρχείο και ποιοι δεν χρησιμοποιούνται. Για τη διαχείρηση λοιπόν των δεδομένων, τα αντιστοιχούν σε μονάδες σταθερού μεγέθους, αποκαλούμενες και ως μπλοκ (ή δέσμες), που περιέχουν ένα συγκεκριμένο αριθμό τομέων δίσκου (συνήθως 1-64) που είναι και το μικρότερο ποσό χώρου στο δίσκο που μπορεί να διατεθεί για να διατηρήσει ένα αρχείο. Αυτό όμως έχει ως αποτέλεσμα να υπολοίπεται χώρος σε περιπτώσεις αρχείων που το μέγεθος τους δεν είναι ακριβές πολλαπλάσιο του μεγέθους των μπλοκ, ο οποίος δεν μπορεί να αξιοποιηθεί. Αφετέρου, η χρήση ενός μκρού μεγέθους μπλοκ, μπορεί να οδηγήσει σε επιπλέον κόστος διαχείρησης του αποθηκευτικού χώρου αν προορίζεται για αποθήκευση κυρίως μεγάλου μεγέθους αρχείων.

- **Ονόμα αρχείου**

  Ως όνομα αρχείου, ορίζεται το όνομα που αποδίδεται σε ένα αρχείο κατά την αποθήκευσή του, και χρησιμοποιείται για τον προσδιορισμό της τοποθεσίας του κατά μονοσήμαντο τρόπο. Σε ορισμένα συστήματα αρχείων, τα ονόματα αρχείων είναι δομημένα, με ιδιαίτερη σύνταξη για τις επεκτάσεις τους και τους αριθμούς έκδοσης. Σε άλλες περιπτώσεις, τα ονόματα αρχείων είναι απλές φράσεις και τα μεταδεδομένα κάθε αρχείου αποθηκεύονται αλλού.

- **Κατάλογος**

  Τα περισσότερα συστήματα αρχείων χρησιμοποιούν καταλόγους που επιτρέπουν στον χρήστη να ομαδοποιεί τα αρχείων σε ξεχωριστές συλλογές. Η δομή ενός καταλόγου αρχείων μπορεί να είναι επίπεδη, ή να επιτρέπει ιεραρχίες όπου κάθε κατάλογος να μπορεί να περιέχει άλλους υποκαταλόγους.

- **Μεταδεδομένα**

  Πολλές πληροφορίες καταγραφής συνήθως συνοδεύουν κάθε αρχείο. Το μέγεθος και η τοποθεσία του αρχείου, καθώς και η χρονική στιγμή δημιουργίας, προσπέλασης και τελευταίας τροποποίησης του είναι μερικές από αυτές. Άλλες χρήσιμες πληροφορίες αποτελούν ο τύπος και τα δικαιώματα πρόσβασης κάθε αρχείου, που προσδιορίζουν πως και από ποιους χρήστες μπορεί να χρησιμοποιηθεί. Τα μεταδεδομένα αποθηκεύονται και ενημερώνονται από το σύστημα αρχείων συνήθως σε ξεχωριστή τοποθεσία από τα δεδομένα στα οποία αντιστοιχούν. Εκτός από τα μεταδεδομένα που χαρακτηρίζουν τα αρχεία, υπάρχουν και άλλες παράμετροι αναγκαίες για τη λειτουργία του συστήματος αρχείων, όπως το μέγεθος των μπλοκ και τα διαθέσιμα ή ελαττωματικά τμήματα του αποθηκευτικού μέσου.

- **Έλεγχος πρόσβασης**

  Υπάρχουν διάφοροι μηχανισμοί που χρησιμοποιούνται από τα συστήματα αρχείων για τον έλεγχο της πρόσβασης στα δεδομένα. Συνήθως ο σκοπός είναι να αποτραπεί η

ανάγνωση ή η τροποποίηση αρχείων από έναν χρήστη ή ομάδα χρηστών. Ένας άλλος λόγος είναι να διασφαλιστεί ότι τα δεδομένα τροποποιούνται με ελεγχόμενο τρόπο, επιτρέποντας την πρόσβαση μόνο σε ένα συγκεκριμένο πρόγραμμα.

- **Ακεραιότητα**

  Μια σημαντική ευθύνη ενός συστήματος αρχείων είναι η διασφάλιση της συνέπειας των δομών του, ανεξάρτητα από τις ενέργειες των προγραμμάτων που που έχουν πρόσβαση σε αυτό. Κάποιες από τις πιθανές ενέργειες που λαμβάνει μπορεί να είναι η ενημέρωση των μεταδεδομένων και των χειρισμό δεδομένων που βρίσκονται αποθηκευμένα στην προσωρινή μνήμη τα οποία δεν έχουν ενημερωθεί στο φυσικό μέσο αποθήκευσης επειδή το πρόγραμμα που τα τροποποιούσε τερμάτησε με μη αναμενόμενο τρόπο. Άλλες πιθανές αστοχίες που μπορεί να προκύψουν είναι αποτυχίες του λειτουργικού συστήματος, ή διακοπή της τροφοδοσίας, από το οποίες το σύστημα αρχείων πρέπει να μπορεί να ανανήψει.

2.2 Κατανεμημένο Σύστημα

Τα δίκτυα υπολογιστών έχουν μπει για τα καλά στην καθημερινή μας ζωή. Το Διαδίκτυο (Internet) είναι ένα τέτοιο δίκτυο, όπως επίσης είναι και τα δίκτυα κινητής τηλεφωνίας, τα εταιρικά δίκτυα, τα δίκτυα στα πανεπιστήμια, τα οικιακά ασύρματα δίκτυα, καθώς και τα δίκτυα στα αυτοκίνητα. Όλα αυτά τα δίκτυα έχουν τα κοινά χαρακτηριστικά που ονομάζουμε κατανεμημένα συστήματα.

2.2.1 Τι είναι ένα κατανεμημένο σύστημα

Ως κατανεμημένο σύστημα ορίζουμε ένα σύστημα που αποτελείται από δομοστοιχεία λογισμικού και υλισμικού, τα οποία βρίσκονται σε διαφορετικούς αλλά διασυνδεδεμένους μεταξύ τους υπολογιστές. Τα δομοστοιχεία αυτά συντονίζουν τις δράσεις τους μόνο μέσω ανταλλαγής μηνυμάτων [3].

Οι διασυνδεδεμένοι υπολογιστές σε ένα δίκτυο μπορεί να είναι γεωγραφικά απομακρυσμένοι, και να βρίσκονται σε οποιαδήποτε απόσταση μεταξύ τους. Μπορεί να βρίσκονται σε διαφορετικές ηπείρους, ή στο ίδιο κτίριο, ή ακόμα και στο ίδιο δωμάτιο.

Βάση αυτού του γενικού ορισμού εξάγουμε τρία βασικά γνωρίσματα των κατανεμημένων συστημάτων:

- **Ταυτοχρονισμός**

  Σε ένα δίκτυο υπολογιστών, η ταυτόχρονη εκτέλεση πολλαπλών προγραμμάτων και εφαρμογών είναι η νόρμα. Για παράδειγμα, μπορεί ένα σύνολο χρηστών να δουλεύουν σε διαφορετικούς υπολογιστές, και την ίδια στιγμή να μοιράζονται όλοι τις ίδιες πηγές πληροφορίας, όπως ιστοσελίδες ή αρχεία, ανάλογα με τις ανάγκες της εργασίας τους. Οπότε, η ικανότητα ενός συστήματος να διαχειριστεί κοινούς πόρους, πηγές δεδομένων και υπηρεσίες πρέπει να αυξάνεται καθώς προστίθενται περισσότεροι πόροι στο δίκτυο (π.χ. υπολογιστές).

- **Απουσία κοινού καθολικού χρόνου σε όλο το σύστημα**
  Όταν χρειάζεται οι κατανεμημένες εφαρμογές λογισμικού να συντονίσουν τις λειτουργίες τους, ανταλλάσσουν μηνύματα. Ο συντονισμός συχνά βασίζεται στην πληροφορία που περιγράφει τον ακριβή χρόνο εκτέλεσης της κάθε λειτουργίας. Όμως υπάρχουν περιορισμοί ως προς τη χρονική ακρίβεια την οποία οι διαφορετικοί υπολογιστές και εφαρμογές λογισμικού μπορούν να πετύχουν κατά τον συγχρονισμό τον ρολογιών τους. Ουσιαστικά, δεν υπάρχει μία και μοναδική ένδειξη κοινού καθολικού φυσικού χρόνου ανάμεσα σε διαφορετικούς υπολογιστές, ειδικά εάν αυτοί είναι αρκετά απομακρυσμένοι μεταξύ τους.

- **Ανεξάρτητες αστοχίες**
  Όλα τα συστήματα υπολογιστών μπορεί να αστοχήσουν κάποια στιγμή. Είναι στην ευθύνη των σχεδιαστών των συστημάτων να μεθοδεύσουν τη διαχείριση των συνεπειών που θα έχουν αυτές οι αστοχίες. Ειδικότερα, τα κατανεμημένα συστήματα μπορούν να αστοχήσουν με πολλούς διαφορετικούς τρόπους. Για παράδειγμα, αστοχίες στα δίκτυα επικοινωνίας έχουν σαν αποτέλεσμα την αποκοπή υπολογιστών και των εφαρμογών τους από το σύστημα, όμως αυτό δε σημαίνει ότι οι ίδιοι δε θα πρέπει να συνεχίσουν να επιτελούν τις λειτουργίες τους. Επιπροσθέτως, αυτοί οι υπολογιστές και οι εφαρμογές τους μπορεί να μην είναι σε θέση να διαχωρίσουν εάν το δίκτυο αστόχησε, ή απλώς είναι πολύ αργό. Ομοίως, η αστοχία ενός υπολογιστή, ή μιας εφαρμογής λογισμικού, δε γίνεται άμεσα γνωστή στα υπόλοιπα στοιχεία τους συστήματος με τα οποία επικοινωνεί.

Το βασικό κίνητρο για τη σχεδίαση κατανεμημένων συστημάτων είναι η κοινή χρήση πόρων. Η έννοια του πόρου είναι μεν κάτι αφηρημένο, αλλά χρησιμοποιείται για να ορίσει μία γενική γκάμα από στοιχεία τα οποία μπορούν να χρησιμοποιηθούν σε ένα κατανεμημένο σύστημα. Για παράδειγμα, πόρος μπορεί να είναι τα δεδομένα σε μια βάση δεδομένων, ή τα αρχεία σε ένα σύστημα αρχείων, μπορεί να είναι ένας υπολογιστής, μπορεί να είναι μια υπηρεσία που προσφέρεται σε έναν υπολογιστή, ή ακόμη και μια ροή εικόνων από μια ψηφιακή κάμερα, ή η σύνδεση ενός κινητού τηλεφώνου.

2.2.2 Τεχνικές προκλήσεις

Καθώς το εύρος και η πολυπλοκότητα των κατανεμημένων συστημάτων και των εφαρμογών τους αυξάνεται, τόσο ενισχύονται και οι σχεδιαστικές προκλήσεις:

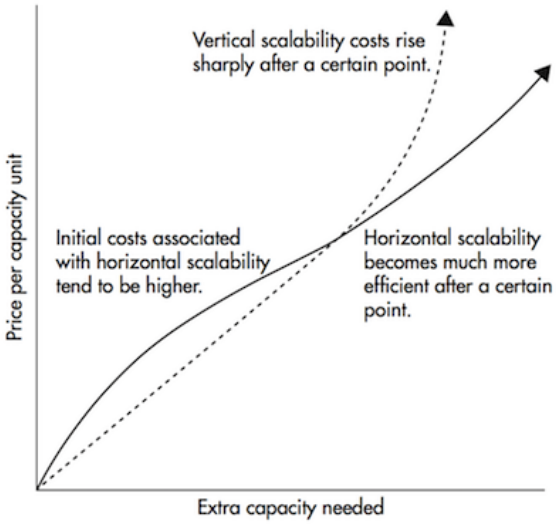
- **Ετερογένεια**
  Η ετερογένεια έχει να κάνει με τα διαφορετικά είδη δικτύων, λειτουργικών συστημάτων, υλισμικού και λογισμικού, καθώς και υλοποιήσεις εφαρμογών από διαφορετικούς μηχανικούς. Τα πρωτόκολλα διαδικτυακής επικοινωνίας σε συνδυασμό με χρήση περιβαλλόντων μεσολογισμικού (middleware) είναι σε θέση να συγκαλύψουν αυτές τις διαφορές.

- **Διαφάνεια διεπαφών ή ανοικτά συστήματα**
  Ο όρος διαφάνεια διεπαφών (openess) σε ένα υπολογιστικό σύστημα αναφέρεται στο χαρακτηριστικό το οποίο προσδιορίζει κατά πόσο το σύστημα μπορεί να επεκταθεί ή να

επαναϋλοποιηθεί με διαφορετικούς τρόπους. Η ευελιξία των κατανεμημένων συστημάτων προσδιορίζεται κυρίως από το βαθμό ευκολίας με τον οποίο νέες υπηρεσίες διαμοιρασμού και διαχείρισης πόρων μπορούν να προστεθούν στο υπάρχον σύστημα και να γίνουν διαθέσιμες προς χρήση σε διαφορετικές εφαρμογές πελάτες. Τα ανοικτά συστήματα χαρακτηρίζονται από το γεγονός ότι οι βασικές προγραμματιστικές διεπαφές είναι δημοσιευμένες. Άρα τα ανοικτά κατανεμημένα συστήματα βασίζονται στη χρήση κοινά προσυμφωνημένων μηχανισμών επικοινωνίας και σαφώς προδιαγεγραμμένων και δημοσιευμένων προγραμματιστικών διεπαφών (APIs) για την πρόσβαση σε κοινούς πόρους. Αν και τα ανοικτά κατανεμημένα συστήματα μπορούν να κατασκευαστούν από ετερογενή δομοστοιχεία υλισμικού και λογισμικού, και τα οποία προέρχονται από διαφορετικούς κατασκευαστές, καθίσταται αναγκαία η συμμόρφωση κάθε τέτοιου δομοστοιχείου σε σχέση με τα υποκείμενα πρωτόκολλα προκειμένου να διασφαλισθεί η ορθή λειτουργία του συστήματος.

- **Ασφάλεια**
  Πολλές από τις πληροφορίες που είναι διαθέσιμες και συντηρούνται σε ένα κατανεμημένο σύστημα έχουν μεγάλη αξία για τους χρήστες. Η ασφάλεια λοιπόν αυτών των πληροφοριών αποτελεί ζήτημα μείζονος σημασίας. Η ασφάλεια των πηγών πληροφορίας σε ένα τέτοιο σύστημα έχει τρεις παραμέτρους: Εμπιστευτικότητα-Confidentiality (προστασία των πληροφοριών από μη εντεταλμένους χρήστες), Ακεραιότητα-Integrity (προστασία ενάντια σε μη εντεταλμένη αλλαγή ή την παραφθορά των δεδομένων) και Διαθεσιμότητα-Availability (προστασία ενάντια σε παρεμβολές όσον αφορά στην πρόσβαση και χρήση πόρων). Συνήθως, η χρήση τεχνικών κρυπτογράφησης μας προσφέρουν ικανοποιητική προστασία όσον αφορά τη μη εξουσιοδοτημένη πρόσβαση σε πόρους που υπόκεινται σε κοινή χρήση, καθώς και την πρόσβαση τρίτων κατά τη διάρκεια μετάδοσης δεδομένων μεταξύ διαφορετικών υπολογιστών μέσω του δικτύου.

- **Κλιμακωσιμότητα**
  Ένα κατανεμημένο σύστημα είναι κλιμακώσιμο ή επεκτάσιμο, εάν το κόστος για να προσθέσουμε καινούργιους χρήστες είναι γραμμικό σε σχέση με τους πόρους που απαιτούνται για να προσφερθούν λειτουργίες σε αυτούς. Οι αλγόριθμοι οι οποίοι χρησιμοποιούνται ώστε πολλοί χρήστες να έχουν ταυτόχρονη πρόσβαση στα ίδια δεδομένα πρέπει να είναι αποδοτικοί και να αποφεύγεται η διαμόρφωση στενωπών, που υπονομεύουν τελικά την απόδοση. Επίσης τα δεδομένα θα πρέπει να δομούνται ιεραρχικά έτσι ώστε το σύστημα να έχει τη μέγιστη απόδοση. Δεδομένα τα οποία προσπελαύνονται πολύ συχνά μπορούν να υπάρχουν σαν πιστά αντίγραφα σε διαφορετικούς διακομιστές. Αξίζει να σημειωθεί πως στα κατανεμημένα συστήματα έχουμε συνήθως οριζόντια κλιμάκωση με προσθήκη περισσότερων υπολογιστών στο σύστημα, η οποία εμφανίζει σημαντικά πλεονεκτήματα έναντι της κάθετης κλιμάκωσης που αφορά την αναβάθμιση του υλισμικού ενός υπολογιστή (Εικόνα 2.1).



Σχήμα 2.1: *Οριζόντια έναντι κάθετης επέκτασης συστήματος*

- **Διαχείριση αστοχιών**
  Δεδομένου ότι κάθε διαδικασία ή δομοστοιχείο σε ένα δίκτυο υπολογιστών μπορεί να αστοχήσει ανεξάρτητα από τα άλλα, ο σχεδιασμός ενός κατανεμημένου συστήματος πρέπει να πραγματοποιείται με γνώμονα την αντοχή σε τέτοιου είδους σφάλματα σφάλματα. Πρέπει δηλαδή να υπάρχουν μηχανισμοί ανίχνευσης, συγκάληψης και ανάνηψης από τυχόν αστοχίες που μπορεί να προκύψουν. Επιπλέον, μια από τις πιο συνηθισμένες τεχνικές πρόληψης των μερικών αστοχιών ενός συστήματος είναι ο πλεονασμός πόρων.

- **Ταυτοχρονισμός**
  Η παρουσία πολλών χρηστών σε ένα κατανεμημένο σύστημα συνοδεύεται από την ανάγκη για παράλληλη πρόσβαση σε κοινούς πόρους. Κάθε πόρος σε ένα κατανεμημένο σύστημα θα πρέπει να έχει σχεδιαστεί ώστε να λειτουργεί με ασφάλεια σε ένα περιβάλλον όπου εκδίδονται αιτήματα πρόσβασης σε αυτόν παράλληλα και ταυτόχρονα από διαφορετικούς χρήστες.

- **Διαφάνεια**
  Στόχος είναι η συγκάλυψη από τον χρήστη, των τεχνικών λεπτομερειών ενός κατανεμημένου συστήματος. Για παράδειγμα ένας χρήστης δεν χρειάζεται να γνωρίζει που βρίσκεται μία υπηρεσία, εάν αυτή έχει ένα ή περισσότερα πιστά αντίγραφα στο Διαδίκτυο, και ποιες είναι οι τεχνικές λεπτομέρειες υλοποίησης της. Ακόμα και οι πιθανές αστοχίες του δικτύου θα πρέπει να λαμβάνονται υπόψιν κατά το σχεδιασμό και την υλοποίηση ώστε να αντιμετωπίζονται κατάλληλα χωρίς να γίνονται αντιληπτές στον τελικό χρήστη.

- **Ποιότητα υπηρεσίας (QoS)**
  Από τη στιγμή που οι χρήστες είναι σε θέση να χρησιμοποιούν μια υπηρεσία, όπως για παράδειγμα μια υπηρεσία προσπέλασης αρχείων σε ένα κατανεμημένο σύστημα, είναι θεμιτή η μέριμνα για την ποιότητα της προσφερόμενης υπηρεσίας. Οι βασικές

μη-λειτουργικές απαιτήσεις των συστημάτων οι οποίες επηρεάζουν την ποιότητα της υπηρεσίας , όπως αυτή εννοείται από τους χρήστες, είναι η αξιοπιστία (reliability), η ασφάλεια (security) και η απόδοση (performance). Συμπληρωματικά σε αυτές, η προσαρμοστικότητα (adaptability) έχει προταθεί σαν μία ακόμη μη-λειτουργική απαίτηση που επηρεάζει την ποιότητα υπηρεσίας, ειδικά σε συστήματα των οποίων η διαμόρφωση και η διαθεσιμότητα των πόρων παρουσιάζει συχνές μεταβολές.

2.3 Τι είναι ένα κατανεμημένο σύστημα αρχείων

Ένα κατανεμημένο σύστημα αρχείων (DFS) είναι μια κατανεμημένη υλοποίηση του παραδοσιακού συστήματος αρχείων, η οποία επιτρέπει σε χρήστες και προγράμματα να διαχειρίζονται και να μοιράζονται δεδομένα, ανεξάρτητα από την πραγματική τους τοποθεσία στο δίκτυο [4]. Επιπλέον παρέχει μια ενοποιημένη, λογική όψη των δεδομένων που βρίσκονται διάσπαρτα στο σύστημα, χωρίς να είναι ορατή η πραγματική τους τοποθεσία. Συνδυάζει δηλαδή την ευχρηστία των συστημάτων αρχείων με την ευελιξία των κατανεμημένων συστημάτων, συνοδευόμενα όμως και από τις αντίστοιχες σχεδιαστικές προκλήσεις.

Υπάρχουν πολλές υλοποιήσεις κατανεμημένων συστημάτων αρχείων, οι οποίες διαφέρουν ως προς την απόδοσή τους, τη μεταβλητότητα του περιεχομένου, τον χειρισμό ταυτόχρονων εγγραφών, τον χειρισμό μόνιμης ή προσωρινής απώλειας κόμβων και την πολιτική τους για την αποθήκευση του περιεχομένου των αρχείων.

2.4 Hadoop Distributed File System (HDFS)

2.4.1 Τι είναι το HDFS

Το HDFS είναι ένα κατανεμημένο σύστημα αρχείων, σχεδιασμένο να λειτουργεί σε συνηθισμένα, χαμηλού κόστους υπολογιστικά συστήματα εμπορίου. Το HDFS παρέχει υψηλής ταχύτητας πρόσβαση σε δεδομένα εφαρμογών και είναι κατάλληλο για εφαρμογές που χειρίζονται μεγάλα σύνολα δεδομένων. Μια συστάδα HDFS μπορεί να αποτελείται από εκατοντάδες ή και χιλιάδες μηχανήματα, καθένα από τα οποία αποθηκεύουν ένα τμήμα του συνόλου των δεδομένων του συστήματος αρχείων. Με τόσα συστατικά στο σύστημα, όπου το καθένα έχει μη αμελητέα πιθανότητα να αστοχήσει, κάθε χρονική στιγμή είναι πολύ πιθανό κάποιος κόμβος του συστήματος να βρίσκεται εκτός λειτουργίας. Ως εκ τούτου, η ανίχνευση αστοχιών και η γρήγορη, αυτόματη ανάνηψη από αυτές αποτελεί έναν από τους βασικότερους μηχανισμούς του HDFS, προσδίδοντάς του μεγάλη ανοχή σε σφάλματα [5].

2.4.2 Αρχιτεκτονική NameNode και DataNode

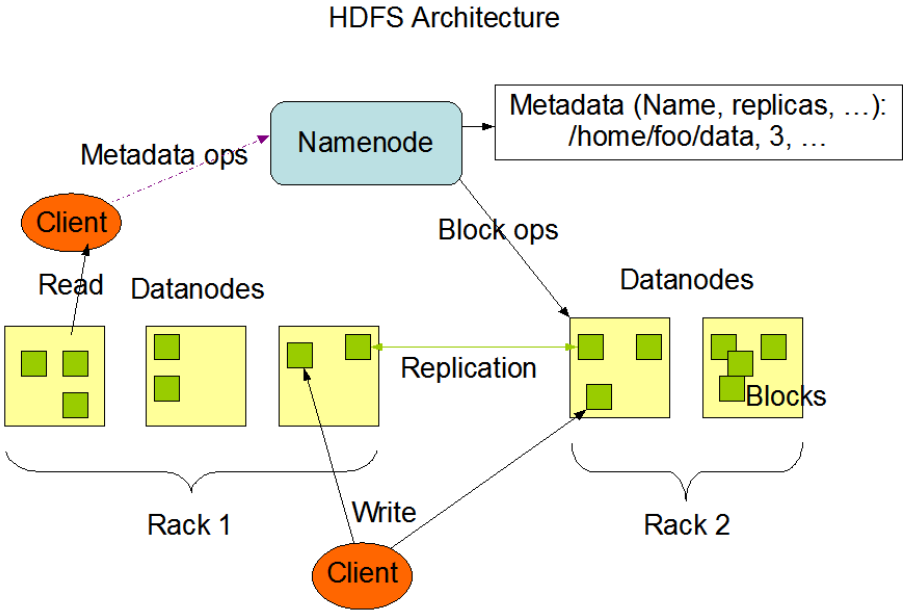
Το HDFS έχει αρχιτεκτονική master/slave. Μια συστάδα HDFS αποτελείται από έναν κύριο διακομιστή που διαχειρίζεται τον χώρο ονομάτων του συστήματος αρχείων και ρυθμίζει την πρόσβαση στα αρχεία από τους χρήστες, που ονομάζεται NameNode. Επιπλέον, συμμετέχουν ένας ή περισσότεροι DataNodes, συνήθως ένας ανά κόμβο της συστάδα, οι οποίοι και είναι υπεύθυνοι για την αποθήκευση των δεδομένων. Το HDFS διαθέτει έναν ενιαίο χώρο

ονομάτων συστήματος αρχείων, κοινό σε όλους τους κόμβους της συστάδας, επιτρέποντας την οργάνωση των δεδομένων του χρήστη σε αρχεία (αντίστοιχα με ένα τοπικό σύστημα αρχείων).

Εσωτερικά, κάθε αρχείο χωρίζεται σε ένα ή περισσότερα μπλοκ, τα οποία αποθηκεύονται σε ένα σύνολο από DataNodes. Ο NameNode εκτελεί τις λειτουργίες του χώρου ονομάτων του συστήματος αρχείων, όπως άνοιγμα, κλείσιμο και μετονομασία αρχείων και καταλόγων. Καθορίζει επίσης την αντιστοίχιση των μπλοκ σε DataNodes. Οι DataNodes είναι υπεύθυνοι για την εξυπηρέτηση των αιτημάτων ανάγνωσης και εγγραφής από τους πελάτες του συστήματος αρχείων. Επιπλέον, οι DataNodes εκτελούν λειτουργίες δημιουργίας, διαγραφής και αναπαραγωγής των μπλοκ κατόπιν εντολής από τον NameNode.

Ο NameNode και ο DataNode είναι τα δύο κύρια δομοστοιχεία λογισμικού του HDFS, τα οποία είναι υλοποιημένα σε Java και έχουν σχεδιαστεί για να τρέχουν σε συμβατικά μηχανήματα. Συνήθως τα μηχανήματα αυτά έχουν εγακτεστημένο λειτουργικό σύστημα GNU/Linux. Όμως χάρη στην εξαιρετική φορητότητα της Java, το HDFS μπορεί να λειτουργήσει σε οποιοδήποτε μηχάνημα υποστηρίζει τη συγκεκριμένη γλώσσα προγραμματισμού, καθιστώντας το προσβάσιμο από ένα ευρύ φάσμα υπολογιστικών συστημάτων χωρίς κάποια ειδική απαίτηση.

Μια τυπική τοπολογία μιας συστάδας HDFS (Εικόνα 2.2) διαθέτει ένα αποκλειστικό μηχάνημα που εκτελεί μόνο το λογισμικό του NameNode. Κάθε ένα από τα υπόλοιπα μηχανήματα τρέχουν τον κώδικα του DataNode. Η αρχιτεκντονική του συστήματος δεν απαγορεύει την εκτέλεση πολλαπλών DataNode στο ίδιο μηχάνημα, όμως κάτι τέτοιο σπάνια εφαρμόζεται σε ένα πραγματικό περιβάλλον ανάπτυξης. Η ύπαρξη ενός μόνο NameNode σε μια συστάδα απλοποιεί σημαντικά την αρχιτεκτονική του συστήματος. Ο NameNode εποπτεύει και ενορχηστρώνει τις διάφορες λειτουργίες του συστήματος, ενώ παράλληλα λειτουργεί ως αποθετήριο για όλα τα μεταδεδομένα του HDFS. Το HDFS όμως είναι σχεδιασμένο ώστε να μην υπάρχει ροή δεδομένων χρήστη μέσω του NameNode σε καμία φάση της λειτουργίας του.



Σχήμα 2.2: *Τυπική αρχιτεκτονική hdfs συστάδας*

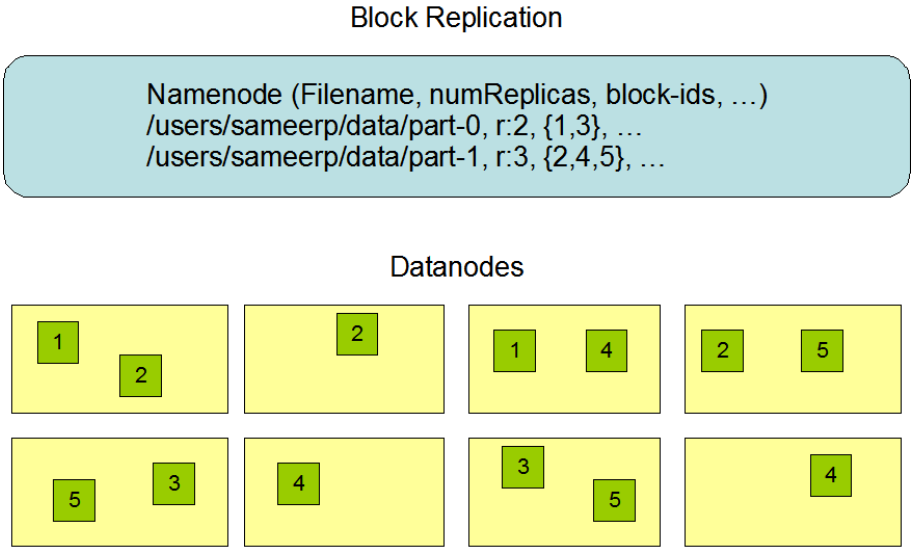
2.4.3 Χώρος ονομάτων συστήματος αρχείων

Το HDFS υποστηρίζει μια παραδοσιακή ιεραρχική οργάνωση αρχείων. Ένας χρήστης ή μια εφαρμογή μπορεί να δημιουργήσει καταλόγους και να αποθηκεύσει αρχεία μέσα σε αυτούς. Η ιεραρχία του χώρου ονομάτων είναι παρόμοια με τα περισσότερα παραδοσιακά συστήματα αρχείων. Κάθε χρήστης μπορεί να δημιουργήσει και να αφαιρέσει αρχεία, να μετακινήσει ένα αρχείο από τον έναν κατάλογο στον άλλο ή να μετονομάσει ένα αρχείο.

Ο NameNode διατηρεί τον χώρο ονομάτων του συστήματος αρχείων. Οποιαδήποτε αλλαγή στον χώρο ονομάτων ή στις ιδιότητές του, καταγράφεται από τον NameNode. Ένας χρήστης ή μια εφαρμογή μπορεί να καθορίσει τον αριθμό των αντιγράφων ενός αρχείου που αποθηκεύει στο HDFS. Ο αριθμός των αντιγράφων ενός αρχείου ονομάζεται συντελεστής αντιγραφής. Όλες αυτές οι πληροφορίες αποθηκεύονται από τον NameNode.

2.4.4 Αντίγραφα δεδομένων

Το HDFS έχει σχεδιαστεί για να αποθηκεύει αξιόπιστα πολύ μεγάλου μεγέθους αρχεία. Κάθε αρχείο αποθηκεύεται ως μια ακολουθία μπλοκ. Όλα τα μπλοκ ενός αρχείου έχουν το ίδιο μέγεθος, με εξαίρεση ίσως το τελευταίο. Για διασφάλιση ανοχής σε σφάλματα, το HDFS διατηρεί πολλαπλά αντίγραφα για κάθε μπλοκ σε διαφορετικά μηχανήματα. Ο συντελεστής αντιγραφής και το μέγεθος των μπλοκ είναι παραμετροποιήσιμα ανά αρχείο. Ειδικά στην περίπτωση του συντελεστή αντιγραφής, η αρχική παραμετροποίηση κατά τη δημιουργία του αρχείου δεν είναι δεσμευτική, αλλά μπορεί να μεταβληθεί και αργότερα. Τα αρχεία στο HDFS γράφονται μία φορά και έχουν αυστηρά έναν συντάκτη ανά πάσα στιγμή. Ο NameNode λαμβάνει όλες τις αποφάσεις σχετικά με την αντιγραφή των μπλοκ. Επιπλέον, λαμβάνει περιοδικά Heartbeat και Blockreport από καθέναν από τους DataNodes της συστάδας. Η λήψη ενός Heartbeat υποδηλώνει ότι ο DataNode λειτουργεί σωστά, ενώ ένα Blockreport περιέχει μια λίστα με όλα τα μπλοκ που αυτός διαθέτει στο τοπικό σύστημα αρχείων του.



Σχήμα 2.3: *Αντίγραφα μπλοκ σε διαφορετικούς DataNodes*

2.4.5 Ακεραιότητα δεδομένων

Ένα μπλοκ δεδομένων που έχει ληφθεί από έναν DataNode είναι πιθανό να αλλοιωθεί. Αυτό μπορεί να προκύψει λόγω σφαλμάτων του αποθηκευτικού μέσου, του δικτύου ή ακόμα και από πρόβλημα στο λογισμικού. Το λογισμικό χρήστη του HDFS χρησιμοποιεί αθροίσματα ελέγχου (checksum) για τον έλεγχο των αρχείων. Όταν ένας πελάτης δημιουργεί ένα αρχείο στο HDFS, υπολογίζει τα αθροίσματα ελέγχου για κάθε μπλοκ του αρχείου και τα αποθηκεύει σε ξεχωριστό κρυφό αρχείο στον ίδιο χώρο ονομάτων του HDFS. Όταν ένας πελάτης ανακτά τα περιεχόμενα του αρχείου, επαληθεύει ότι τα δεδομένα που έλαβε από κάθε DataNode ταιριάζουν με τα αθροίσματα ελέγχου που είναι αποθηκευμένα στο σχετικό κρυφό αρχείο. Σε περίπτωση που κάποιο άθροισμα ελέγχου διαφέρει, τότε ο πελάτης μπορεί να επιλέξει να ανακτήσει το εν λόγω μπλοκ από κάποιον άλλο DataNode που διαθέτει ένα αντίγραφο, επαναλαμβάνοντας εκ νέου την διαδικασία.

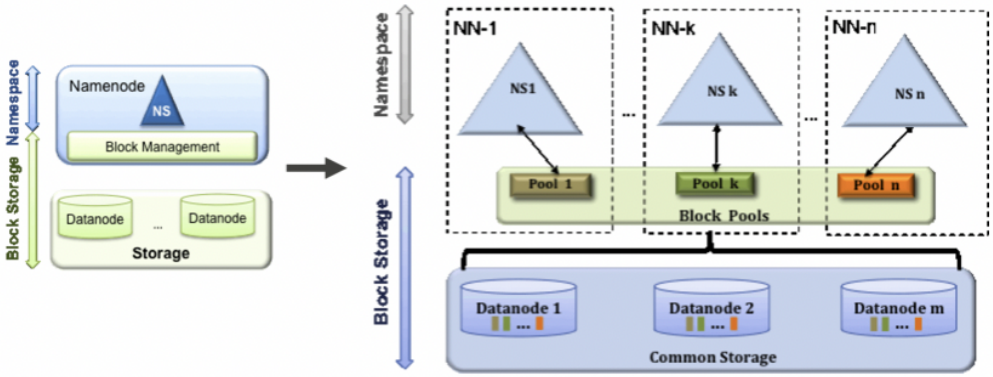
Ένας ακόμη μηχανισμός ανίχνευσης κατεστραμμένων μπλοκ στο HDFS είναι το Block Scanner. Το Block Scanner αποτελεί ένα τμήμα του λογισμικού του DataNode, το οποίο εκτελείται περιοδικά, διαβάζοντας όλα τα μπλοκ που βρίσκονται στο τοπικό σύστημα αρχείων και ελέγχοντας για πιθανές αλλοιώσεις του περιεχομένου τους. Σε περίπτωση που βρεθεί κάποιο προβληματικό μπλοκ, ο DataNode το αναφέρει στον NameNode, που με τη σειρά του μεριμνά για την αντικατάστασή του από κάποιο άλλο υγιές αντίγραφο του ίδιου μπλοκ. Έτσι το HDFS είναι σε θέση να εντοπίσει και να διορθώσει τυχόν φθορές στα δεδομένα, χωρίς να απαιτείται κάποια παρέμβαση από το χρήστη [6].

2.4.6 Ομοσπονδιακή συστάδα HDFS (federation cluster)

Η αρχιτεκτονική μιας συστάδας HDFS όπως προσδιορίστηκε ως τώρα αποτελείται από μόνο έναν NameNode που διατηρεί ένα χώρο ονομάτων, και πολλούς DataNodes που αποθηκεύουν τα δεδομένα. Η αύξηση της αποθηκευτικής ικανότητας του συστήματος επιτυγχάνεται με οριζόντια κλιμάκωση, προσθέτοντας περισσότερους DataNodes στη συστάδα.

Στην ειδική περίπτωση περίπτωση κλιμάκωσης ως προς την υπηρεσία ονομάτων του συστήματος, υποστηρίζεται η παραμετροποίηση μιας συστάδας που χρησιμοποιεί πολλούς NameNodes, καθένας εκ των οποίων διατηρεί ένα ξεχωριστό χώρο ονομάτων (Εικόνα 2.4). Οι NameNode είναι ανεξάρτητοι και δεν απαιτείται μεταξύ τους συντονισμός, ενώ οι Datanodes χρησιμοποιούνται ως κοινός αποθηκευτικός χώρος. Κάθε Datanode στέλνει περιοδικά Heartbeat και Blockreport προς όλους τους Namenodes της συστάδας. Επιπλέον χειρίζονται τις εντολές που λαμβάνουν από όλους τους NameNodes. Η επέκταση αυτή επιτρέπει την χρήση πολλών ανεξάρτητων χώρων ονομάτων, χωρίς την ανάγκη διαμοιρασμού των DataNodes σε ξεχωριστές συστάδες [7].

Τα μπλοκ που ανήκουν σε ένα χώρο ονομάτων συγκροτούν ένα σύνολο που ονομάζεται Block Pool και έχει το δικό του αναγνωριστικό. Οι DataNodes αποθηκεύουν μπλοκ για όλα τα Block Pools της συστάδας HDFS. Όμως κάθε Block Pool διαχειρίζεται ανεξάρτητα από τα υπόλοιπα. Αυτό επιτρέπει σε κάθε χώρο ονομάτων να παράγει αναγνωριστικά για κάθε καινούργιο μπλοκ χωρίς την ανάγκη συντονισμού με τα υπόλοιπους. Δηλαδή, μπορούν συνυπάρχουν μπλοκ με το ίδιο αναγνωριστικό αν αυτά ανήκουν σε διαφορετικούς χώρους ονομάτων. Η αστοχία ενός NameNode, δεν αποτρέπει τον DataNode από το να συνεχίσει να



Σχήμα 2.4: *HDFS Federation Cluster Overview*

εξυπηρετεί τους υπόλοιπους NameNodes που ανήκουν στη συστάδα. Δηλαδή, ένας χώρος ονομάτων μαζί με το εν λόγω Block Pool αποτελεί μια αυτοτελή, αυτόνομη μονάδα διαχείρισης.

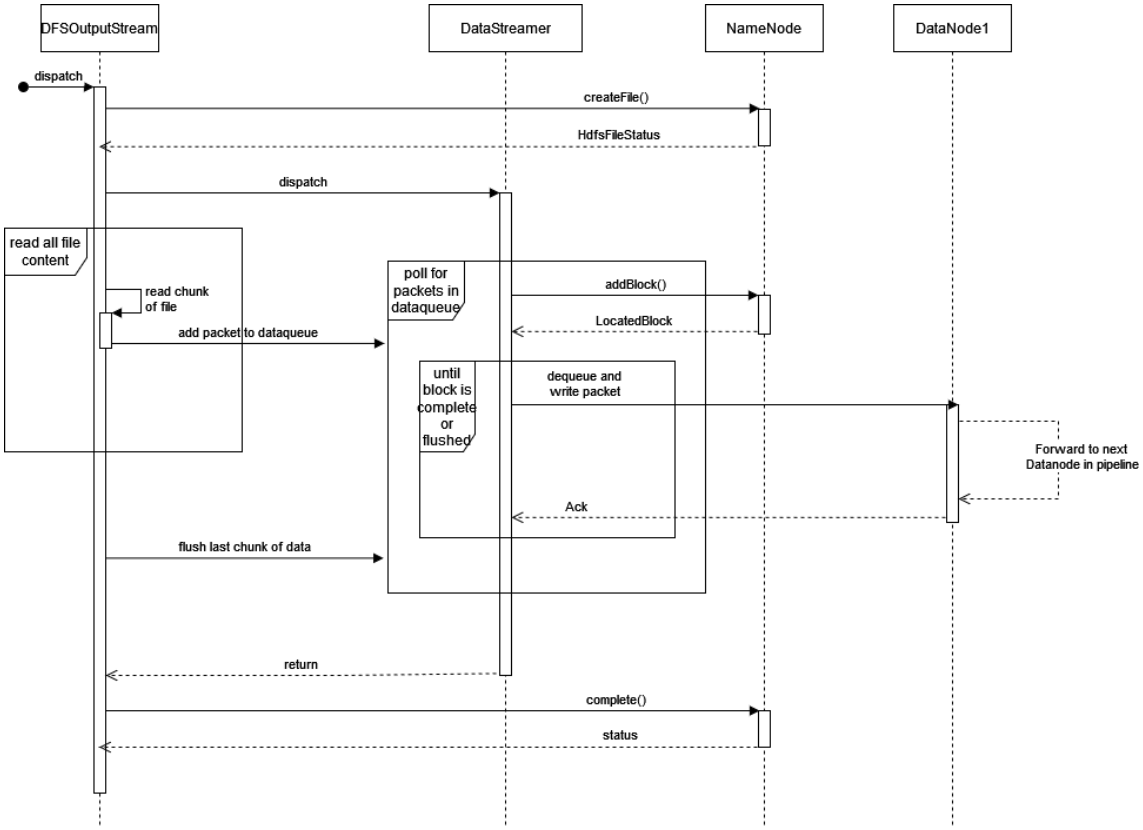
2.4.7 Γράφοντας ένα αρχείο

Κατά τη διαδικασία εγγραφής ενός αρχείου στο HDFS, το λογισμικό που εκτελεί ο χρήστης, αρχικά επικοινωνεί με τον NameNode και αιτείται τη δημιουργία ενός καινούργιου κενού αρχείου στο χώρο ονομάτων. Στη συνέχεια διαβάζοντας το αρχείο από το τοπικό σύστημα αρχείων, χωρίζει το περιεχόμενό του σε μπλοκ σταθερού μεγέθους (πιθανώς εκτός του τελευταίου που θα είναι μικρότερο). Για κάθε μπλοκ που δημιουργεί, ο χρήστης ενημερώνει τον NameNode ώστε να το προσθέσει στο σύστημα, λαμβάνοντας ως απάντηση μία ακολουθία από DataNodes στους οποίους πρέπει να μεταδοθεί το μπλοκ αυτό.

Για τη μετάδοση των δεδομένων ενός μπλοκ, προηγείται η προσωρινή εγκατάσταση ενός pipeline που συνδέει όλους τους DataNodes που πρόκειται να τα λάβουν το μπλοκ. Ο χρήστης στέλνει σταδιακά το περιεχόμενο του μπλοκ στον πρώτο DataNode του pipeline, ενώ κάθε ένας DataNode για κάθε τμήμα δεδομένων που λαμβάνει το αναμεταδίδει στον επόμενο στη σειρά, και επιστρέφει ένα μήνυμα επιβεβαίωσης στον προηγούμενο επιβεβαιώνοντας την επιτυχή εγγραφή στο τοπικό σύστημα αρχείων του. Αξίζει να σημειωθεί πως εκτός των δεδομένων του μπλοκ, μεταδίδεται και ένα ξεχωριστό αρχείο μεταδεδομένων που φέρει αθροίσματα ελέγχου (checksums) για το μπλοκ αυτό, καθιστώντας εφικτό τον έλεγχος τυχόν ανεπιθύμητων αλλοιώσεών του περιεχομένου του σε μετέπειτα στάδιο (2.4.5).

Πιο συγκεκριμένα, στην πλευρά του χρήστη εκτελούνται δύο νήματα που συνεργάζονται μεταξύ τους για την επίτευξη των παραπάνω, με τη λειτουργία τους να παρουσιάζονται στο διάγραμμα 2.5. Όπως φαίνεται, η διαδικασίες ανάγνωσης και μετάδοσης των δεδομένων αλληλοεπικαλύπτονται. Το DFSOutputStream διαβάζοντας από το τοπικό σύστημα αρχείων, χωρίζει τα δεδομένα σε μικρότερα πακέτα μεγέθους 64kb, τα οποία και προσθέτει στο τέλος μίας ουράς (FIFOδομή δεδομένων), όπου αποθηκεύονται προσωρινά για την επιτάχυνση της διαδικασίας μετάδοσής τους. Η ουρά αυτή ορίζει μια σχέση παραγωγού-καταναλωτή

μεταξύ των δύο νημάτων. Το DataStreamer για κάθε καινούριο μπλοκ που πρόκειται να μεταδοθεί, κατασκευάζει το pipeline και στη συνέχεια αφαιρεί πακέτα από την ουρά, μεταδίδοντάς τα για εγγραφή στους DataNodes του pipeline. Μόλις συμπληρωθεί ένα μπλοκ, η διαδικασία επαναλαμβάνεται κατά αντίστοιχο τρόπο για το επόμενο, έως ότου ολοκληρωθεί η εγγραφή όλου του αρχείου. Αφότου μεταδωθούν και τα τελευταία δεδομένα στην ουρά, το DFSOutputStream ενημερώνει τον NameNode για την επιτυχή ολοκλήρωση της διαδικασίας εγγραφής του αρχείου.



Σχήμα 2.5: *Διάγραμμα sequence εγγραφής αρχείου στο HDFS*

2.4.8 Heartbeat και Blockreport

Κάθε DataNode που συμμετέχει σε μια HDFS συστάδα, είναι υποχρεωμένος να στέλνει μηνύματα heartbeat στον διορισμένο NameNode, τα οποία υποδηλώνουν την παρουσία του στη συστάδα, ενώ φέρουν χρήσιμες πληροφορίες όπως τα αναγνωριστικά του κόμβου (διεύθυνση και θύρες επικοινωνίες) και την εναπομένουσα χωρητικότητά του. Ο NameNode δεν εκκινεί απευθείας συνδέσεις με τους DataNodes, αλλά μεταδίδει όλες τις απαιτούμενες οδηγίες για την διαχείρηση των αντιγράφων των μπλοκ, μέσω των απαντήσεών του στα heartbeat. Έτσι γίνεται φανερή η ανάγκη συχνής αποστολής των heartbeat για την ορθή λειτουργία του HDFS, ορίζοντας παράλληλα μια αυστηρή πολιτική όπου η απουσία λήψης heartbeat για έναν κόμβο της συστάδας (για διάστημα δέκα λεπτών από προεπιλογής), τον καθιστά προσωρινά εκτός λειτουργίας, με τα αντίγραφα μπλοκ που φέρει να θεωρούνται προσωρινά μη διαθέσιμα.

Κατά αντίστοιχο τρόπο, κάθε DataNode στέλνει ένα blokcreport στον NameNode ανά ορισμένο χρονικό διάστημα (έξι ώρες από προεπιλογής), που περιλαμβάνει μία λίστα με όλα τα μπλοκ που ισχυρίζεται πως κατέχει στο τοπικό σύστημα αρχείων του. Η διαδικασία αυτή συμβάλει στον συντονισμό των μεταδεδομένων του NameNode με τις πραγματικές τοποθεσίες των μπλοκ στη συστάδα. Εκτός από την αυτόματη δρομολόγηση της εκτέλεσης ενός blockreport, παρέχεται και η δυνατότητα επιβολής του από έναν χρήστη του συστήματος, στοχευμένα για κάποιον DataNode. Αξίζει να αναφερθεί πως κατά τη διάρκεια ενός blockreport ο DataNode αδυνατεί να διεκπεραιώσει οποιαδήποτε άλλη λειτουργία, καθιστώντας τον προσωρινά μη αποκρίσιμο.

Κεφάλαιο 3

Merkle trees

3.1 Τι είναι ένα Merkle tree

Ένα Merkle tree, γνωστό και ως hash tree, είναι μια δενδρική δομή δεδομένων όπου κάθε κόμβος-φύλλο του δέντρου περιέχει το hash ενός τμήματος δεδομένων. Κάθε ενδιάμεσος κόμβος φέρει το hash της συνένωσης του περιεχομένου των θυγατρικών του κόμβων. Δεδομένου ότι ένα hash μπορεί να ερμηνευθεί ως η σύνοψη των αρχικών δεδομένων που κατακερματίζονται, η ρίζα του Merkle tree εξαρτάται από το σύνολο των δεδομένων βάση του οποίου κατασκευάζεται το δέντρο. Δηλαδή, οποιαδήποτε αλλαγή σε ένα τμήμα δεδομένων, έχει ως αποτέλεσμα την μεταβολή των hashes του μονοπατιού από το αντίστοιχο φύλλο του δέντρου έως και της ρίζας. Για τον υπολογισμό των hashes χρησιμοποιείται μια κρυπτογραφική συνάρτηση κατακερματισμού, με αρκετά συνήθη επιλογή τη SHA-2. Η πλειοψηφία των υλοποιήσεων των Merkle trees είναι δυαδικά δέντρα (κάθε ενδιάμεσος κόμβος έχει δύο θυγατρικούς κόμβους), αλλά δεν αποκλείεται η χρήση περισσότερων θυγατρικών κόμβων σε ορισμένες εφαρμογές που το απαιτούν.

Το μοντέλο των Merkle trees, προτάθηκε από των Ralph Merkle στις αρχές της δεκαετίας του 1980, από τον οποίο και έλαβε το όνομα του. Ακόμα και σήμερα αποτελεί μια διαδεδομένη δομή δεδομένων, με εφαρμογή σε γνωστές τεχνολογίες όπως το blockchain, το Git και σε No-SQL κατανεμημένες βάσεις δεδομένων, επιτρέποντας τη γρήγορη και ασφαλή επαλήθευση της συνέπειας και του περιεχομένου μεγάλων συνόλων δεδομένων [8].

3.2 Κρυπτογραφική συνάρτηση κατακερματισμού

Μια κρυπτογραφική συνάρτηση κατακερματισμού (ή κρυπτογραφική συνάρτηση σύνοψης) είναι ένας μαθηματικός αλγόριθμος που απεικονίζει συμβολοσειρές δυαδικών ψηφίων αυθαίρετου μήκους σε συμβολοσειρές σταθερού μήκους [9]. Ιδανικά, η έξοδος μιας τέτοιας συνάρτησης, πρέπει να φαίνεται όσο το δυνατόν τυχαία και να μπορεί να υπολογιστεί σε πολυωνυμικό χρόνο ως προς το μήκος της εισόδου. Επιπλέον, μια κρυπτογραφική συνάρτηση κατακερματισμού, $H : X \to Y$, έχει τα εξής χαρακτηριστικά:

- **Μη-αντιστρέψιμη ή μονόδρομη συνάρτηση**: Δεδομένης μίας εξόδου $H(x)$, δεν μπορεί να βρεθεί το x σε πολυωνυμικό χρόνο. Ιδανικά, θα θέλαμε να μην μπορείνα βρεθεί η είσοδο, παρά μόνο με εξαντλητική αναζήτηση.

- **Διάχυση (avalanche effect)**: Μια μικρή αλλαγή μόλις ενός bit της εισόδου, έχει ως αποτέλεσμα την μεταβολή τουλάχιστον των μισών bits της εξόδου.
- **Ντετερμινισμός**: Για μια δεδομένη είσοδο, η συνάρτηση κατακερματισμού θα πρέπει να επιστρέφει πάντα το ίδιο αποτέλεσμα για την είσοδο αυτή.
- **Αντίσταση εύρεσης συγκρούσεων**: Δεν μπορούν να βρεθούν σε πολυωνυμικό χρόνο δύο συμβολοσειρές x, y τέτοιες ώστε να ισχύει $x \neq y$ και $H(x) = H(y)$.
- **Μη προβλέψιμη**: Η έξοδος πρέπει να φαίνεται τυχαία και να μην παρουσιάζει καμία συσχέτιση με την είσοδο.

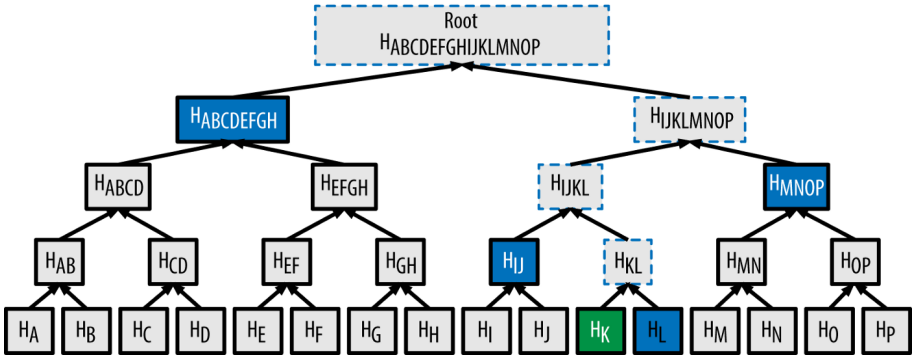
3.3 Merkle proofs

Τα Merkle proofs χρησιμοποιούνται για την απόδειξη των παρακάτω θέσεων:

- Τα δεδομένα ανήκουν στο Merkle tree.
- Την απόδειξη της εγκυρότητας ενός τμήματος δεδομένων ενός dataset, χωρίς την ανάγκη αποθήκευσης ολόκληρου του dataset.
- Την επαλήθευση της εγκυρότητας ενός συνόλου δεδομένων που περιλαμβάνονται σε ένα μεγαλύτερο dataset, χωρίς την αποκάλυψη ούτε του πλήρους dataset, ούτε του υποσυνόλου του.

Ένα Merkle proof επαληθεύεται ξεκινώντας από το εν λόγω φύλλο, και υπολογίζοντας τους ενδιάμεσους κόμβους του μονοπατιού προς της ρίζα. Στο τέλος ελέγχεται αν η ρίζα του δέντρου περιέχει το σωστό hash, δηλαδή είναι η ίδια με τη δημοσιευμένη. Η ορθότητα των αποδείξεων βασίζεται στην ανθεκτικότητα σε συγκρούσεις των κρυπτογραφικών συναρτήσεων κατακερματισμού.

Ακολουθεί ένα παράδειγμα για μεγαλύτερη σαφήνεια ως προς τη λειτουργία των Merkle Proofs:



Σχήμα 3.1: *Παράδειγμα Merkle proof*

Έστω ότι έχει κατασκευαστεί το Merkle tree (Εικόνα 3.1) για ένα dataset, και έχει δημοσιευτεί η ρίζα του. Προκειμένου να αποδείξουμε ότι τα δεδομένα [K] ανήκουν στο dataset, δεν απαιτείται να αποκαλυφθεί το περιεχόμενο των δεδομένων κανενός υποσυνόλου του dataset, αλλά μόνο ορισμένα hashes του δέντρου.

1. Αρχικού εφαρμόζουμε την κρυπτογραφική συνάρτηση κατακερματισμού στο [K] και λαμβάνουμε το H(K).

2. Ο κατακερματισμός της συνένωσης του H(K) με το hash του αγνώστου τμήματος δεδομένων L, μας δίνει το H(KL).

3. Το H(KL) σε συνδυασμό με το H(IJ) παράγει το H[IJKL].

4. Τα H(IJKL) και H(MNOP) κατακερματίζονται παράγοντας το H(IJKLMNOP).

5. Τέλος, από τον κατακερματισμό της συνένωσης των H(IJKLMNOP) και H(ABCDEFGH), προκύπτει το H(ABCDEFGHIJKLMNOP) που ταυτίζεται με την δημόσια ρίζα.

Έτσι, καταφέραμε να αποδείξουμε πως τα δεδομένα [K] πράγματι ανήκουν στο Merkle tree, άρα και στο ευρύτερο dataset [A, B, C, ... ,N, O, P], χρησιμοποιώντας μόνο τα H(L), H(IJ), H(MNOP), H(ABCDEFGH), χωρίς να αποκαλύψουμε το περιεχόμενο του K ή οποιαδήποτε άλλο τμήμα δεδομένων [10]. Για την απόδειξη επιπλέον της ακεραιότητας και κατοχής των αντίστοιχων δεδομένων [K], θα έπρεπε να τα αποκαλύψουμε ώστε ο κατακερματισμός τους να πραγματοποιηθεί κατά τη διαδικασία της επαλήθευσης.

Κεφάλαιο 4

Blockchain

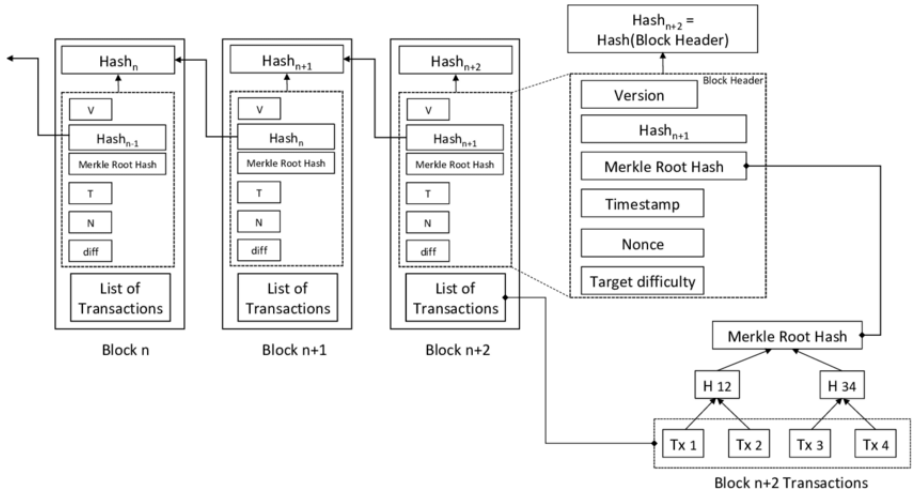
4.1 Τι είναι το blockchain

Το blockchain ορίζεται ως μια κατανεμημένη βάση δεδομένων ή δημόσιο κατάστιχο, όλων των συναλλαγών που εκτελέστηκαν ή και γενικότερα των δεδομένων που μοιράστηκαν μεταξύ των συμμετεχόντων μελών του συστήματος. Οι διάφορες καταχωρήσεις οργανώνονται σε ομάδες, μπλοκς, τα οποία με τη σειρά τους συνδέονται μεταξύ τους κατά μονοσήμαντο τρόπο με χρήση κρυπτογραφικών μεθόδων, σχηματίζοντας μια αλυσίδα. Ένα blockchain διατηρείται και ενημερώνεται από ένα δίκτυο ομότιμων κόμβων, δημόσιο ή ιδιωτικό, όπου όλοι οι κόμβοι διατηρούν ένα αντίγραφο της αλυσίδας. Κάθε πληροφορία προτού συμπεριληφθεί σε ένα καινούργιο μπλοκ, επαληθεύεται βάση ενός αλγορίθμου συναίνεσης που συνήθως απαιτεί την συμφωνία της πλειοψηφίας των μελών του συστήματος, και αφότου εισαχθεί στο blockchain καθίσταται αμετάβλητη [11].

Δηλαδή, το blockchain λειτουργεί ως ένα αποκεντρωμένο (decentralized), κατανεμημένο κατάστιχο, το οποίο είναι κοινό για όλους τους συμμετέχοντες, μιας και όλοι οι εμπλεκόμενοι αποθηκεύουν ένα αντίγραφό του, εξασφαλίζοντας την ασφάλεια και η διαφάνεια των συναλλαγών. Επιπλέον, η διαδικασία διεκπεραίωσης των συναλλαγών απαλλάσσεται από την ανάγκη ύπαρξης μιας ενδιάμεσης έμπιστης αρχής (όπως μιας τράπεζας), αφού η εμπιστοσύνη των συναλλασσόμενων μερών βασίζεται στην αλγοριθμική επιβεβαίωση. Επιπλέον, η τροποποίηση οποιασδήποτε προήγουμενης πληροφορίας καθίσταται αδύνατη, καθώς επηρεάζει όλες τις μεταγενέστερες καταχωρήσεις στη δομή.

Ο κρυπτογράφος David Chaum πρότεινε για πρώτη φορά ένα πρωτόκολλο που μοιάζει με το σημερινό blockchain στη διατριβή του το 1982 "Computer Systems Established, Maintained, and Trusted by Mutually Suspicious Groups" [12]. Περαιτέρω εργασία σε ένα κρυπτογραφικά ασφαλές blockchain έλαβε χώρα το 1991 από τους Stuart Haber και W. Scott Stornetta, σε προσπάθεια υλοποίησης ενός συστήματος όπου οι χρονικές σημάνσεις εγγράφων δεν θα μπορούσαν να παραβιαστούν [13]. Το 1992, οι Haber, Stornetta και Dave Bayer ενσωμάτωσαν Merkle trees στο σχέδιο του συστήματος, γεγονός που βελτίωσε την αποτελεσματικότητά του επιτρέποντας τη συλλογή πολλών καταχωρήσεων σε ένα μπλοκ [14].

Σημαντικό ορόσημο στην εξέλιξή του αποτέλεσε ο σχεδιασμός του πρώτου αποκεντρωμένου μοντέλου blockchain το 2008, από ένα άτομο (ή ομάδα ατόμων) γνωστό με το ψευδώνυμο Satoshi Nakamoto. Το σχέδιο αυτό υιοθετήθηκε τον επόμενο χρόνο από τον Nakamoto



Σχήμα 4.1: *Δομή του blockchain, και περιεχόμενο ενός μπλοκ*

ως βασικό συστατικό στην υλοποίηση του bitcoin, όπου λειτουργεί ως δημόσιο κατάστιχο όλων των συναλλαγών στο δίκτυο. Από τότε ακολούθησαν πολλές ακόμα υλοποιήσεις βασισμένες στην τεχνολογία του blockchain όπως το Ethereum και το Ripple.

4.2 Block

Οι διάφορες καταχωρήσεις στο blockchain οργανώνονται σε ομάδες που ονομάζονται μπλοκ. Ένα μπλοκ συνίσταται από την κεφαλίδα και το σώμα του. Το σώμα του μπλοκ περιέχει έναν μετρητή (transaction counter), καθώς και τις συναλλαγές. Η κεφαλίδα περιέχει έξι πεδία μεταβλητών:

- **Block Version**: Προσδιορίζει ποιο σύνολο κανόνων επαλήθευσης πρέπει να ακολουθείται.
- **Parent Block Hash**: Ένα πεδίο μεγέθους 256-bit που περιέχει το hash του προηγούμενου μπλοκ της αλυσίδα. Κατά τον τρόπο αυτό επιτυγχάνεται η μονοσήμαντη διασύνδεση των διαδοχικών μπλοκ, αλλά και η α-μεταβλητότητα των προηγούμενων συναλλαγών. Εξαίρεση αποτελεί το πρώτο μπλοκ της αλυσίδας (genesis block) που δε συνδέεται με κάποιο προηγούμενο.
- **Merkle Tree Root Hash**: Η ρίζα του Merkle tree των καταχωρήσεων που περιέχει το σώμα του μπλοκ.
- **Timestamp**: Η χρονική στιγμή που προστέθηκε το μπλοκ στην αλυσίδα, σε μορφή δευτερολέπτων που έχουν περάσει από την 1η Ιανουαρίου του 1970. Κατά τον τρόπο αυτό ορίζεται η χρονική αλληλουχία των καταχωρήσεων που συμπεριλαμβάνονται στο blockchain.
- **Nonce**: Ένα πεδίο που περιέχει έναν 32-bit αριθμό. Οι miners χρησιμοποιούν το πεδίο αυτό για την επίλυση ενός υπολογιστικού προβλήματος βάση του αλγορίθμου PoW.

- **Difficulty**: Περιέχει έναν αριθμό που καθορίζει πόσο δύσκολη θα είναι η εύρεση του hash για το PoW. Χρησιμοποιείται για την ρύθμιση του χρονικού διαστήματος εισαγωγής ενός καινούργιου μπλοκ στην αλυσίδα.

4.3 Ψηφιακές υπογραφές

Κάθε χρήστης του συστήματος διαθέτει ένα ζεύγος ιδιωτικού-δημοσίου κλειδιού. Το ιδιωτικό κλειδί πρέπει να διατηρείται μυστικό και χρησιμοποιείται για την υπογραφή των συναλλαγών. Οι ψηφιακά υπογεγραμμένες συναλλαγές εκπέμπονται σε όλο το δίκτυο προκειμένου να επαληθευτούν και να συμπεριληφθούν στο blockchain. Η τυπική διαδικασία ψηφιακής υπογραφής χωρίζεται σε δύο φάσεις: τη φάση υπογραφής, και τη φάση επαλήθευσης [15].

Για παράδειγμα, η Αλίκη επιθυμεί να στείλει ένα μήνυμα στον Μπομπ. Για το σκοπό αυτό, η Αλίκη κρυπτογραφεί τα δεδομένα με το ιδιωτικό της κλειδί και στέλνει στον Μπομπ το κρυπτογραφημένο αποτέλεσμα και το αρχικό μήνυμα. Στη συνέχεια ο Μπομπ μπορεί να επαληθεύσει με χρήση του δημοσίου κλειδιού της Αλίκης πως το μήνυμα δεν έχει αλλοιωθεί, και πως πράγματι ανήκει στην Αλίκη (integrity and authenticity).

Ο τυπικός αλγόριθμος ψηφιακής υπογραφής που χρησιμοποιείται στο blockchain είναι ο αλγόριθμος ψηφιακής υπογραφής ελλειπτικής καμπύλης (ECDSA) [16].

4.4 Κύρια χαρακτηριστικά του blockchain

4.4.1 Αποκέντρωση

Σε συμβατικά συγκεντρωτικά συστήματα συναλλαγών, κάθε συναλλαγή πρέπει να επικυρώνεται μέσω μιας κεντρική αξιόπιστης αρχής (π.χ. τράπεζα), με αποτέλεσμα να επιφέρουν επιπλέον κόστος, αλλά και περιορισμό της απόδοσης με τον κεντρικό διακομιστή να αποτελεί στενωπός του συστήματος. Αντιθέτως, το blockchain δεν απαιτεί την ύπαρξη μιας έμπιστης οντότητας, ενώ χρησιμοποιεί έναν αλγόριθμο ομοφωνίας (consensus) για τη διατήρηση της συνοχής των δεδομένων στο δίκτυο.

4.4.2 Μονιμότητα

Οι συναλλαγές που δημιουργούνται μπορούν να ελέγχονται πολύ γρήγορα. Στο ενδεχόμενο όπου μία συναλλαγή διαπιστωθεί μη έγκυρη τότε δε γίνεται αποδεκτή από έντιμους miners, οπότε και δε συμπεριλαμβάνεται στο blockchain. Είναι σχεδόν αδύνατον να διαγραφθεί ή να επαναφερθεί μια συναλλαγή που ήδη έχει συμπεριληφθεί στο blockchain. Οπότε στην περίπτωση που κάποιος μη έντιμος miner συμπεριλάβει στο μπλοκ που δημιουργεί μια μη έγκυρη συναλλαγή, τότε οι υπόλοιποι κόμβοι του δικτύου μπορούν να το ανιχνεύσουν άμεσα, καθώς δεν υπάρχει εμπιστοσύνη μεταξύ των κόμβων του δικτύου, με αποτέλεσμα να μην εντάσσουν το άκυρο μπλοκ στην αλυσίδα που διατηρούν, αποφεύγοντας τελικά την ανάγκη επιστροφής σε προηγούμενη κατάσταση του συστήματος, πράγμα που όπως είδαμε δεν επιτρέπει το πρωτόκολλο. Δηλαδή, όλες οι συναλλαγές που έχουν επικυρωθεί και συμπεριληφθεί στο blockchain θεωρούνται έγκυρες και δεν μπορούν να μεταβληθούν.

4.4.3 Ανωνυμία

Κάθε χρήστης μπορεί να αλληλεπιδράσει με το blockchain μέσω μιας διεύθυνσης που δημιουργεί, η οποία και τον αντιπροσωπεύει στις συναλλαγές που εκτελεί χωρίς την ανάγκη αποκάλυψης της πραγματικής του ταυτότητας. Ωστόσο, το blockchain δεν μπορεί από μόνο του να εγγυηθεί τελικά την απόλυτη ιδιωτικότητα των χρηστών λόγω εγγενών περιορισμών. Η χρήση της ίδιας διεύθυνσης σε πολλές συναλλαγές επιτρέπει τη μεταξύ τους συσχέτιση. Επιπλέον, όπως παρουσιάζεται στα [17, 18, 19], είναι εφικτή σε ορισμένες περιπτώσεις η συσχέτιση της διεύθυνση ενός χρήστη με κάποιο ψευδώνυμό του εώς και τα προσωπικά του στοιχεία ή την IP του. Για την αντιμετώπιση τέτοιων φαινομένων χρησιμοποιούνται τεχνικές όπως Mixing, ενώ στην περίπτωση του ZCash εκμεταλλεύονται τα zk-SNARKs προσφέροντας πλήρη ανωνυμία μέσω της ιδέας των αποδδείξεων μηδενικής γνώσης.

4.4.4 Διαφάνεια

Τα δεδομένα που συμπεριλαμβάνονται στο blockchcain είναι ορατά από όλους τους συμμετέχοντες στο σύστημα. Έτσι μπορεί οποιοσδήποτε επιθυμεί να ανακτήσει και να επαληθεύσει οποιαδήποτε συναλλαγή εκτελέστηκε στο παρελθόν.

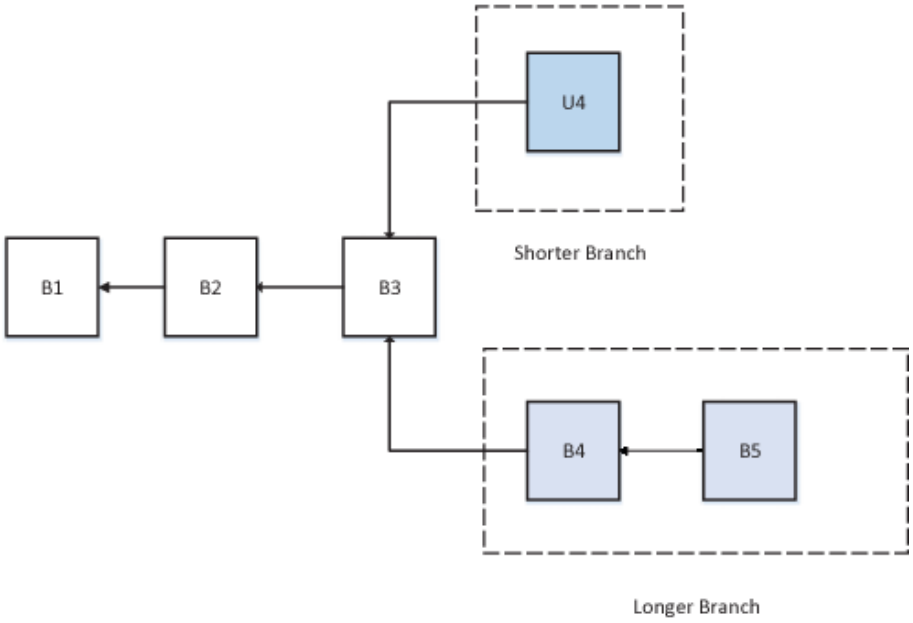
4.5 Ομοφωνία (Consensus)

Σε ένα κατανεμημένο σύστημα, η επίτευξη της ομοφωνίας μεταξύ όλων των κόμβων ισοδυναμεί με το κλασικό πρόβλημα των Βυζαντινών Στρατηγών [20]. Έχουν αναπτυχθεί πολλοί αλγόριθμοι για την επίλυση του προβλήματος, με τον πιο διαδεδομένο να είναι ο Proof of Work (PoW).

Το PoW αποτελεί μια στρατηγική επίτευξης ομοφωνίας που χρησιμοποιείται ευρέως σε τεχνολογίες όπως το Bitcoin και το Ethereum. Βάση της στρατηγικής αυτής, κάθε κόμβος τους συστήματος σχηματίζει το δικό του υποψήφιο καινούργιο μπλοκ και υπολογίζει το hash της κεφαλίδας του. Η κεφαλίδα όπως είδαμε στο 4.2 περιέχει ένα nonce το οποίο οι miners μεταβάλλουν για να εξερευνούν διαφορετικά hashes. Στόχος τους είναι να καταλήξουν σε ένα hash που να είναι μικρότερο από μια δεδομένη τιμή (difficulty). Μόλις ένας miner πετύχει την επίλυση αυτού του προβλήματος δημοσιοποιεί το μπλοκ που δημιούργησε με το συγκεκριμένο nonce, και όλοι οι κόμβοι πρέπει να συμφωνήσουν ομόφωνα για την ορθότητά του, προσθέτοντάς το στο δικό τους αντίγραφο του blockchain. Οι κόμβοι που υπολογίζουν τα hashes ονομάζονται miners και η διαδικασία αυτή mining.

Σε ένα αποκεντρωμένο σύστημα με πολλούς miners αναμένεται να παράγονται ταυτόχρονα καινούργια έγκυρα μπλοκ, με αποτέλεσμα να δημιουργούνται διακλαδώσεις, δηλαδή περισσότερα από ένα διαφορετικά έγκυρα αντίγραφα του blockchain (εικόνα 4.2). Όμως είναι σχεδόν απίθανο δύο ανταγωνιζόμενες διακλαδώσεις να συνεχίσουν να επεκτείνονται ταυτόχρονα, οπότε το PoW επιβάλει την υιοθέτηση της μεγαλύτερη έγκυρης αλυσίδας ως τη σωστή, καταφέρνοντας να επιλύσει τέτοιου είδους διαφορές.

Λόγω των υπολογιστικών απαιτήσεων του PoW, οι miners ξοδεύουν πολλούς πόρους στη διαδικασία αυτή. Για το λόγο αυτό προβλέπεται μία ανταμοιβή προς τον υποψήφιο που επιτυγχάνει να προσθέσει ένα μπλοκ στην αλυσίδα, η οποία διαφέρει ανάλογα με την



Σχήμα 4.2: *Σενάριο διακλάδωσης στο blochchain*

εφαρμογή (συνήθως αποτυπώνεται ως είσπραξη ενός ποσού του εν λόγω κρυπτονομίσματος). Έτσι εξασφαλίζεται το ένα κίνητρο για τους miners ώστε να ακολουθούν το πρωτόκολλο, συναινώντας στην ορθή λειτουργία του συστήματος [15].

4.6 Ethereum

4.6.1 Τι είναι το Ethereum

Το Ethereum είναι μια πλατφόρμα βασισμένη στο blockchain, που θεσπίζει ένα δίκτυο ομότιμων κόμβων, το οποίο υποστηρίζει την ασφαλή εκτέλεση και επαλήθευση κώδικα εφαρμογών, γνωστές ως smart contracts. Τα smart contracts επιτρέπουν την εκτέλεση συναλλαγών μεταξύ των συμμετεχόντων μελών χωρίς την ανάγκη ύπαρξης μιας έμπιστης αρχής, με την εκτέλεσή τους να καταναλώνει ether (ETH), το εγγενές κρυπτονόμισμα του συστήματος. Οι καταχωρήσεις συναλλαγών στο Ethereum είναι αμετάβλητες, επαληθεύσιμες και διανέμονται με ασφάλεια σε όλο το δίκτυο, δίνοντας σε όλους τους συμμετέχοντες πλήρη ιδιοκτησία και ορατότητα στα δεδομένα τους [21].

Το Ethereum ορίζει έναν ενιαίο, κανονικό υπολογιστή (Ethereum Virtual Machine ή EVM). Όλοι οι συμμετέχοντες κόμβοι του δικτύου διατηρούν ένα αντίγραφο της κατάστασης αυτού του υπολογιστή, στην οποία και πρέπει να συμφωνούν. Επιπλέον, οποιοσδήποτε συμμετέχων μπορεί να μεταδώσει ένα αίτημα για εκτέλεση κάποιου αυθαίρετου υπολογισμού από τον υπολογιστή αυτό. Κάθε φορά που μεταδίδεται ένα τέτοιο αίτημα, άλλοι συμμετέχοντες στο δίκτυο επαληθεύουν, επικυρώνουν και εκτελούν τον υπολογισμό. Αυτή η εκτέλεση προκαλεί μια αλλαγή στην κατάσταση του EVM, η οποία μεταδίδεται σε ολόκληρο το δίκτυο.

Τα αιτήματα για υπολογισμούς συνήθως χαρακτηρίζονται ως συναλλαγές. Όλες οι συναλλαγές καθώς και η παρούσα κατάσταση του EVM αποθηκεύονται στο blockchain, το οποίο με τη σειρά του αποθηκεύεται και συμφωνείται από όλους τους κόμβους [22]. Το

blockchain όπως ήδη αναφέραμε, διαθέτει κατάλληλους μηχανισμούς για την εξασφάλιση της ακεραιότητας και διαφάνειας των συναλλαγών, καθώς και την επίτευξη ομοφωνίας μεταξύ των συμμετεχόντων κόμβων (PoW).

4.6.2 Ether

Το ether είναι το κρυπτονόμισμα που χρησιμοποιεί το Ethereum. Το Ethereum επιτρέπει στους προγραμματιστές να δημιουργούν αποκεντρωμένες εφαρμογές (dapps) που μοιράζονται μια κοινή ¨δεξαμενή¨ υπολογιστικής ισχύος, η οποία όμως είναι πεπερασμένη. Επομένως, είναι αναγκαία η ύπαρξη ενός μηχανισμού που να προσδιορίζει ποιος θα χρησιμοποιήσει ανά πάσα στιγμή αυτούς τους πόρους. Διαφορετικά, ένα dapp θα μπορούσε να καταναλώνει όλους τους πόρους του δικτύου, γεγονός που θα υπονόμευε τη διαθεσιμότητα του συστήματος.

Το κρυπτονόμισμα ether υποστηρίζει έναν μηχανισμό τιμολόγησης για την υπολογιστική ισχύ του Ethereum. Όταν οι χρήστες θέλουν να πραγματοποιήσουν μια συναλλαγή, πρέπει να πληρώσουν ένα ποσό ether για να αναγνωριστεί η συναλλαγή τους στο blockchain. Αυτό το κόστος χρήσης είναι γνωστό ως gas fee και το ποσό χρέωσης εξαρτάται από το μέγεθος της υπολογιστικής ισχύος που απαιτείται για την εκτέλεση της συναλλαγής καθώς και τη γενικότερη ζήτηση υπολογιστικής ισχύος σε όλο το δίκτυο εκείνη τη χρονική στιγμή. Επομένως, ακόμα κι αν ένα κακόβουλο dapp υποβάλει έναν άπειρο βρόχο, η συναλλαγή τελικά θα εξαντλούσε το προσφερόμενο ether (που είναι πεπερασμένο) και θα τερματιζόταν, επιτρέποντας στο δίκτυο να επιστρέψει στην κανονική λειτουργία του.

4.6.3 Accounts

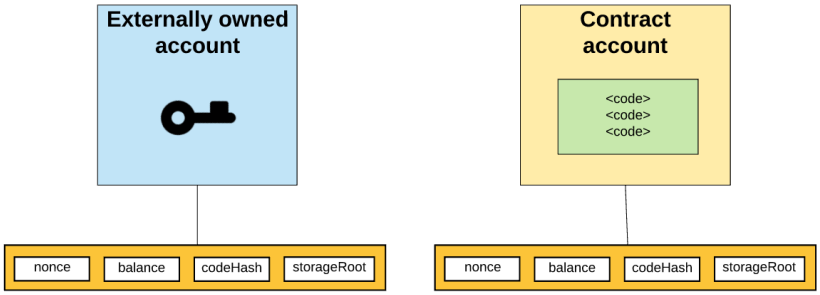
Ένας λογαριασμός στο Ethereum είναι μια οντότητα με ένα υπόλοιπο ether, που μπορεί να στείλει αιτήματα συναλλαγών στο σύστημα. Κάθε λογαριασμός έχει μία κατάσταση που το χαρακτηρίζει καθώς και μία 160-bit διεύθυνση. Η διεύθυνση αυτή αποτελεί ένα δημόσιο αναγνωριστικό που καθορίζει τον λογαριασμό κατά μοναδικό τρόπο. Υπάρχουν δύο ήδη λογαριασμών στο Ethereum:

- **Externally owned account (EOA)**: που ελέγχεται από έναν χρήστη μέσω ενός ιδιωτικού κλειδιού. Το κλειδί αυτό καθορίζεται κατά τη δημιουργία του λογαριασμού και διατηρείται κρυφό, ενώ με χρήση ECDSA παράγεται το δημόσιο κλειδί (4.3), που αποτελεί τη διεύθυνση του λογαριασμού. Ένα EOA μπορεί να εκκινήσει μια συναλλαγή στο σύστημα όποτε επιθυμεί, αν διαθέτει επαρκές ether για την κάλυψη του κόστους. Συναλλαγές μεταξύ EOA αφορούν αποκλειστικά μεταφορά ETH/token. Συνήθως χρησιμοποιούνται εφαρμογές όπως το Metamask ή το Ethereum wallet για τη διαχείριση των λογαριασμών και των συσχετιζόμενων κλειδιών τους, καθώς και για την αλληλεπίδραση με αυτά.

- **Contract account**: που ελέγχονται από τον κώδικα με τον οποίο σχετίζονται. Σε αντίθεση με τους EOA, δεν μπορούν να εκκινήσουν καινούργιες συναλλαγές από μόνοι τους. Μπορούν όμως να ενεργοποιήσουν συναλλαγές (internal transactions) ως απόκριση σε άλλες συναλλαγές που έχουν δεχτεί.

Τελικά οποιαδήποτε ενέργεια παρατηρείται στο Ethereum έχει ενεργοποιηθεί από συναλλαγές που πραγματοποιήθηκαν από κάποιο EOA. Τέλος, η κατάσταση κάθε λογαριασμού αποτελείται από τέσσερα πεδία:

- **nonce**: Για ένα EOA, αναπαριστά το πλήθος των συναλλαγών που έχει στείλει. Για ένα contract account, είναι το πλήθος τον συμβολαίων που έχουν δημιουργηθεί από τον εν λόγω λογαριασμό.
- **balance**: Το υπόλοιπο του λογαριασμό σε Wei. 1e+18 Wei ισοδυναμούν με ένα ETH.
- **StorageRoot**: Χαρακτηρίζεται και ως storage hash. Είναι ο κατακερματισμός τις ρίζας ενός Merkle Patricia tree. Το δέντρο αυτό κωδικοποιεί τον κατακερματισμό του περιεχομένου του αποθηκευτικού χώρου του λογαριασμού, και είναι κενό από προεπιλογή.
- **codeHash**: Στην περίπτωση ενός contract account, το πεδίο αυτό είναι το hash του EVM κώδικα του εν λόγω λογαριασμού. Στα EOA το πεδίο αυτό παραμένει κενό.



Σχήμα 4.3: *Είδη λογαριασμών στο Ethereum.*

4.6.4 Smart contracts

Ένα smart contract είναι απλώς ένα πρόγραμμα που εκτελείται στο Ethereum blockchain. Είναι μια συλλογή κώδικα (οι λειτουργίες του) και δεδομένων (η κατάστασή του) που βρίσκονται σε μια συγκεκριμένη διεύθυνση στο blockchain.

Τα smart contracts όπως είδαμε είναι ένα είδος λογαριασμού στο Ethereum, που σημαίνει ότι έχουν υπόλοιπο λογαριασμού και μπορούν να στέλνουν συναλλαγές στο δίκτυο. Επιπλέον, δεν ελέγχονται από κάποιον χρήστη, αλλά αφότου δημιουργηθούν εκτελούνται όπως έχουν προγραμματιστεί. Ένας χρήστης μπορεί να αλληλεπιδρά με ένα smart contract, υποβάλλοντας συναλλαγές που εκτελούν κάποια από τις προκαθορισμένες λειτουργίες του συμβολαίου. Τα smart contracts μπορούν να ορίζουν κανόνες, όπως ένα συνηθισμένο συμβόλαιο, και να τους επιβάλουν αυτόματα μέσω του κώδικά τους. Αξίζει να σημειωθεί ότι ένα smart contract αφότου δημιουργηθεί δεν μπορεί να διαγραφθεί, ενώ κάθε αλληλεπίδραση με αυτό είναι μη αναστρέψιμη.

4.6.5 Transactions

Οι συναλλαγές είναι κρυπτογραφικά υπογεγραμμένες οδηγίες από λογαριασμούς προς το EVM. Η απλούστερη περίπτωση συναλλαγής είναι η μεταφορά ether μεταξύ δύο λογαριασμών. Μία συναλλαγή που μεταβάλλει την κατάσταση του EVM πρέπει να μεταδοθεί σε ολόκληρο το δίκτυο. Οποιοσδήποτε κόμβος του συστήματος μπορεί να υποβάλει μια αίτηση εκτέλεσης μιας συναλλαγής, όμως πρέπει να επικυρωθεί και να εκτελεστεί από τους miners για να καταστεί έγκυρη. Μια υποβληθείσα συναλλαγή περιέχει τις εξής πληροφορίες:

- **recipient**: Η διεύθυνση του παραλήπτη. Μπορεί να προσδιορίζει ένα EOA σε μια συναλλαγή μεταφοράς ether, ή ένα λογαριασμό συμβολαίου για την εκτέλεση κάποιας λειτουργίας του.
- **signature**: Το αναγνωριστικό του αποστολέα. Δημιουργείται κατά την υπογραφή της συναλλαγής με το ιδιωτικό κλειδί του αποστολέα και επιβεβαιώνει την εξουσιοδότηση της διεκπεραίωσής της.
- **value**: Το ποσό ETH προς μεταφορά από τον αποστολέα προς τον παραλήπτη, σε wei.
- **data**: Προαιρετικό πεδίο, με χρήση στην εκτέλεση λειτουργιών ενός smart contract. Περιλαμβάνει πληροφορίες, όπως την συνάρτηση που πρόκειται να κληθεί καθώς και τα ορίσματα που χρειάζεται για την εκτέλεσή της. Το περιεχόμενο του καθορίζεται από το abstract binary interface (abi), που με τη σειρά του δημιουργείται κατά την μεταγλώττιση του κώδικα ενός smart contract και ορίζει τον τρόπο αλληλεπίδρασης με αυτό.
- **gasLimit**: Το μέγιστο ποσό gas, που μπορεί να καταναλωθεί από το σύνολο των απαιτούμενων υπολογιστικών βημάτων για την εκτέλεση της συναλλαγής.
- **maxPriorityFeePerGas**: Το μέγιστο ποσό σε gas που μπορεί να συμπεριληφθεί ως φιλοδώρημα προς τον miner.
- **maxFeePerGas**: Το μέγιστο ποσό gas, που ο αποστολέας διατίθεται να πληρώσει για την εκτέλεση της συναλλαγής.
- **nonce**: Σχετίζεται με το nonce που περιγράφηκε στο 4.6.3. Προκειμένου να αποφευχθούν επιθέσεις double spending, το Ethereum επιβάλει όλες οι συναλλαγές ενός EOA να αποστέλλονται και να εκτελούνται σειριακά. Το nonce λοιπόν, αποτελεί έναν μετρητή που ξεκινά από το μηδέν για έναν καινούργιο λογαριασμό, και μετά από κάθε συναλλαγή αυξάνεται κατά ένα. Επιβάλει έτσι όλες οι συναλλαγές να φτάνουν στο δίκτυο και να εκτελούνται με τη σειρά.

Σύμφωνα με τα παραπάνω, μια συναλλαγή μεταφοράς ether έχει την παρακάτω μορφή:

```
1.{
2.  from:  "0xEA674fdDe714fd979de3EdF0F56AA9716B898ec8",
3.  to:  "0xac03bb73b6a9e108530aff4df5077c2b3d481e5a",
4.  gasLimit:  "21000",
```

```
5.  maxFeePerGas:  "300",
6.  maxPriorityFeePerGas:  "10",
7.  nonce:  "0",
8.  value:  "10000000000"
10.}
```

Είναι πλέον φανερό ότι υπάρχουν διαφορετικά ήδη transaction, καθένα με τη δική του λειτουργία. Πιο συγκεκριμένα έχουμε τρία ήδη:

- **Συνήθης συναλλαγές**: μεταφοράς ether μεταξύ δύο διαφορετικών λογαριασμών.
- **Συναλλαγές δημιουργίας συμβολαίων**: όπου το πεδίο του παραλήπτη φέρει τη διεύθυνση του καινούργιου συμβολαίου, και το πεδίο δεδομένων τον κώδικα.
- **Εκτέλεσης λειτουργίας συμβολαίου**: που επιτρέπει την αλληλεπίδραση με ένα λογαριασμό συμβολαίου, και την εκτέλεσης του κώδικά του.

4.6.6 Gas and fees

Gas ονομάζεται η μονάδα μέτρηση της υπολογιστικής ισχύος που απαιτεί η εκτέλεση μια λειτουργία στο δίκτυο του Ethereum. Δεδομένου ότι η εκτέλεση μιας συναλλαγής καταναλώνει υπολογιστικούς πόρους, επιβάλλονται κάποια τέλη. Έτσι η επιτυχής διεκπεραίωση μιας συναλλαγής στο Ethereum συνοδεύεται από την πληρωμή κάποιου ποσού, ανάλογου των υπολογιστικών πόρων που καταναλώθηκαν.

Τα τέλη gas καταβάλλονται στο εγγενές νόμισμα του Ethereum, το ether (ETH). Οι τιμές του gas αναπαρίστανται σε gwei, το οποίο είναι μια υποδιαίρεση του ETH - κάθε gwei ισούται με 0,000000001 ETH (1e-9 ETH). Για παράδειγμα, αντί να πούμε ότι το gas κοστίζει 0,000000001 ether, χρησιμοποιούμε την αναπαράσταση του 1 gwei. Η λέξη ¨gwei¨ σημαίνει ¨giga-wei¨ και ισούται με 1.000.000.000 wei. Το Wei (που πήρε το όνομά του από τον Wei Dai, δημιουργό του b-money) αποτελεί τη μικρότερη μονάδα του ETH.

Για την εκτέλεση μιας συναλλαγής στο δίκτυο, οι χρήστες μπορούν να καθορίσουν ένα μέγιστο όριο που είναι διατεθειμένοι να πληρώσουν για να εκτελεστεί η συναλλαγή τους. Αυτή η προαιρετική παράμετρος είναι γνωστή ως maxFeePerGas. Για να εκτελεστεί μια συναλλαγή, το μέγιστο όριο χρέωσης πρέπει να υπερβαίνει το συνολικό, πραγματικό κόστος της συναλλαγής. Στον αποστολέα της συναλλαγής επιστρέφεται η διαφορά μεταξύ του μεγίστου ορίου χρέωσης και του αθροίσματος του βασικού τέλους εκτέλεσης και του φιλοδωρήματος (που εισπράττει ο miner).

Τέλος αξίζει να αναφερθεί ότι στο Ethereum, ορίζεται ένα όριο gas για τα μπλοκ. Δηλαδή, το συνολικό κόστος όλων των συναλλαγών που συμπεριλαμβάνει σε κάθε μπλοκ, δεν πρέπει να ξεπερνά το όριο αυτό. Επί του παρόντος, το όριο αυτό στο κύριο δίκτυο του Ethereum (mainnet) είναι 30e+10 gas.

Κεφάλαιο 5

zk-SNARK

5.1 Αποδείξεις Μηδενικής Γνώσης

Μια απόδειξη μηδενικής γνώσης γνώσης (Zero-Knowledge Proof ή ZKP) είναι ένα πρωτόκολλο που επιτρέπει σε μια οντότητα, τον αποδείκτη (prover), να πείσει με μεγάλη πιθανότητα μια άλλη υπολογιστική οντότητα, τον επαληθευτή (verifier), για την εγκυρότητα μιας πρότασης (assertion, statement) χωρίς να αποκαλύψει στον επαληθευτή τίποτα απολύτως πέραν της εγκυρότητας καθαυτής.

Εδώ λοιπόν η έννοια της απόδειξης διαφέρει από την έννοια του αντίστοιχου όρου στα μαθηματικά. Οι μαθηματικές αποδείξεις ενός μαθηματικού θεωρήματος ή μιας πρότασης είναι κάτι πολύ συγκεκριμένο και έχει μάλλον μία στατική δομή: είναι μία σειρά από μαθηματικά επιχειρήματα, τα οποία μπορούν να γραφούν με ξεκάθαρο και δομημένο τρόπο, ώστε ο αναγνώστης της απόδειξης να μπορεί να την κατανοήσει και να την εξηγήσει σε κάποιον άλλον. Στις αποδείξεις μηδενικής γνώσης, η έννοια της απόδειξης είναι δομικά διαφορετική: αναζητούμε επιχειρήματα για την εγκυρότητα μιας πρότασης, ή την αλήθεια ενός ισχυρισμού, τα οποία όμως δεν πρέπει να μεταφέρουν κανένα είδος γνώσης στον επαληθευτή. Μετά την ολοκλήρωση της αλληλεπίδρασης με τον αποδείκτη, ο επαληθευτής δε θα πρέπει να είναι σε θέση να πείσει κάποιον τρίτο για την εγκυρότητα του ισχυρισμού, με χρήση της γνώσης που έλαβε κατά τη συμμετοχή του στο πρωτόκολλο [23]. Ακολουθεί ένα κλασικό παράδειγμα απόδειξης μηδενικής γνώσης:

Έστω ότι έχουμε ένα φίλο με αχρωματοψία, και διαθέτουμε δύο μπάλες, μία κόκκινη και μία πράσινη, που είναι πανομοιότητες ως προς τα υπόλοιπα χαρακτηριστικά τους. Στον φίλο μας φαίνονται εντελώς όμοιες μεταξύ τους, και είναι δύσπιστος ως προς τον ισχυρισμό μας ότι τελικά είναι διακρίσιμες. Θα θέλαμε λοιπόν να του αποδείξουμε ότι είναι πράγματι διαφορετικές, χωρίς όμως να αποκαλύψουμε το χρώμα της καθεμίας. Ένα αποδεικτικό σύστημα που μπορεί να πετύχει το παραπάνω είναι το εξής:

1. Δίνουμε τις μπάλες στον φίλο μας, ο οποίος λαμβάνει το ρόλο του επαληθευτή. Κρύβει τις μπάλες πίσω από την πλάτη του, και επιλέγει να μας παρουσιάσει μία από τις δύο.

2. Έπειτα, ξανακρύβει τις μπάλες και με πιθανότητα 50% επιλέγει πάλι να μας παρουσιάσει μία από τις δύο.

3. Βλέποντας τα χρώματα μπορούμε να διακρίνουμε αν ο φίλος μας τις άλλαξε, ή παρουσίασε την ίδια. Σε περίπτωση που ισχυριζόμαστε ψευδώς ότι είναι διαφορετικού χρώματος, τότε θα απαντήσουμε σωστά με μόλις 50% πιθανότητα.

4. Επαναλαμβάνεται η διαδικασία των βημάτων 2 και 3 έως ότου ο φίλος μας πειστεί για την ορθότητα του ισχυρισμού μας.

Επαναλαμβάνοντας τη διαδικασία αρκετές φορές, αν ισχυρισμός μας είναι ψευδής, η πιθανότητα να απαντήσουμε σωστά κατά τυχαίο τρόπο κάθε φορά είναι τελικά αμελητέα. Αν και ο φίλος μας πείθεται πως οι μπάλες είναι διαφορετικές, τελικά δεν αποκτά καμία γνώση που να του επιτρέπει να τις ξεχωρίσει [24].

Το πρωτόκολλο του παραδείγματος βασίζεται σε ένα διαλογικό (interactive) αποδεικτικό σύστημα, όπου ο επαληθευτής επιλέγει ένα πλήθος από τυχαίες προκλήσεις (challenges), τις οποίες ο αποδείκτης πρέπει να απαντήσει. Υπάρχουν όμως και μη διαλογικά πρωτόκολλα αποδείξεων μηδενικής γνώσης. Συνοψίζοντας, μια απόδειξη μηδενικής γνώσεις συνίσταται στην περιγραφή ενός πρωτοκόλλου επικοινωνίας μεταξύ δύο οντοτήτων, του αποδείκτη και του επαληθευτή (prover/verifier). Τα χαρακτηριστικά που συνθέτουν μια απόδειξη μηδενικής γνώσης είναι:

- **Πληρότητα**: ο αποδείκτης μπορεί να πείσει τον επαληθευτή με υψηλή πιθανότητα, όταν πρόκειται για αληθή πρόταση.

- **Ορθότητα**: ο επαληθευτής θα απορρίψει τις λανθασμένες προτάσεις με υψηλή πιθανότητα.

- **Μηδενική γνώση**: Αυτά που θα μάθει ο επαληθευτής μετά την επικοινωνία του με τον αποδείκτη, θα μπορούσε να τα είχε υπολογίσει και μόνος του χωρίς να συμμετάσχει στο πρωτόκολλο.

5.2 Τι είναι τα zk-SNARKs

Το ακρωνύμιο zk-SNARK σημαίνει “Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge”, και αναφέρεται σε μια μέθοδος κατασκευής αποδείξεων, βασισμένη σε κρυπτογραφικές μεθόδους, που επιτρέπει σε κάποιον (αποδείκτης) να αποδείξει την κατοχή μιας συγκεκριμένης πληροφορίας (π.χ. password, hash preimage), χωρίς να αποκαλύψει την πληροφορία αυτή [25]. Επιπλέον, τα zk-SNARKs είναι μη-διαλογικά πρωτόκολλα, δηλαδή δεν απαιτείται διάλογος μεταξύ αποδείκτη και επαληθευτή υπό τη μορφή ανταλλαγής μηνυμάτων για πολλούς γύρους, διαχωρίζοντας τα από τα διαλογικά αποδεικτικά συστήματα.

Όπως υποδηλώνει και το όνομα, τα zk-SNARKs έχουν τα εξής χαρακτηριστικά:

- **Zero-Knowledge (Μηδενική Γνώση)**: Ο επαληθευτής δεν μπορεί να εξάγει καμία γνώση από την απόδειξη, παρά μόνο την εγκυρότητά της (αντίστοιχα με τις αποδείξεις μηδενικής γνώσης 5.1).

- **Succinct (Σύντομο)**: Οι αποδείξεις είναι μικρές και μπορούν να επαληθευτούν πολύ γρήγορα και με ευκολία, ανεξάρτητα από την πολυπλοκότητα και το μέγεθος των θέσεων και τον προγραμμάτων που αντιπροσωπεύουν.

- **Non-Interactive (Μη-διαλογικό)**: Σε αντίθεση με τα διαλογικά πρωτόκολλα, δεν απαιτείται η αμφίδρομη επικοινωνία μεταξύ αποδείκτη και επαληθευτή. Η απόδειξη αποτελείται από μόλις ένα μήνυμα που ο αποδείκτης κατασκευάζει και στέλνει στον επαληθευτή. Στη συνέχεια ο επαληθευτής, διατέτοντας την απόδειξη, είναι σε θέση να αποφανθεί για την ορθότητά της, χωρίς την ανάγκη περαιτέρω αλληλεπίδρασης με τον αποδείκτη.

- **ARgument (Επιχείρημα)**: Ο χαρακτηρισμός αυτός αποτελεί απλό φορμαλισμό. Τα zk-SNARKs βασίζονται σε κρυπτογραφικές μεθόδους και μη-ντετερμινισμό, με αποτέλεσμα να μην αποτελούν επίσημες αποδείξεις με τη μαθηματική έννοια του όρου (αν και η πιθανότητα ο αποδείκτης να πείσει για κάτι που δε γνωρίζει είναι πρακτικά αμελητέα).

- **Knowledge (Γνώση)**: Μια απόδειξη αντιπροσωπεύει τόσο την ορθότητα ενός ισχυρισμού, όσο και την κατοχή των αποδεικτικών στοιχείων από τον αποδείκτη [26].

Μια αρκετά αφαιρετική περιγραφή ενός πρωτοκόλλου zk-SNARK, συγκροτείται από τρεις αλγόριθμους G, P, V, οι οποίοι ορίζονται ως εξής:

- $G(lambda, C) = (pk, vk)$
  Μια γεννήτρια που με ορίσματα μια κρυφή τυχαία παράμετρο lambda, και ένα πρόγραμμα C, παράγει ένα ζεύγος κλειδιών pk, vk (proving key, verification key). Τα κλειδιά αυτά συνήθως χαρακτηρίζονται ως CRS (Common Reference String), και είναι δημόσια και επαναχρησιμοποιήσημα. Η μυστική παράμετρος (γνωστή και ως toxic waste) είναι απαραίτητο να καταστραφεί μετά την παραγωγή των κλειδιών, καθώς η κατοχή της μπορεί να επιτρέψει σε κάποιον να παράγει ψευδής αποδείξεις.

- $P(pk, x, w) = prf$
  Η συνάρτηση που εκτελεί ο αποδείκτης. Για την κατασκευή μιας απόδειξης απαιτούνται το proving key, μια δημόσια είσοδος x, και η ιδιωτική γνώση του αποδείκτη. Η ιδιωτική γνώση συνήθως χαρακτηρίζεται και ως μάρτυρας (witness).

- $V(vk, x, prf)$
  Γνωρίζοντας πλέον ο επαληθευτής την απόδειξη που κατασκεύασε ο αποδείκτης, μπορεί να την ελέγξει χρησιμοποιώντας την ίδια δημόσια είσοδο x και το verification key. Αν ο έλεγχος είναι επιτυχής, τότε επαληθεύεται ότι ο αποδείκτης γνωρίζει ένα μάρτυρα, που ικανοποιεί τον εν λόγω ισχυρισμό. Δηλαδή ο αποδείκτης γνωρίζει ένα w, τέτοιο ώστε $C(x, w) == true$ [27].

5.3 Διαδικασίας κατασκευής zk-SNARK

Τα zk-SNARKs δεν μπορούν να εφαρμοστούν απευθείας σε έναν αυθαίρετο υπολογισμό. Το προς απόδειξη πρόβλημα, πρέπει πρώτα να μετασχηματιστεί σε ένα ισοδύναμο Quadratic Arithmetic Program (QAP). Στη συνέχεια μπορεί να εφαρμοστεί με συστηματικό τρόπο μια ξεχωριστή διαδικασία για την κατασκευή των αποδείξεων μηδενικής γνώσης, η οποία αξιοποιεί ιδιότητες πολυωνύμων. Έτσι η κατασκευή των αποδείξεων διακρίνεται σε δύο κύρια

στάδια (καθώς και ορισμένα ενδιάμεσα στάδια), που θα παρουσιαστούν συνοπτικά στην ενότητα αυτή.

Computation → Arithmetic Circuit → R1CS → QAP → zk-SNARK

5.3.1 Μετατροπή προγράμματος σε QAP

Τα zk-SNARKs υποστηρίζουν την απόδειξη αυθαίρετα μεγάλων και πολύπλοκων υπολογισμών. Αν και η απευθείας κατασκευή ενός αριθμητικού κυκλώματος είναι εφικτή, σε πιο σύνθετα προγράμματα κάτι τέτοιο μπορεί να είναι αρκετά δύσκολο. Για το σκοπό αυτό χρησιμοποιείται μια ειδικά σχεδιασμένη γλώσσα προγραμματισμού, που υποστηρίζει βασικές αριθμητικές πράξεις $(+, -, \cdot, /)$ και αναθέσεις μεταβλητών, καθώς και οποιαδήποτε άλλη λειτουργία μπορεί να εκφραστεί βάση αυτών.

Για παράδειγμα, θεωρούμε ότι γνωρίζουμε τη λύση μιας τριτοβάθμιας εξίσωσης.

$$x^3 + x + 5 = 35$$

Η αντίστοιχη αναπαράσταση σε πρόγραμμα θα είχε την εξής μορφή:

```
def qeval(x):
    y = x * x * x
    return x + y + 5
```

Από πρόγραμμα σε αριθμητικό κύκλωμα

Για την αναπαράσταση ενός προγράμματος σε ισοδύναμο σύστημα R1CS, προηγείται flattening του κώδικα, σε μια αλληλουχία απλών statements της μορφής:

- x = y
- x = y (op) z, όπου (op) ένας αριθμητικός τελεστής

Δηλαδή η ισοδύναμη αναπαράσταση του παραδείγματος θα είναι:

```
sym_1 = x * x
y = sym_1 * x
sym_2 = y + x
~out = sym_2 + 5
```

Από αριθμητικό κύκλωμα σε R1CS

Ένα R1CS (Rank 1 Constraint System) ορίζεται ως μια αλληλουχία από ομάδες τριών διανυσμάτων (a, b, c), ενώ η λύση ενός τέτοιου συστήματος είναι ένα διάνυσμα s που ικανοποιεί την ισότητα $s.a \cdot s.b - s.c = 0$ (ο τελεστής ¨.¨ ορίζει το εσωτερικό γινόμενο μεταξύ

δύο διανυσμάτων). Κάθε ομάδα διανυσμάτων (a, b, c) αναπαριστά τον γραμμικό συνδυασμό ή/και τον πολλαπλασιασμό ορισμένων στοιχείων του διανύσματος λύσης, ορίζοντας τελικά έναν περιορισμό.

Για τη μετατροπή του αριθμητικού κυκλώματος σε R1CS, ορίζεται ένα διάνυσμα αντιστοίχησης (mapping vector) που περιέχει μία μεταβλητή ~one για την αναπαράσταση της μονάδας στην πρώτη θέση του (χρησιμοποιείται στην εφαρμογή των zk-SNARKs αναπαριστώντας τον σταθερό όρο των πολυωνύμων), όλες τις μεταβλητές-εισόδους του προγράμματος, την έξοδο του προγράμματος ~out, καθώς και όλες τις ενδιάμεσες μεταβλητές που εμφανίζονται (y, sym_1, sym_2). Το διάνυσμα-λύση του R1CS θα είναι η κατάλληλη ανάθεση όλων μεταβλητών που ικανοποιούν το πρόβλημα, με την ίδια ακριβώς σειρά. Στη συνέχεια, κάθε statement του flattened κώδικα μπορεί να αναπαρασταθεί με συστηματικό τρόπο, ως μία ομάδα διανυσμάτων (a, b, c) του R1CS, όπου όλα τα διανύσματα θα έχουν το ίδιο μέγεθος με το διάνυσμα αντιστοίχησης, και θα εκφράζουν τη σχέση μεταξύ των μεταβλητών που εμφανίζονται στο αριστερό και δεξί μέλος του statement (είτε ως γραμμικό συνδυασμό, είτε ως πολλαπλασιασμό τους).

Επιστρέφοντας στο παράδειγμα, το διάνυσμα αντιστοίχησης του προβλήματος έχει τη μορφή:

$$mapping\_vector = [\sim one, x, \sim out, sym\_1, y, sym\_2]$$

Παρατηρώντας το πρώτο statement του κώδικα μπορούμε εύκολα να εξάγουμε την κατάλληλη αναπαράστασή του στο R1CS:

$$a = [0,\ 1,\ 0,\ 0,\ 0,\ 0]$$
$$b = [0,\ 1,\ 0,\ 0,\ 0,\ 0]$$
$$c = [0,\ 0,\ 0,\ 1,\ 0,\ 0]$$

Η κατάλληλη ανάθεση του διανύσματος-λύσης είναι προφανές ότι ικανοποιεί τη σχέση:

$$a.s \cdot b.s - c.s = 0, \text{ όπου } s = [1, 3, 35, 9, 27, 30]$$

Επαναλαμβάνοντας την ίδια διαδικασία για όλα τα statements καταλήγουμε στην επιθυμητή μορφή R1CS.

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 5 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, B = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, C = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Προκειμένου να επαληθευτεί ότι ένα διάνυσμα s αποτελεί λύση του R1CS, πρέπει να ισχύει η σχέση $a.s \cdot b.s - c.s = 0$ για καθεμία από τις ομάδες διανυσμάτων.

Από R1CS σε QAP

Η αναπαράσταση σε QAP, υλοποιεί την ίδια ακριβώς λογική ως προς τους περιορισμούς του συστήματος, χρησιμοποιώντας πολυώνυμα. Στο R1CS έχουμε τρεις ομάδες από k

διανύσματα μήκους l (όπου k αντιστοιχεί στο πλήθος των περιορισμών που ορίζονται, ενώ l το μέγεθος του διανύσματος λύσης). Η ισοδύναμη μορφή του QAP θα αποτελείται λοιπόν από τρεις ομάδες l πολυωνύμων, βαθμού k-1, όπου η αποτίμηση των πολυωνύμων για κάθε $x \in [1, k]$ θα αναπαριστά έναν από τους περιορισμούς του R1CS. Τελικά κάθε στήλη του R1CS αντικαθίσταται από ένα πολυώνυμο.

Έστω μια στήλη του R1CS με στοιχεία $a_1, a_2, ..., a_k$, τότε ορίζονται k σημεία της μορφής $(1, a_1), (2, a_2), ..., (k, a_k)$ και στη συνέχεια υπολογίζεται το πολυώνυμο βαθμού k-1 που περνά από αυτά τα σημεία (το οποίο είναι και μοναδικό). Για το σκοπό αυτό μπορεί να χρησιμοποιηθεί πολυωνυμική παρεμβολή Lagrange, όμως σε πραγματικές εφαρμογές όπου τα σημεία είναι πολύ περισσότερα από του παραδείγματος που παρουσιάζουμε, εφαρμόζονται διαφορετικοί αλγόριθμοι που χρησιμοποιούν Fast Fourier transform για τον προσδιορισμό του παρεμβαλόμενου πολυωνύμου με σημαντικά μικρότερη υπολογιστική πολυπλοκότητα.

Εξετάζοντας την πρώτη στήλη στο R1CS του παραδείγματος ορίζουμε τα σημεία (1, 0), (2, 0), (3, 0), (4, 5). Με χρήση του αλγορίθμου πολυωνυμικής παρεμβολής Lagrange, διαπιστώνουμε ότι το πολυώνυμο τρίτου βαθμού που περνά από τα σημεία αυτά είναι το $0.833x^3 - 5x^2 + 9.166x - 5$. Επαναλαμβάνοντας την ίδια διαδικασία σε όλες τις στήλες καταλήγουμε στο παρακάτω σύστημα (οι συντελεστές παρουσιάζονται σε αύξουσα σειρά):

$$A(x) = \begin{bmatrix} -5.0 & 9.166 & -5.0 & 0.833 \\ 8.0 & -11.333 & 5.0 & -0.666 \\ 0 & 0 & 0 & 0 \\ -6.0 & 9.5 & -4.0 & 0.5 \\ 4.0 & -7.0 & 3.5 & -0.5 \\ -1.0 & 1.833 & -1.0 & 0.166 \end{bmatrix}, B(x) = \begin{bmatrix} 3.0 & -5.166 & 2.5 & -0.333 \\ -2.0 & 5.166 & -2.5 & 0.333 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

$$C(x) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -1.0 & 1.833 & -1.0 & 0.166 \\ 4.0 & -4.333 & 1.5 & -0.166 \\ -6.0 & 9.5 & -4.0 & 0.5 \\ 4.0 & -7.0 & 3.5 & -0.5 \end{bmatrix}$$

Λόγω της μεθόδου κατασκευής των πολυωνύμων, αν αναθέσουμε στο παραπάνω σύστημα όπου $x = m$, για κάποιο $m \in [1, k]$, θα λάβουμε την ομάδα των τριών διανυσμάτων του m-οστού περιορισμού που ορίσαμε στο R1CS. Έτσι, μπορούμε να επαληθεύσουμε ότι ένα διάνυσμα s αποτελεί λύση του προβλήματος, υπολογίζοντας το $p(x) = s.A(x) \cdot s.B(x) - s.C(x)$ και επαληθεύοντας ότι μηδενίζεται για κάθε $x \in [1, k]$ (στα υπόλοιπα σημεία το πολυώνυμο $p(x)$ μπορεί να έχει οποιαδήποτε συμπεριφορά), δηλαδή σε κάθε σημείο που αντιστοιχεί σε πύλη του κυκλώματος.

Έλεγχος του QAP

Η μετατροπή του προβλήματος στο ισοδύναμο QAP επιτρέπει την ταυτόχρονη επαλήθευση όλων των περιορισμών που ορίστηκαν στο R1CS. Για το σκοπό αυτό, αντί να αποτιμήσουμε

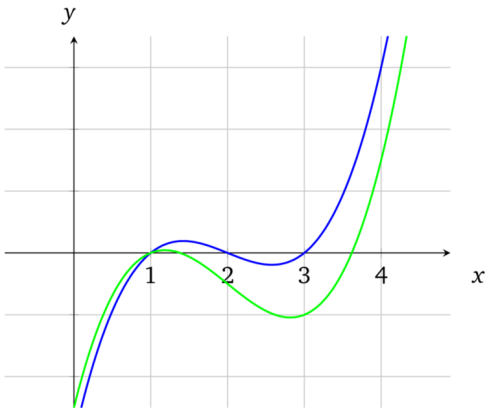
το $p(x)$ για όλα τα σημεία που αντιστοιχούν σε πύλες του κυκλώματος, το διαιρούμε με ένα άλλο πολυώνυμο $t(x)$, και ελέγχουμε πως η διαίρεση δεν αφήνει υπόλοιπο. Το $t(x)$ είναι το απλούστερο πολυώνυμο που μηδενίζεται στα σημεία που αντιστοιχούν σε λογικές πύλες, δηλαδή έχει τη μορφή $t(x) = (x-1)(x-2)...(x-k)$, με το $p(x)$ να πρέπει να είναι πολλαπλάσιο του $t(x)$ [28].

Συνοψίζοντας τα παραπάνω, ξεκινώντας από ένα υπολογιστικό πρόγραμμα μετασχηματίζουμε το πρόβλημα σε ισοδύναμη μορφή QAP, αναπαριστώντας κάθε υπολογιστικό βήμα του προγράμματος ως έναν περιορισμό, και υλοποιώντας τον με χρήση πολυωνύμων. Έτσι η απόδειξη του υπολογισμού του προγράμματος για κάποιες εισόδους που γνωρίζει ο αποδείκτης, ισοδυναμεί πλέον με τη γνώση του πολυωνύμου $p(x) = s.A(x) \cdot s.B(x) - s.C(x)$, που με τη σειρά του προϋποθέτει τη γνώση του διανύσματος λύσης s. Σε πραγματικές εφαρμογές, οι υπολογισμοί πραγματοποιούνται με αριθμητική υπολοίπου σε ένα πεπερασμένο σώμα ακεραίων τάξης n $(Z_n, +, \cdot)$, όπου n κάποιος μεγάλος πρώτος αριθμός.

5.3.2 Απόδειξη γνώσης πολυωνύμου

Πολυώνυμο ως αποδεικτικό μέσο

Από το θεμελιώδες θεώρημα της άλγεβρας, έχουμε ως άμεση απόρροια πως κάθε πολυώνυμο βαθμού d, έχει το πολύ d πραγματικές ρίζες. Θεωρώντας δύο μη ίσα πολυώνυμα f, g το πολύ βαθμού d, τότε αυτά θα έχουν το πολύ d κοινά σημεία, αφού το πολυώνυμο f-g είναι το πολύ βαθμού d. Για παράδειγμα, έστω τα πολυώνυμα $f(x) = x^3 - 6x^2 + 11x - 6$ και $g(x) = x^3 - 6x^2 + 10x - 5$, τα οποία παρουσιάζονται στην εικόνα 5.1.



Σχήμα 5.1: *Γραφικές παραστάσεις δύο μη-ίσων πολυωνύμων*

Είναι φανερό ότι μια τόσο μικρή διαφορά στους συντελεστές τους επιφέρει πολύ διαφορετικά αποτελέσματα. Ακόμη περισσότερο, είναι αδύνατον να βρεθούν δύο μη ίσα πολυώνυμα που μοιράζονται ένα συνεχόμενο κοινό τμήμα, εκτός από την τετριμμένη περίπτωση του μοναδικού σημείου.Πιο συγκεκριμένα αν αποτιμηθούν δύο μη ίσα πολυώνυμα το πολύ βαθμού d σε ένα τυχαίο $x \in [1, k]$, τότε η πιθανότητα τα δύο σημεία να ταυτίζονται είναι το πολύ d/k, που για επαρκώς μεγάλο πεδίο ορισμού, η πιθανότητα αυτή είναι αμελητέα.

Χωρίς βλάβη της γενικότητας, συμπεραίνουμε ότι η αποτίμηση ενός πολυωνύμου σε ένα αυθαίρετο σημείο, σχετίζεται άμεσα με την αναπαράσταση της μοναδικής του ταυτότητας. Σε αυτόν ακριβώς τον ισχυρισμό βασίζεται η κατασκευή και η ορθότητα των zk-SNARKs.

Παραγοντοποίηση πολυωνύμου

Βάση του θεμελιώδους θεωρήματος της άλγεβρας, κάθε επιλύσιμο πολυώνυμο μπορεί να αναλυθεί σε γινόμενο γραμμικών παραγόντων. Έτσι μπορούμε να αναπαραστήσουμε ένα έγκυρο πολυώνυμο ως το γινόμενο των παραγόντων του:

$$(x - a_1)(x - a_2)...(x - a_n) = 0$$

Έστω ότι ο αποδείκτης γνωρίζει ένα πολυώνυμο $p(x)$ που έχει τουλάχιστον τις ρίζες $a_1, ..., a_k$ (πιθανώς να έχει κι άλλες). Προκειμένου να αποδείξει τον ισχυρισμό του χωρίς να αποκαλύψει το $p(x)$, αρκεί να δείξει πως το $p(x)$ είναι το γινόμενο του βασικού πολυωνύμου $t(x) = (x - a_1)...(x - a_k)$, που έχει τις ρίζες αυτές, και ενός αυθαίρετου πολυωνύμου $h(x)$, τέτοιο ώστε:

$$p(x) = t(x)h(x)$$

Δεδομένου ότι το $p(x)$ περιέχει ως παράγοντα το $t(x)$, πρέπει να έχει και όλες του τις ρίζες. Ο υπολογισμός του κατάλληλου $h(x)$ μπορεί να γίνει μέσω της διαίρεσης $h(x) = p(x)/t(x)$. Αν ο αποδείκτης δεν μπορεί να βρει κατάλληλο $h(x)$, αυτό σημαίνει πως το πολυώνυμο που διαθέτει, δεν έχει ως παράγοντα το $t(x)$, και η διαίρεση τους αφήνει υπόλοιπο. Έτσι μπορεί να χρησιμποιηθεί το εξής απλό πρωτόκολλο για τη σύγκριση των πολυωνύμων $p(x)$ και $t(x)h(x)$ (όπου μόνο το $t(x)$ είναι γνωστό στον επαληθευτή):

1. Ο επαληθευτής επιλέγει έναν τυχαίο αριθμό r, και υπολογίζει το $t(r)$. Στη συνέχεια αποκαλύπτει το r στον αποδείκτη.

2. Ο αποδείκτης υπολογίζει το $h(x) = p(x)/t(x)$. Στη συνέχεια αποτιμά τα $p(r)$ και $h(r)$ τα οποία και στέλνει στον επαληθευτή.

3. Ο επαληθευτής ελέγχει αν $p = th$. Έτσι μπορεί να διαπιστώσει αν το $p(x)$ έχει τις επιθυμητές ρίζες.

Στο πρωτόκολλο αυτό, ο αποδείκτης έχει τη δυνατότητα να επινοήσει έναν τυχαίο h και να θέσει $p = th$, κάτι που θα γίνει δεκτό από τον επαληθευτή. Επιπλέον, δεδομένου ότι μαθαίνει το τυχαίο r, μπορεί να κατασκευάσει οποιοδήποτε πολυώνυμο που να έχει απλώς ένα κοινό σημείο στο r με το $t(r)h(r)$. Αυτά αποτελούν ζητήματα που επιλύονται με χρήση των μαθηματικών εργαλείων που παρουσιάζονται στη συνέχεια.

Ομομορφική κρυπτογράφηση

Η ομορφοφική κρυπτογράφηση είναι μια μορφή κρυπτογράφησης που επιτρέπει την εκτέλεση αριθμητικών υπολογισμών στα κρυπτογραφημένα δεδομένα, χωρίς την ανάγκη αποκρυπτογράφησής τους. Αυτή η μέθοδος μπορεί να χρησιμοποιηθεί για την απόκρυψη των τιμών που ανταλλάσσονται μεταξύ αποδείκτη και επαληθευτή στο πρωτόκολλο που αναφέραμε, επιτρέποντας όμως την εκτέλεση των αριθμητικών υπολογισμών που απαιτούνται. Για το σκοπό αυτό επιλέγονται και γνωστοποιούνται το μέγεθος n του πεδίο τιμών (δηλαδή το Z_n, όπου n ένας επαρκώς μεγάλος πρώτος αριθμός), καθώς και ένα στοιχείο γεννήτορας g. Έτσι για έναν αυθαίρετο αριθμό x μπορεί να υπολογιστή η κρυπτογράφησή του $E(x) = g^x(mod n)$.

Η ασφάλεια αυτού του σχήματος κρυπτογράφησης βασίζεται στη δυσκολία εύρεσης του x όταν μας είναι γνωστά τα g, n και E(x), γνωστό και ως το πρόβλημα διακριτού λογαρίθμου (DLP).

Χρησιμοποιώντας αυτό το σχήμα κρυπτογράφησης, μπορεί ο επαληθευτής να επιλέγει ένα τυχαίο r, το οποίο και κρατάει κρυφό, και στη συνέχεια να υπολογίζει και να στέλνει στον αποδείκτη όλα τα $E(r^i), \forall i \in [0, d]$, όπου d ο βαθμός του προς απόδειξη πολυωνύμου. Ο αποδείκτης είναι σε θέση να αποτιμήσει το πολυώνυμο που διαθέτει εφαρμόζοντας τους συντελεστές του $c_0, ..., c_d$ στις κρυπτογραφημένες τιμές, χωρίς να μάθει τελικά σε πιο σημείο αντιστοιχεί.

$$E(r^d)^{c_d} \cdot ... \cdot E(r^0)^{c_0} = g^{c_d r^d} \cdot ... \cdot g^{c_0 r^0} = g^{c_d r^d + ... + c_0 r^0} = E(c_d r^d + ... + c_0 r^0)$$

Στην πραγματικότητα, τα zk-SNARKs χρησιμοποιούν κρυπτογραφία ελλειπτικών καμπύλων για την κρυπτογράφηση των τιμών, ορίζοντας τις αντίστοιχες λειτουργίες που αναφέραμε.

Επαληθεύσιμη ύψωση σε δύναμη

Η γνώση του πολυωνύμου στα zk-SNARKs, είναι ισοδύναμη με τη γνώση των συντελεστών των επιμέρους όρων του. Όμως μέχρι στιγμής ο αποδείκτης μπορεί να επινοήσει δύο αυθαίρετες τιμές z_p, z_h που ικανοποιούν τη σχέση $z_p = (z_h)^{t(r)}$, ξεγελώντας τον επαληθευτή. Προκειμένου να επιβάλλεται στον αποδείκτη να χρησιμοποιεί τις κρυπτογραφημένες τιμές που του παρέχονται και να κατασκευάζει την απόδειξη με χρήση των συντελεστών του πολυωνύμου που διαθέτει, χρησιμοποιείται το Knowledge-of-Exponent Assumption (KEA) που προτάθηκε στο [29].

Έστω ότι ο επαληθευτής διαθέτει μια αυθαίρετη τιμή a και επιθυμεί να την υψώσει ο αποδείκτης σε μια δύναμη. Βάση του KEA μπορεί να χρησιμοποιηθεί το παρακάτω πρωτόκολλο:

1. Ο επαληθευτής επιλέγει ένα τυχαίο x, υπολογίζει το $a' = a^x (mod n)$ και παρέχει το ζεύγος (a, a') στον αποδείκτη.

2. Ο αποδείκτης επιλέγει έναν αριθμό y και κατασκευάζει τα $b = a^y (mod n)$ και $b' = a'^y (mod n)$. Στη συνέχεια αποκαλύπτει το ζεύγος (b, b').

3. Διαθέτοντας πλέον την απάντηση, ο επαληθευτής ελέγχει πως ισχύει η ισότητα $b^x = b'$.

Σύμφωνα με το KEA, το ζεύγος (b, b') μπορεί να προήλθε μόνο μέσω ύψωσης του (a, a') σε κάποια δύναμη, ενώ ο αποδείκτης δεν μπορεί να εξαπατήσει τον επαληθευτή αφού είναι αδύνατον να εξάγει το x από το a' (DLP). Αντίστοιχα, ο επαληθευτής μπορεί να ελέγχει πως οι τιμές που κατασκευάζει ο αποδείκτης, προέρχονται από την ύψωση των κρυπτογραφημένων τιμών σε μια δύναμη, χωρίς να μαθαίνει ποια είναι αυτή η δύναμη.

Πολλαπλασιασμός κρυπτογραφημένων τιμών

Ένα cryptographic pairing (billinear mapping) συμβολίζεται ως μια συνάρτηση $e(g^*, g^*)$, που για ορίσματα δύο κρυπτογραφημένα στοιχεία g^a, g^b που ανήκουν σε ένα σύνολο στοιχε-

ίων, ορίζει την απεικόνηση του γινομένου τους σε ένα στοιχείο ενός διαφορετικού συνόλου.

$$e(g^a, g^b) = e(g, g)^{ab}$$

Δεδομένου ότι το πεδίο τιμών διαφέρει από το πεδίο ορισμού, δεν είναι εφικτός ο συνδυασμός της εξόδου με μια κρυπτογραφημένη τιμή του πεδίου ορισμού. Το παραπάνω σχήμα απεικόνισης βασίζεται σε ιδιότητες ελλειπτικών καμπύλων και έχει τις εξής βασικές ιδιότητες:

$$e(g^a, g^b) = e(g^b, g^a) = e(g^{ab}, g^1) = e(g^1, g^{ab}) = e(g^1, g^b)^a = \ldots = e(g^1, g^1)^{ab}$$

$$e(g^a, g^b) \cdot e(g^c, g^d) = g^{ab} \cdot g^{cd} = g^{ab+cd} = e(g, g)^{ab+cd}$$

Με χρήση του cryptographic pairing, καθίσταται εφικτή η αναπαράσταση του γινομένου δύο τιμών, όταν αυτές βρίσκονται εξίσου σε κρυπτογραφημένη μορφή. Η διαδικασία αυτή βοηθά στη δημιουργία δημοσίων και επαναχρησιμοποιήσιμων παραμέτρων (CRS), χωρίς την ανάγκη διατήρησης κρυφών τιμών από τον επαληθευτή. Έτσι, οποιοσδήποτε έχει πρόσβαση σε αυτές τις παραμέτρους μπορεί να επαληθεύσει μια απόδειξη, χωρίς να απαιτείται η επανάληψη ολόκληρου του πρωτοκόλλου ξεχωριστά για κάθε επαληθευτή.

Πρωτόκολλο για πολυώνυμο

Λαμβάνοντας όλα τα παραπάνω υπόψιν, καταλήγουμε στο εξής πρωτόκολλο zk-SNARKOP (zero-knowledge Succint Non-Interactive Argument of Knowledge of Polynomial):

- **Κατασκευή δημοσίων παραμέτρων**
  Για δύο τυχαίους αριθμούς r, a υπολογίζονται τα $g^a, \{g^{r^i}\}_{i\in[1,d]}, \{g^{ar^i}\}_{i\in[0,d]}$. Στη συνέχεια ορίζεται και δημοσιοποιείται το ζεύγος proving και verification key, ενώ οι τυχαίες τιμές καταστρέφονται χωρίς να αποκαλύφθούν σε κανέναν.
  proving key: $\left(\{g^{r^i}\}_{i\in[1,d]}, \{g^{ar^i}\}_{i\in[0,d]}\right)$
  verification key: $\left(g^a, g^{t(r)}\right)$

- **Κατασκευή απόδειξης**
  Ο αποδείκτης γνωρίζει ένα πολυώνυμο της μορφής $p(x) = c_d x^d + \ldots + c_1 x^1 + c_0 x^0$. Κατασκευάζει το πολυώνυμο $h(x) = p(x)/t(x)$ και υπολογίζει τα $g^{p(r)}, g^{ap(r)}, g^{h(r)}$, χρησιμοποιώντας τις κρυπτογραφημένες τιμές του proving key. Επιπλέον, επιλέγει ένα τυχαίο δ και πολλαπλασιάζει με αυτό τα στοιχεία που υπολόγισε (χρησιμοποιείται διαφορετικό δ σε κάθε απόδειξη, κρύβοντας την πραγματική συμπεριφορά του πολυωνύμου). Τέλος, αποκαλύπτει την απόδειξη
  proof: $\left(g^{\delta p(r)}, g^{\delta h(r)}, g^{\delta ap(r)}\right)$

- **Επαλήθευση**
  Ο επαληθευτής ελέγχει αν η αποτίμηση του πολυωνύμου κατασκευάστηκε με χρήση των κρυπτογραφημένων τιμών που περιέχονται στο proving key, δηλαδή αν ισχύει η ισότητα $e(g^{\delta ap(r)}, g) = e(g^{\delta p(r)}, g^a)$. Τέλος, ελέγχεται αν το πολυώνυμο του αποδείκτη έχει πράγματι τις επιθυμητές ρίζες, δηλαδή έχει ως παράγοντα το t(x), $e(g^{\delta p(r)}, g) = e(g^{t(r)}, g^{\delta h(r)})$.

Γενίκευση

Το παραπάνω πρωτόκολλο επιτρέπει την απόδειξη γνώσης ενός πολυωνύμου που διαθέτει ορισμένες ρίζες. Αν και κάτι τέτοιο είναι αρκετά χρήσιμο, ο αποδείκτης μπορεί εύκολα να κατασκευάσει και να αποδείξει οποιοδήποτε πολυώνυμο προκύπτει από τον πολλαπλασιασμό του t(x) με ένα αυθαίρετο h(r). Έτσι τα zk-SNARKs επεκτείνουν το πρωτόκολλο αυτό, περιορίζοντας τη μορφή του πολυωνύμου ώστε να ισχύει $p(x) = L(x) \cdot R(x) - O(x)$, το οποίο αντιστοιχεί στη μορφή του QAP. Επιπλέον, επιτρέπουν την επιβολή ορισμένων δημοσίων εισόδων από τον επαληθευτή στον προς απόδειξη υπολογισμό. Η βάση του πλήρους πρωτοκόλλου παρουσιάζεται αναλυτικότερα στο [30].

5.4 ZoKrates

Το ZoKrates είναι μια εργαλειοθήκη για κατασκευή zk-SNARK αποδείξεων, και την επαλήθευση τους στο Ethereum. Δεδομένου ότι ο σχεδιασμός του QAP για μεγάλους υπολογισμούς είναι μια πολύπλοκη διαδικασία, ορίζει μια γλώσσα υψηλού επιπέδου, στην οποία μπορούν να κατασκευαστούν τα προς απόδειξη προγράμματα. Η γλώσσα αυτή περιέχει κάποιος εγγενής περιορισμούς ως προς την εκφραστικότητας της. Στην συνέχεια κατασκευάζει το ισοδύναμο QAP, επιτρέποντας την εκτέλεση των επιμέρους σταδίων του πρωτοκόλλου των zk-SNARKs, χωρίς ο προγραμματιστής να χρειάζεται να γνωρίζει πως αυτά λειτουργούν. Προσφέρει δηλαδή μια υψηλού επιπέδου διεπαφή για off-chain δημιουργία και επαλήθευση zk-SNARK προγραμμάτων, καθώς επίσης μπορεί να κατασκευάσει κατάλληλο smart contract για την on-chain επαλήθευση των zk-SNARK αποδείξεων [31].

Μέρος II

Πρακτικό Μέρος

Κεφάλαιο 6

Αρχιτεκτονική συστήματος

6.1 Κακόβουλος κόμβος

Όπως παρουσιάστηκε στο κεφάλαιο 2.4, η αρχιτεκτονική του HDFS διαθέτει μηχανισμούς για τον εντοπισμό αλλοιώσεων για τα δεδομένων. Όμως όλοι αυτοί οι μηχανισμοί βασίζονται στον ισχυρισμό πως οι κόμβοι του συστήματος είναι έμπιστοι. Ένας κακόβουλος DataNode θα μπορούσε να εμποδίσει την ομαλή λειτουργία του συστήματος με πολλούς διαφορετικούς τρόπους, από τον ψευδή ισχυρισμό κατοχής μπλοκ, έως και την αλλοίωση των δεδομένων που πραγματικά διαθέτει.

Υπό κανονικές συνθήκες το BlockScanner θα ήταν σε θέση να εκθέσει τυχόν ανεπιθύμητες τροποποιήσεις σε ένα μπλοκ, διασταυρώνοντας το περιεχόμενό του με τα αθροίσματα ελέγχου στο αντίστοιχο αρχείο μεταδεδομένων. Όμως, η λειτουργία του μπορεί εύκολα να απενεργοποιηθεί από την πλευρά του DataNode (θέτοντας την παράμετρο dfs.datanode.scan.period.hours σε έναν αρνητικό αριθμό στο αρχείο hdfs-site.xml), καθιστώντας οποιεσδήποτε κακόβουλες ενέργειες στα δεδομένα μη ανιχνεύσιμες κατά αυτόματο τρόπο. Επιπλέον, τα αρχεία δεδομένων και μεταδεδομένων ενός αντιγράφου μπλοκ, βρίσκονται εξίσου στο τοπικό σύστημα αρχείων του DataNode. Οπότε τα αθροίσματα ελέγχου δεν μπορούν να θεωρηθούν έμπιστα, ενώ η τροποποίησή τους θα μπορούσε να υποκρύψει οποιεσδήποτε αλλοιώσεις στο περιεχόμενο των δεδομένων, τόσο κατά τη λειτουργία του BlockScanner, όσο και κατά την ανάκτηση και τον έλεγχο του μπλοκ από τον χρήστη.

Αναφορικά με τη διαδικασία του blockreport, ένας DataNode μπορεί να διαγράψει ένα μπλοκ ενώ συνεχίζει να δηλώνει την κατοχή του στον NameNode, απλά διατηρώντας και στέλνοντας μόνο τα αντίστοιχα μεταδεδομένα. Επιπλέον, στην περίπτωση όπου ο NameNode της HDFS συστάδας είναι κακόβουλος, μπορεί να συνεργαστεί με κάποιον DataNode παρουσιάζοντας ένα αλλοιωμένο μπλοκ ως έγκυρο.

6.2 Επισκόπηση αρχιτεκτονικής συστήματος

Είναι φανερό πως σε μια συστάδα αναξιόπιστων κόμβων, οι μηχανισμοί του HDFS αδυνατούν να ανταπεξέλθουν. Έτσι, εκμεταλλευόμενοι την ασφάλεια και τη διαφάνεια του Ethereum blockchain, παρουσιάζουμε ένα σύστημα που μπορεί να παρέχει ισχυρές εγγυήσεις για την ασφάλεια και την ακεραιότητα των δεδομένων, δημιουργώντας ένα έξυπνο συμβόλαιο που θα λειτουργεί ως κεντρική αρχή ελέγχου.

Οι χρήστες, για κάθε μπλοκ δεδομένων που αποθηκεύουν στο HDFS, το χωρίζουν σε μικρότερα τμήματα σταθερού μεγέθους (chunks) και κατασκευάζουν το Merkle tree. Στη συνέχεια αποθηκεύουν τη ρίζα του στο έξυπνο συμβόλαιο.

Κάθε DataNode κατά τη διάρκεια ενός blockreport, υποχρεούται να κατασκευάζει κατάλληλα Merkle proofs για κάθε μπλοκ που διαθέτει, που θα αποδεικνύουν την ακεραιότητα ενός τμήματος δεδομένων του μπλοκ αυτού. Προκειμένου να αποφεύγεται η δημοσίευση του περιεχομένου των αρχείων στο blockchain, οι DataNodes δημιουργούν αποδείξεις βάση της μεθόδου των zk-SNARKs για τα διάφορα Merkle proofs, οι οποίες και επαληθεύεται τελικά on-chain.

Έτσι κάθε DataNode μπορεί να διαψευστεί ή να πείσει για την ακεραιότητα των δεδομένων που διαθέτει κατά ασφαλή και επαληθεύσιμο τρόπο, χωρίς να θυσιάζεται η εμπιστευτικότητα του περιεχομένου των αρχείων, αφού οι αποδείξεις που υποβάλλονται στο blockchain δεν αποκαλύπτουν τίποτα σχετικά με τα ίδια τα δεδομένα. Τα αποτελέσματα των ελέγχων καταγράφονται στο blockchain και είναι διαθέσιμα προς τους χρήστες για αξιολόγηση.

Η υλοποίηση μπορεί να διακριθεί σε τρία βασικά τμήματα, τη μέθοδο παραγωγής των αποδείξεων, το έξυπνο συμβόλαιο και τις τροποποιήσεις στην αρχιτεκτονική του HDFS. Αυτά παρουσιάζονται αναλυτικότερα στο επόμενα κεφάλαια, συνοδευόμενα από τις σχεδιαστικές αποφάσεις που λήφθηκαν κατά τον σχεδιασμό τους.

6.3 Αποδεικτικό μέσο

Η απόδειξη της ακεραιότητας ενός μπλοκ δεδομένων επιτυγχάνεται με την κατασκευή και την επαλήθευση αποδείξεων βασισμένων στα Merkle proofs, για ένα ή περισσότερα τμήματα αυτού. Δηλαδή, ο ισχυρισμός γνώσης που πρέπει να αποδείξει ένας DataNode μπορεί να εκφραστεί ως \η κατοχή ενός τμήματος δεδομένων και ενός συνόλου από hashes του Merkle tree, τέτοια ώστε η κατασκευή κατάλληλου μονοπατιού από αυτά οδηγεί στο ζητούμενο Merkle root". Για την κατασκευή κατάλληλων zk-SNARK αποδείξεων απαιτείται η έκφραση αυτού, σε μορφή υπολογιστικού προγράμματος στη ψευδογλώσσας του ZoKrates (6.1).

Ως δημόσιες εισόδους στο πρόγραμμα χρειαζόμαστε το Merkle root του μπλοκ, καθώς και έναν δείκτη (index) προς το ζητούμενο chunk στο οποίο αντιστοιχεί η απόδειξη. Ο δείκτης αυτός επιλέγεται κατά τυχαίο τρόπο όπως θα δούμε στη συνέχεια και είναι απαραίτητο να επιβάλλεται κατά τη διαδικασία της επαλήθευσης, αλλιώς ο DataNode θα μπορούσε να υποβάλει οποιαδήποτε απόδειξη επιθυμεί, αποφεύγοντας τυχόν αλλοιωμένα τμήματα του μπλοκ.

Οι ιδιωτικοί είσοδοι του προγράμματος, είναι τα hashes για την κατασκευή του μονοπατιού (αντίστοιχα της διαδικασίας που παρουσιάζεται στο 3.3), καθώς και το περιεχόμενο του προς απόδειξη τμήματος δεδομένων. Η χρήση μόνο του hash του τμήματος δεδομένων στην απόδειξη, θα επέτρεπε στον DataNode να αποθηκεύει τα φύλλα του Merkle tree, και να κατασκευάζει τις αποδείξεις χρησιμοποιώντας αυτά, χωρίς απαραίτητα να κατέχει τα εν λόγω δεδομένα. Έτσι η συμπερίληψη των δεδομένων στην απόδειξη χρησιμοποιείται για την απόδειξη της κατοχής των εν λόγω δεδομένων.

Λόγω εγγενών περιορισμών του ZoKrates, οι επαναλήψεις των βρόχων του προγράμματος δεν μπορούν να καθορίζονται δυναμικά, αλλά πρέπει να είναι γνωστές κατά τη μεταγλώττι-

ση, καθώς επίσης οι πίνακες στις εισόδους είναι στατικού μεγέθους. Έτσι, αναγκαζόμαστε να περιορίσουμε τα Merkle proofs ώστε να χρησιμοποιούν ένα σταθερού μεγέθους Merkle tree, ανεξαρτήτως του μεγέθους του μπλοκ. Δηλαδή, τόσο το μέγεθος των τμημάτων δεδομένων που αναπριστούν τα φύλλα (chunks), όσο και το ύψος του δέντρου, καθορίζεται κατά την παραμετροποίηση του συστήματος, και παραμένουν ως έχειν καθ' όλη τη διάρκεια της λειτουργίας του. Έτσι ορίζεται ένα μέγιστο υποστηριζόμενο μέγεθος μπλοκ, $blockSizeSafeMax = chunkSize * 2^{treeHeight}$. Σε περίπτωση που το μπλοκ, είναι μικρότερο του μέγιστου υποστηριζόμενου μεγέθους, προτιμάται η συμπλήρωση του υπολειπόμενου τμήματος με την επανάληψη της ακολουθίας του περιεχομένου του, όσες φορές απαιτείται. Αυτό συμβαίνει μόνο κατά τη δημιουργία των Merkle trees, οπότε τα αποθηκευμένα μπλοκ στο σύστημα αρχείων του DataNode μπορούν να έχουν οποιοδήποτε μέγεθος ορίζεται από το HDFS (ή και μικρότερο). Έτσι κάθε chunk του Merkle tree περιέχει κάποιο τμήμα δεδομένων του αντίστοιχου μπλοκ, απλοποιώντας την επιλογή των δεικτών/προκλήσεων που θα εκδίδονται για κάθε μπλοκ.

```
import "hashes/blake2/blake2s" as blake2
import "hashes/utils/256bitsDirectionHelper" as multiplex
import "utils/pack/u32/pack128" as pack

def merkle_proof<N,H>(u32 index,
                      private u32[N][16] chunk,
                      private u32[H][8] siblings,
                      private bool[H] pathDirection) -> field[2]:
  // hash selected chunk
  u32[8] hs = blake2(chunk)
  u32 count = 0
  u32 lvl = 1
  // follow path to root
  for u32 i in 0..H do
    hs = blake2([multiplex(pathDirection[i], hs, siblings[i][0..8])])
    count = count + if pathDirection[i] then lvl else 0 fi
    lvl = lvl * 2
  endfor
  assert(count == index)
  return [pack(hs[0..4]), pack(hs[4..8])]


def main(u32 index, field[2] rootDigest,
         u32 randomness,
         private u32[8][16] chunk,
         private u32[16][8] siblings,
         private bool[16] pathDirection) -> bool:
  assert(rootDigest == merkle_proof(index, chunk, siblings, pathDirection))
```

```
return true
```

Listing 6.1: Πρόγραμμα επαλήθευσης Merkle proof

Η βοηθητική ιδιωτική μεταβλητή pathDirection χρησιμοποιείται για τον καθορισμό της κατεύθυνσης κατά την κατασκευή του μονοπατιού (δεδομένου ότι δεν υποστηρίζονται πράξεις υπολοίπου), καθώς και την επιβεβαιώσει υποβολής του σωστού chunk. Τέλος, χρησιμοποιείται ένα randomness στις δημόσιες εισόδους του προγράμματος, που λειτουργεί ως nullifier για τις αποδείξεις. Κατά την κατασκευή των αποδείξεων από το ZoKrates χρησιμοποιούμε το αποδεικτικό σχήμα Groth16, λόγω του χαμηλού κόστους gas κατά την επαλήθευση στο blockchain, καθώς και το μικρότερο χρόνο κατασκευής των αποδείξεων. Όμως μια οντότητα που έχει πρόσβαση σε μια απόδειξη αυτού του σχήματος, μπορεί είτε να την ξαναχρησιμοποιήσει είτε να κατασκευάσει μια διαφορετική έγκυρη απόδειξη για τον ίδιο ακριβώς ισχυρισμό, χωρίς να γνωρίζει τελικά την ιδιωτική γνώση [32]. Η χρήση μιας τυχαίας δημόσιας εισόδου που εφαρμόζεται κατά την επαλήθευση και είναι διαφορετική ανά blockreport και ανά DataNode, μπορεί να επιλύσει το παραπάνω πρόβλημα.

Για τον υπολογισμός των hashes χρησιμοποιούμε την κρυπτογραφική συνάρτηση κατακερματισμού Blake2s [33]. Χάρην της απλότητας και του μικρότερου πλήθους υπολογισμών που απαιτείται για τον υπολογισμό της, η αναπαράστασή της στο R1CS χρησιμοποιεί λιγότερους περιορισμούς, καταλήγοντας σε μικρότερο χρόνο κατασκευής των αποδείξεων σε σχέση με τις περισσότερες διαθέσιμες συναρτήσεις κατακερματισμού, χωρίς όμως να θυσιάζει την ασφάλεια και την ταχύτητα υπολογισμού.

Με χρήση του ZoKrates, κατόπιν μεταγλώττισης του προγράμματος και εκτέλεσης του setup, μοιράζεται το proving key σε όλους τους DataNodes (καθώς και το μεταγλωττισμένο πρόγραμμα) για να μπορούν να κατασκευάζουν τις αποδείξεις. Επιπλέον, με χρήση του verification key κατασκευάζεται το έξυπνο συμβόλαιο για την επαλήθευση των αποδείξεων (Verifier).

6.4 Έξυπνο συμβόλαιο

Το έξυπνο συμβόλαιο που κατασκευάσαμε, χρησιμοποιείται για την αποθήκευση των Merkle roots των μπλοκ, καθώς και την επαλήθευση των Merkle proofs που υποβάλλουν οι DataNodes. Η επιλογή των προκλήσεων για κάθε μπλοκ (δηλαδή τα chunk για τα οποία αναμένεται η υποβολή απόδειξης), καθορίζεται μέσω μιας γεννήτριας ψευδοτυχαίων αριθμών που είναι διαθέσιμη τόσο στους DataNodes όσο και στο έξυπνο συμβόλαιο. Η γεννήτρια αυτή, λαμβάνει ως ορίσματα ένα σπόρο και ένα αναγνωριστικό μπλοκ, και παράγει k ψευδοτυχαίους αριθμούς στο εύρος τιμών $[0, 2^{treeHeight} - 1]$. Δεδομένου ότι οι κόμβοι της HDFS συστάδας είναι αναξιόπιστοι, ο σπόρος επιλέγεται κατά εξίσου ψευδοτυχαίο τρόπο on-chain ξεχωριστά για κάθε DataNode και ανανεώνεται κατά την έναρξη ενός blockreport.

Οπότε, οι βασικές πληροφορίες που αποθηκεύονται στο έξυπνο συμβόλαιο, είναι το Merkle root κάθε μπλοκ, και ο σπόρος κάθε DataNode. Επιπλέον, είναι απαραίτητος ο ορισμός του πλήθους των αποδείξεων ανά μπλοκ που υποβάλλονται (k), το πλήθος των φύλλων του Merkle tree που ορίζουμε κατά την αρχική παραμετροποίηση του συστήματος

($2^{treeHeight}$) καθώς και η διεύθυνση του έξυπνου συμβολαίου που δημιουργεί το ZoKrates και χρησιμοποιείται για τον έλεγχο της κάθε απόδειξης.

Για την εξασφάλιση της ορθής λειτουργίας του συστήματος, το έξυπνο συμβόλαιο υλοποιεί την ελάχιστη δυνατή διεπαφή στις συναρτήσεις που εκτελούν οι DataNodes, ενώ χρησιμοποιεί τη δημόσια διεύθυνση που συνοδεύει τις συναλλαγές για την αναγνώριση του αποστολέα. Έτσι, όσο το ιδιωτικό κλειδί του λογαριασμού κάθε DataNode παραμένει μυστικό, το Ethereum εγγυάται την αυθεντικότητα των συναλλαγών. Οι διεπαφές των συναρτήσεων είναι οι εξής:

- **add_digest(blockpoolId , blockId, root)**: Καλείται από τους χρήστες για την αποθήκευση του Merkle root ενός μπλοκ.

- **create_seed(blockpoolId , randomness) και peek_seed(blockpoolId)**: Χρησιμοποιούνται από κάθε DataNode για την ανανέωση και ανάκτηση του σπόρου τους αντίστοιχα. Για την ανανέωση του σπόρου, παρέχεται μια τυχαία παράμετρος αποκλειστικά για τυπικούς λόγους, αφού εξαιτίας του ντετερμινισμού του blockchain δεν μπορεί να επιτευχθεί η παραγωγή πραγματικά τυχαίων αριθμών χωρίς κάποια εξωτερική παρέμβαση. Η παραγωγή του σπόρου εξαρτάται επιπλέον από τη διεύθυνση του DataNode και τη χρονοσφραγίδα του μπλοκ (στο blockchain) στο οποίο συμπεριλαμβάνεται η συναλλαγή. Η χρονοσφραγίδα αυτή χρησιμοποιείται και ως χρονική σήμανση της εκκίνησης ενός on-chain blockreport, με όλες τις επόμενες υποβολές αποδείξεων του συγκεκριμένου DataNode να τη χρησιμοποιούν στα event που καταγράφουν, έως την επόμενη ανανέωση του σπόρου. Έτσι επιτυγχάνεται ο χρονικός διαχωρισμός των διαφορετικών blockreport ανά DataNode.

- **verify(blockpoolId, blockId, numbers[])**: Η συνάρτηση αυτή χρησιμοποιείται για την επαλήθευση k αποδείξεων που υποβάλει ένας DataNode για το μπλοκ με αναγνωριστικό blockId. Χρησιμοποιώντας το αποθηκευμένο Merkle root του μπλοκ, και τον σπόρο του DataNode αποστολέα (παράγοντας τα challenges μέσω της γεννήτριας ψευδοτυχαίων αριθμών), επαληθεύει την εγκυρότητα των αποδείξεων μέσω του Verifier συμβολαίου που κατασκευάστηκε από το ZoKrates. Κατόπιν της επαλήθευσης, καταγράφεται ένα event της μορφής BlockReport(address datanode, uint time, uint blockId, bool corrupt), που αποκαλύπτει τη δημόσια διεύθυνση του DataNode, το αποτέλεσμα του ελέγχου και τη χρονοσφραγίδα του τρέχοντος blockreport.

Προκειμένου να λάβουμε υπόψιν την αρχιτεκτονική της ομοσπονδιακής HDFS συστάδας (2.4.6), προσθέσαμε ένα επιπλέον επίπεδο διαχωρισμού ως προς το αναγνωριστικό του Block Pool. Δηλαδή μέσω ενός mapping αντιστοιχίζουμε ένα αναγνωριστικό Block Pool σε μία δομή που διατηρεί τους σπόρους και τα Merkle roots που αντιστοιχούν στο Block Pool αυτό. Το blockpoolId παρέχεται στις αντίστοιχες κλήσεις συναρτήσεων του έξυπνου συμβολαίου για την ορθή εκτέλεση των διαφόρων λειτουργιών τους.

Ο πλήρης κώδικας του έξυπνου συμβολαίου παρατίθεται σε αντίστοιχο παράρτημα για λόγους πληρότητας.

6.5 Παρεμβάσεις στο HDFS

6.5.1 Παραμετροποιήσιμα χαρακτηριστικά

Η τροποποίηση της αρχιτεκτονική του HDFS, συνοδεύεται από την εισαγωγή καινούργιων παραμέτρων, που ορίζονται πριν την εκκίνηση του συστήματος. Όπως ήδη αναφέραμε, οι βασικότερες αυτών είναι το ύψος των Merkle trees που κατασκευάζονται για τον έλεγχο των δεδομένων, καθώς και το μέγεθος των τμημάτων δεδομένων (chunks) που αντιπροσωπεύουν τα φύλλα του. Οι παράμετροι αυτοί δεν εξαρτώνται άμεσα από το μέγεθος μπλοκ. Δηλαδή οι χρήστες του συστήματος είναι σε θέση να επιλέγουν το μέγεθος μπλοκ ανά αρχείο που γράφουν ξεχωριστά, όπως ορίζεται από το HDFS. Όμως το μέγιστο πλήθος δεδομένων για κάθε μπλοκ που μπορεί να επαληθευτεί από το πρωτόκολλο, είναι ίσο με $blockSizeSafeMax = chunkSize * 2^{treeHeight}$. Αν το μέγεθος μπλοκ επιλεχθεί μεγαλύτερο, τα δεδομένα που περισσεύουν είναι ευάλωτα σε αλλοιώσεις.

Επιπλέον, το σύστημα επιβάλει την κατοχή ξεχωριστών ethereum λογαριασμών για κάθε DataNode που χρησιμοποιούνται για την αλληλεπίδραση με το έξυπνο συμβόλαιο και καθορίζουν την ταυτότητά τους κατά το blockreport. Κάθε DataNode διαθέτει από ένα wallet αρχείο για τη διαχείριση του λογαριασμού του, στο οποίο έχει πρόσβαση μόνο ο ίδιος και το λογισμικό που εκτελεί. Αντίστοιχα και οι χρήστες του συστήματος πρέπει να διαθέτουν έναν ethereum λογαριασμό για την υποβολή των Merkle roots στο έξυπνο συμβόλαιο. Για την πρόσβαση του λογισμικού στο wallet, πρέπει να οριστούν το μονοπάτι προς το αντίστοιχο αρχείο και ο κωδικός πρόσβασης ως παράμετροι.

Άλλες παράμετροι που απαιτούνται είναι η διεύθυνση του έξυπνου συμβολαίου στο blockchain, η IP διεύθυνση ενός κόμβου του blockchain στον οποίο στέλνονται οι συναλλαγές που δημιουργούνται από τους κόμβους του συστήματος, το αναγνωριστικό του δικτύου ethereum και το πλήθος των Merkle proofs που πρέπει να υποβάλλονται ανά μπλοκ.

Επιπλέον, παρέχουμε τη δυνατότητα παράλληλης κατασκευής των Merkle trees καθώς και των zk-SNARK αποδείξεων κατά το blockreport, με το πλήθος νημάτων που χρησιμοποιούνται στην κάθε λειτουργία να είναι παραμετροποιήσιμο, για την καλύτερη διαχείρηση των πόρων του υπολογιστή. Όλες οι παράμετροι αυτοί, ορίζονται στο αρχείο hdfs-site.xml, μαζί με τις υπόλοιπες του HDFS.

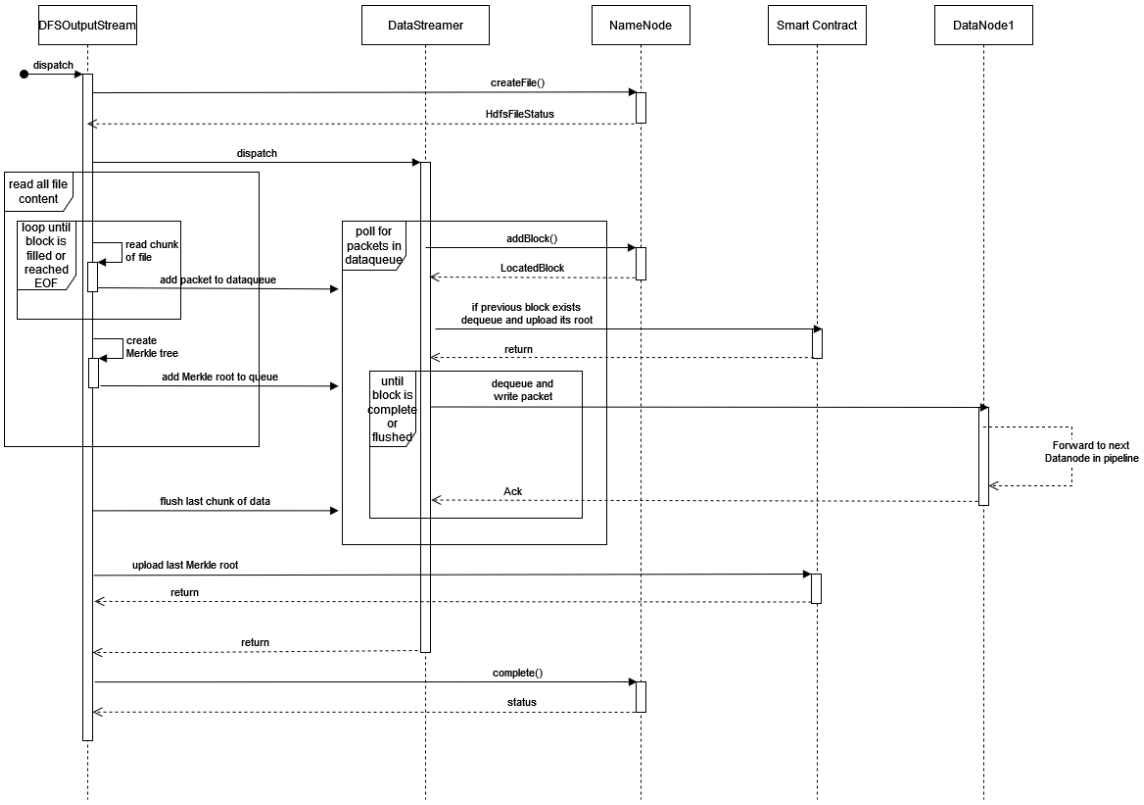
6.5.2 Γράφοντας ένα αρχείο

Θεωρώντας ότι οι κόμβοι του HDFS είναι αναξιόπιστοι, η εύρεση και αποθήκευση των Merkle roots επαφίεται στην ευθύνη του χρήστη. Για κάθε μπλοκ δεδομένων που γράφει στο σύστημα, ορίζει $2^{treeHeight}$ chunks σταθερού μεγέθους ως φύλλα του δέντρου, τα οποία και γεμίζει με το περιεχόμενο του εν λόγω μπλοκ. Το περιεχόμενο του μπλοκ επαναλαμβάνεται έως ότι συμπληρωθούν όλα τα chunks. Στη συνέχεια, χρησιμοποιώντας την κρυπτογραφική συνάρτηση κατακερματισμού Blake2s, κατασκευάζει το Merkle tree και υποβάλει την ρίζα του για αποθήκευση στο έξυπνο συμβόλαιο μέσω της συνάρτησης add_digest(). Το υπόλοιπο πρωτόκολλο λειτουργίας του HDFS παραμένει ως έχειν.

Πιο συγκεκριμένα, όπως παρουσιάστηκε στο θεωρητικό μέρος (Κεφάλαιο 2.4.7), στο λογισμικό που εκτελεί ο χρήστης για την εγγραφή ενός αρχείου στο HDFS, εκτελούνται δύο

νήματα για την ανάγνωση του αρχείου από το τοπικό σύστημα αρχείων και την αποστολή του στους DataNodes αντίστοιχα. Πλέον, το DFSOutputStream διατηρεί έναν buffer όπου συσσωρεύει τα δεδομένα ενός μπλοκ, κατά την ανάγνωσή του από το τοπικό σύστημα αρχείων. Μόλις συμπληρωθεί ένα μπλοκ, δημιουργεί το Merkle tree και εξάγει τη ρίζα του. Δεδομένου ότι δεν έχει δεσμευθεί ακόμα κάποιο αναγνωριστικό για το εν λόγω μπλοκ, το τοποθετεί σε μια ουρά. Η ουρά αυτή, ορίζει μια σχέση παραγωγού-καταναλωτή μεταξύ των δύο νημάτων (όπως και η ουρά για τα πακέτα δεδομένων).

Ο DataStreamer, για κάθε μπλοκ που μεταδίδει επικοινωνεί με τον NameNode για την παραγωγή και δέσμευση ενός αναγνωριστικού. Μετά την ολοκλήρωση της μετάδοσης ενός μπλοκ, αφαιρεί το αντίστοιχο Merkle root από την ουρά και κατασκευάζει μια συναλλαγή για κλήση της συνάρτησης add_digest(blockpoolId, blockId, merkleRoot) του έξυπνου συμβολαίου. Δεδομένου ότι και οι δύο ουρές, πακέτων και Merkle root, είναι FIFO (First In First Out) δεν απαιτείται κάποιος συγχρονισμός για τη σωστή αντιστοίχηση των Merkle roots στα μπλοκ. Εξαίρεση αποτελεί το τελευταίο μπλοκ του αρχείου για το οποίο η συναλλαγή κατασκευάζεται από το DFSOutputStream κατόπιν αποστολής των τελευταίων πακέτων από τον DataStreamer. Το τροποποιημένο πρωτόκολλο παρουσιάζεται στο σχήμα 6.1.



Σχήμα 6.1: *Modified write file sequence diagram*

Η αποστολή όλων των συναλλαγών ανατίθεται σε ξεχωριστό νήμα που διατηρεί μια σύνδεση με τον κόμβο, για την αποφυγή καθυστερήσεων στις βασικές λειτουργίες του HDFS, όμως για λόγους απλότητας δε φαίνεται στο σχήμα. Το νήμα αυτό αναμένει σε μια συγχρονισμένη λίστα για καινούργια ζεύγη αναγνωριστικού μπλοκ και Merkle root, τα οποία προσθέτουν τα δύο βασικά νήματα που μετέχουν στην εγγραφή του αρχείου. Για κάθε ζεύγος που λαμβάνει, δημιουργεί μια νέα συναλλαγή την οποία και στέλνει στον .

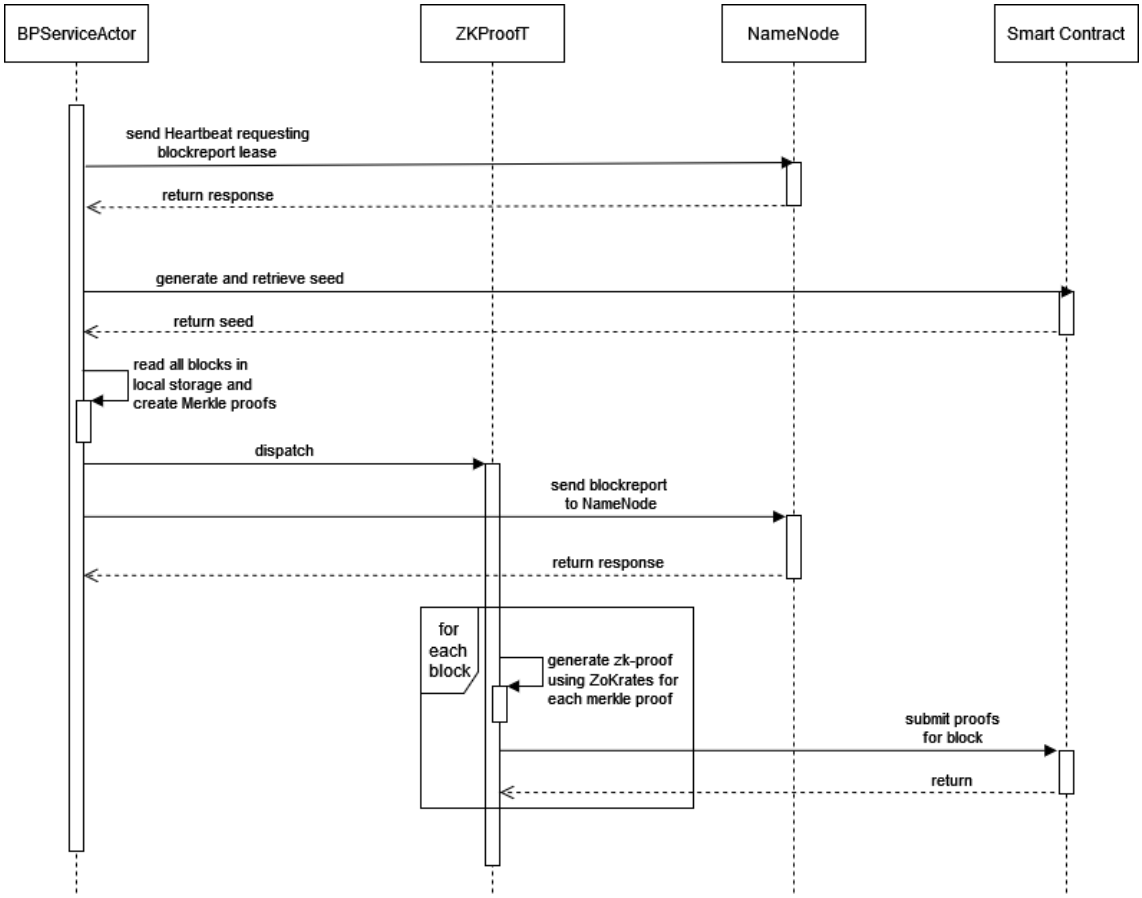
6.5.3 Blockreport

Κάθε DataNode εκκινεί ένα blockreport ανά κάποιο ορισμένο χρονικό διάστημα, ενημερώνοντας τον NameNode σχετικά με τα μπλοκ που διαθέτει. Η διαδικασία αυτή μπορεί επίσης να πραγματοποιηθεί κατά παράγγελμα του χρήστη με χρήση της εντολής hdfs dfsadmin -triggerBlockReport <datanodeIP:port>. Δεδομένου ότι ο DataNode μπορεί να ισχυριστεί την κατοχή οποιουδήποτε μπλοκ, είναι επιθυμητή η απόδειξη αυτού του ισχυρισμού. Για το σκοπό αυτό, κάθε DataNode, κατά τη διάρκεια του blockreport κατασκευάζει κατάλληλα Merkle proofs για όλα τα μπλοκ δεδομένων που διαθέτει. Στη συνέχεια, χρησιμοποιώντας τα Merkle proofs, το ZoKrates και το proving key κατασκευάζει zk-SNARK αποδείξεις τις οποίες και υποβάλει στο έξυπνο συμβόλαιο προς επαλήθευση. Κάθε απόδειξη είναι ικανή να πείσει σχετικά με την κατοχή και την ακεραιότητα ενός τμήματος (chunk) του αντίστοιχου μπλοκ δεδομένων.

Η διαδικασία που ακολουθείται κατά το τροποποιημένο blockreport παρουσιάζεται στο σχήμα 6.2. Το νήμα BPServiceActor είναι υπεύθυνο για την περιοδική αποστολή heartbeat. Προκειμένου να ξεκινήσει ένα blockreport, ο DataNode στέλνει ένα heartbeat αιτούμενος για ένα blockreport lease από τον NameNode, το οποίο και ισχύει για ένα ορισμένο χρονικό διάστημα. Στη συνέχεια επιλέγει ένα τυχαίο αριθμό (randomness) και δημιουργεί μια συναλλαγή για ανανέωση του σπόρου του στο έξυπνο συμβόλαιο μέσω της create_seed(blockpoolId, randomness), τον οποίο και ανακτά μέσω της peek_seed(blockpoolId). Γνωρίζοντας πλέον το σπόρο, μπορεί να παράγει τις προκλήσεις που του αναλογούν για κάθε μπλοκ, χρησιμοποιώντας την γεννήτρια ψευδοτυχαίων αριθμών.

Για κάθε μπλοκ που διαθέτει στο τοπικό σύστημα αρχείων, διαβάζει το περιεχόμενό του και κατασκευάζει το Merkle trees (αντίστοιχα της μεθόδου που ακολουθεί και ο χρήστης). Δεδομένου του σπόρου και του αναγνωριστικού του εν λόγω μπλοκ, βρίσκει τις ζητούμενες k πλήθους προκλήσεις και κατασκευάζει Merkle proofs για τα αντίστοιχα φύλλα του δέντρου. Τα Merkle proofs κάθε μπλοκ αποθηκεύονται προσωρινά σε μία λίστα. Αφότου, ολοκληρωθεί η διαδικασία για όλα τα μπλοκ του DataNode, η λίστα με τα Merkle proofs παραχωρείται σε ξεχωριστό νήμα ZKProofT, ενώ το BPServiceActor στέλνει στον NameNode τη λίστα με τα μπλοκ όπως ορίζει το κλασικό πρωτόκολλο του HDFS επιστρέφοντας στην κανονική του λειτουργία.

Το ZKProofT με τη βοήθεια του ZoKrates, κατασκευάζει zk-SNARK αποδείξεις για τα k Merkle proofs του κάθε μπλοκ που βρίσκεται στη λίστα, και τις υποβάλλει για επαλήθευση στο έξυπνο συμβόλαιο μέσω της verify(blockpoolId, blockId, numbers[]). Η κατασκευή των zk-SNARK αποδείξεων είναι αρκετά απαιτητική διαδικασία ως προς τη διάρκεια και την καταναλισκόμενη υπολογιστική ισχύ. Έτσι η ανάθεσή της ξεχωριστό νήμα επιτρέπει τον διαχωρισμό του blockreport σε off-chain και on-chain στάδια, όπου μόνο κατά το πρώτο ο DataNode δεν είναι αποκρίσιμος. Δηλαδή το on-chain blockreport δεν καθυστερεί την επιστροφή στη κανονική λειτουργία του DataNode, όμως είναι πιθανό να μειώνει τη διεκπεραιωτική του ικανότητα, μέχρις ότου ολοκληρωθεί.



Σχήμα 6.2: *Modified blockreport sequence diagram*

Κεφάλαιο 7

Πειραματική αξιολόγηση

7.1 Μεθοδολογία

Η πειραματική αξιολόγηση πραγματοποιήθηκε σε μία απλή συστάδα HDFS που αποτελείται από μόλις έναν NameNode και έναν DataNode. Έτσι, όλα τα δεδομένα τοποθετούνταν στον μοναδικό DataNode του συστήματος, επιτρέποντας τον χειρισμό του πλήθους των μπλοκ και τελικά των αποδείξεων που υποβάλλονταν κατά το blockreport. Σε όλα τα πειράματα, παραχωρήθηκαν τέσσερις πυρήνες του επεξεργαστή αποκλειστικά για την κατασκευή των Merkle proofs και των zk-SNARK αποδείξεων κατά τα blockreports.

Σχετικά το Ethereum blockchain, χρησιμοποιήθηκε το εργαλείο Geth [34] για την εγκατάσταση και εκτέλεση ενός τοπικού, ιδιωτικού δικτύου με μόλις έναν κόμβο. Για την εξασφάλιση μιας σταθερής ροής καινούργιων μπλοκ στην αλυσίδα, δεσμεύτηκε ένας πυρήνας του επεξεργαστή αποκλειστικά για το mining. Επιπλέον τέθηκε κατάλληλα το difficulty κατά τη δημιουργία του genesis block, ώστε το απαιτούμενο χρονικό διάστημα εισαγωγής καινούργιων μπλοκ στην αλυσίδα να προσεγγίζει κατά μέσο όρο τα 24 δευτερόλεπτα.

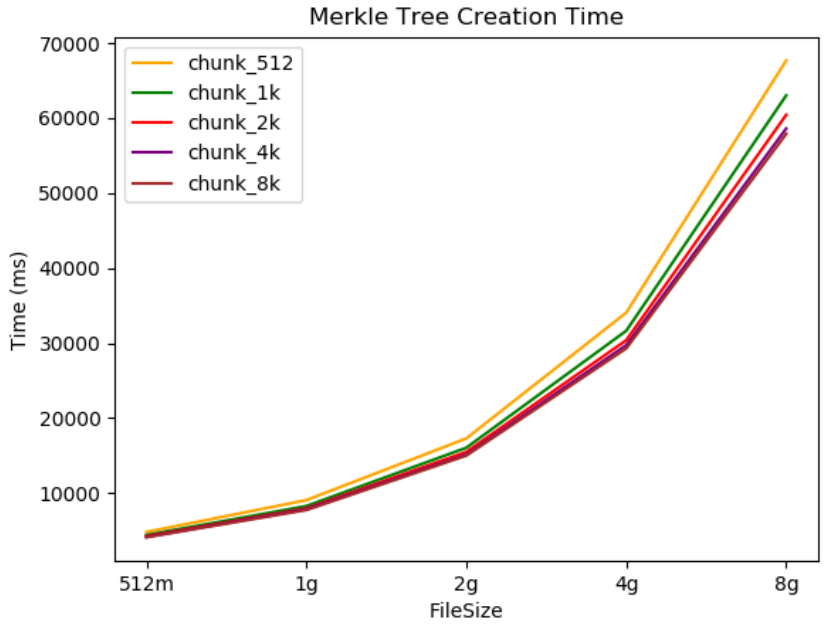
Για κάθε παραμετροποίηση του συστήματος που εξερευνήθηκε, προηγείται η μεταγλώττιση του προγράμματος για την επαλήθευση των Merkle proofs με χρήση του ZoKrates, καθώς και η παραγωγή του καινούργιου ζεύγους proving key και verification key. Αντίστοιχα, επαναλαμβανόταν η δημιουργία του έξυπνου συμβολαίου ώστε να φέρει τις νέες παραμέτρους και το καινούργιο verification key.

Για την εξαγωγή των μετρήσεων πραγματοποιήθηκε η εκτέλεση ενός πειράματος τουλάχιστον πέντε φορές για κάθε παραμετροποίηση του συστήματος, συνάγοντας τελικά την μέση τιμή των επιμέρους χρόνων εκτέλεσης. Επιπλέον, απαιτήθηκε κατάλληλος χειρισμός των ακραίων τιμών που προέκυψαν εξαιτίας πιθανών αστοχιών του υλικού ή άλλων εξωτερικών παραγόντων, με το πείραμα να επαναλαμβάνεται περισσότερες φορές όπου κρίθηκε απαραίτητο. Όλα τα παραπάνω στοιχεία λογισμικού εκτελούνταν στον ίδιο υπολογιστή, του οποίου οι προδιαγραφές παρατίθενται για λόγους πληρότητας.

- GNU/Linux 5.4.0-104-generic, Ubuntu 20.04.4 LTS
- AMD Ryzen 5 3600, 3.6 GHz
- RAM G.Skill RipjawsV DDR4-32000MHz (4x8GB)
- Western Digital Blue 1TB HDD, SATA III, 7200rpm, 64MB Cache

7.2 Εγγραφή αρχείου

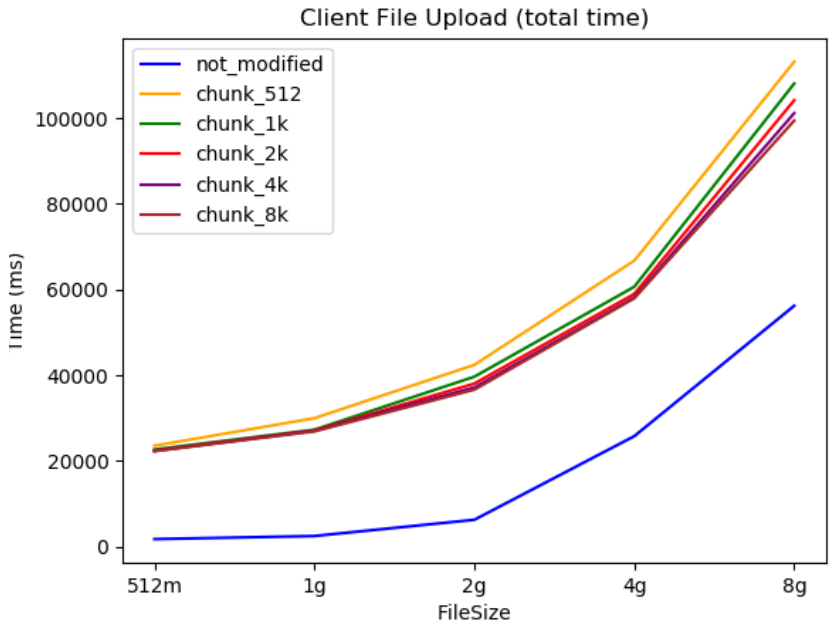
Η εγγραφή ενός αρχείου αποτελεί τη βασικότερη λειτουργία του συστήματος και μία από τις κυριότερες διεπαφές για τον χρήστη, καθιστώντας απαραίτητη την αξιολόγηση του αντίκτυπου που επιφέρουν οι τροποποιήσεις. Χρησιμοποιώντας μέγεθος μπλοκ 64 MBytes, πραγματοποιείται η εγγραφή ενός τυχαίου αρχείου μεγέθους {512MB, 1GB, 2GB, 4GB, 8GB} ({8, 16, 32, 64, 128} μπλοκ αντίστοιχα), και καταγράφεται το συνολικό χρονικό διάστημα διεκπεραίωσης της σε όλες της περιπτώσεις, καθώς και ο χρόνος που απαιτήθηκε αποκλειστικά για την κατασκευή όλων των Merkle trees.



Σχήμα 7.1: *Διάγραμμα διάρκειας κατασκευής των Merkle trees κατά την εγγραφή ενός αρχείου*

Όπως είναι αναμενόμενο, ο χρόνος κατασκευής των Merkle trees είναι μικρότερος για μεγαλύτερο chunk size, δεδομένου ότι μειώνεται το ύψος του δέντρου, άρα και το πλήθος των hashes που απαιτείται να υπολογιστούν. Η αντίστοιχη κατανομή φαίνεται ότι μεταφέρεται και στο συνολικό χρονικό διάστημα εγγραφής του αρχείου στο HDFS. Οπότε η χρήση μεγαλύτερου chunk size και μικρότερου ύψους δέντρους ευνοεί την ταχύτερη εγγραφή των αρχείων στο τροποποιημένο σύστημα.

Αξίζει να αναφέρουμε πως αν και η διαφορά από το μη τροποποιημένο HDFS παρουσιάζεται αρκετά μεγάλη, στην πραγματικότητα παρατηρήθηκε πως η εγγραφή ολοκληρωνόταν περίπου 15 δευτερόλεπτα νωρίτερα από το αναγραφόμενο. Το πρόβλημα φαίνεται να αποδίδεται στην okhhtp4 βιβλιοθήκη της Java, η οποία χρησιμοποιεί ορισμένα non-Daemon threads που τερματίζουν από μόνα τους μετά από ορισμένο χρονικό διάστημα αν δεν πραγματοποιηθούν καινούργιες HTTP συνδέσεις. Δεδομένου ότι αυτά δε διαθέτουν διεπαφή για τον εξαναγκασμένο τερματισμό τους, το JVM καταλήγει να κάνει προσωρινά hang αναμένοντας τον τερματισμό τους [35], κάτι που επιβεβαιώθηκε κατόπιν thread dump κατά τη



Σχήμα 7.2: *Διάγραμμα διάρκειας εγγραφής ενός αρχείου*

λειτουργία του συστήματος.

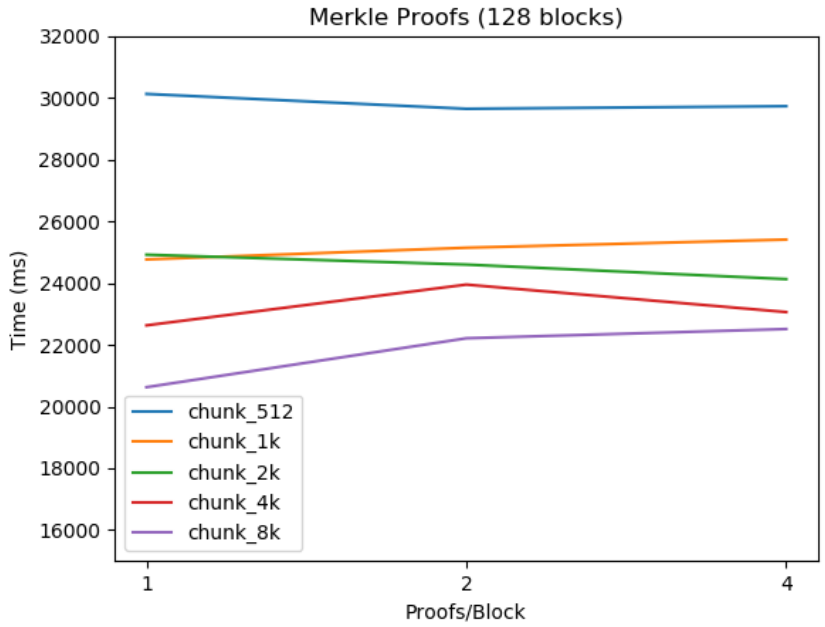
7.3 Blockreport

Για την αξιολόγηση του blockreport του συστήματος χρησιμοποιήθηκε blockSizeSafeMax ίσο με 64 MBytes, ώστε να επιτρέπει επαρκή εξερεύνηση των διαφορετικών παραμετροποιήσεως, χωρίς να υπερβαίνει τις δυνατότητες του υπολογιστή. Στη συνέχεια, για διαφορετικές παραμετροποιήσεις του chunk size και αντίστοιχα του ύψους των Merkle trees, καθώς και των απαιτούμενων αποδείξεων ανά μπλοκ, εκτελέστηκε το blockreport εξάγοντας τη χρονική διάρκεια αυτού και των επιμέρους σταδίων του.

Κατά τη διάρκεια των πειραμάτων ο DataNode διέθετε στο τοπικό του σύστημα αρχείων, δεδομένα συνολικού μεγέθους 8 GBytes, με το πλήθος των μπλοκ να ανέρχεται στα 128. Η επιλογή του μεγέθους πραγματοποιήθηκε με γνώμονα τη χρονική διάρκεια εκτέλεσης των περισσότερο απαιτητικών πειραμάτων, ώστε να μην ξεπερνά το προεπιλεγμένο χρονικά διάστημα μεταξύ διαδοχικών blockreport. Το chunk size λαμβάνει τιμές {512Bytes, 1KByte, 2KBytes, 4KBytes, 8KBytes} με το ύψος των Merkle trees να μεταβάλλεται αντίστοιχα {17, 16, 15, 14, 13}.

Αρχικά παρατίθεται το διάγραμμα που παρουσιάζει τη διάρκεια ανάγνωσης όλων των μπλοκ από το τοπικό σύστημα αρχείων και δημιουργίας των Merkle proofs (Σχήμα 7.3). Όπως φαίνεται στο διάγραμμα, η αύξηση του chunk size ευνοεί την ταχύτερη παραγωγή των Merkle proofs, καθώς μειώνει το μέγεθος των Merkle trees που κατασκευάζονται, καταλήγοντας στην ανάγκη υπολογισμού λιγότερων hashes. Επίσης, είναι φανερό πως η ανάγνωση του περιεχομένου όλων των μπλοκ εισάγει μια σημαντική καθυστέρηση στην διαδικασία. Αντιθέτως, το πλήθος των Merkle proofs δεν παρουσιάζει ιδιαίτερο χρονικό αντίκτυπο, κα-

θώς η κατασκευή του Merkle tree πραγματοποιείται μόλις μία φορά ανά μπλοκ, με όλες τις απαιτούμενες αποδείξεις να εξάγονται στη συνέχεια από αυτό.

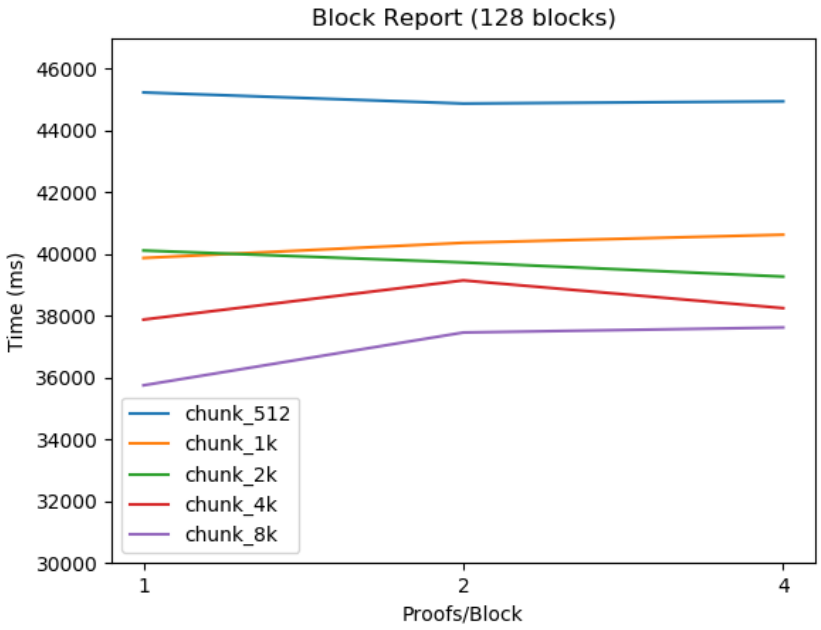


Σχήμα 7.3: *Διάγραμμα διάρκειας κατασκευής των Merkle proofs κατά το blockreport*

Αντίστοιχα, στο Σχήμα 7.4 παρουσιάζεται το συνολικό διάστημα όπου ο DataNode παραμένει μη αποκρίσιμος κατά το blockreport. Δηλαδή, από την εκκίνηση ενός blockreport έως τη λήψη της απάντησης από τον NameNode κατόπιν αναφοράς της λίστας με τα μπλοκ που ο DataNode διαθέτει. Η διαδικασία αυτή αποτελείται από τα διαδοχικά βήματα δημιουργίας και ανάκτησης του καινούργιου σπόρου, της κατασκευής των Merkle proofs και της επικοινωνίας του DataNode με τον NameNode.

Το στάδιο δημιουργίας και ανάκτησης του σπόρου παρατηρήθηκε πως εισάγει μια επιπλέον καθυστέρηση, καθώς η συναλλαγή ανανέωσης του σπόρου πρέπει να συμπεριληφθεί στο blockchain. Η καθυστέρηση αυτή εξαρτάται από τη συχνότητα εισαγωγής καινούργιων μπλοκ στην αλυσίδας, ενώ στην περίπτωσή μας παρατηρήθηκε να προσεγγίζει κατά μέσο όρο τα 15 δευτερόλεπτα. Λαμβάνοντας υπόψη πως χωρίς της τροποποιήσεις που αναφέρουμε, για το ίδιο πλήθος δεδομένων και μέγεθος block, η μέση διάρκεια ενός blockreport του DataNode δεν ξεπερνούσε τα 25 msec, γίνεται φανερή η επιβάρυνση ως προς τη διαθεσιμότητα του κόμβου.

Όμως για το προαναφερθέν πλήθος δεδομένων και την προεπιλεγμένη συχνότητα εκτέλεσης του blockreport που είναι μία φορά ανά έξι ώρες, το χρονικό διάστημα που ο DataNode δεν είναι αποκρίσιμος παραμένει σχετικά μικρό, και αφού οι επιμέρους DataNodes δεν υποχρεούνται να εκτελούν ταυτόχρονα το blockreport, τελικά με μεγάλη πιθανότητα θα βρίσκεται ανά πάσα στιγμή κάποιος διαθέσιμος DataNode για τη διεκπεραίωση των αιτημάτων των χρηστών. Η καθυστέρηση αυτή όμως αυξάνεται γραμμικά με τον όγκο των δεδομένων που πρέπει να αποδείξει ο DataNode, περιορίζοντας τελικά την κλιμακωσιμότητα του συστήματος. Το τελευταίο είναι εφικτό να βελτιωθεί με ανάθεση περισσότερων πυρήνων



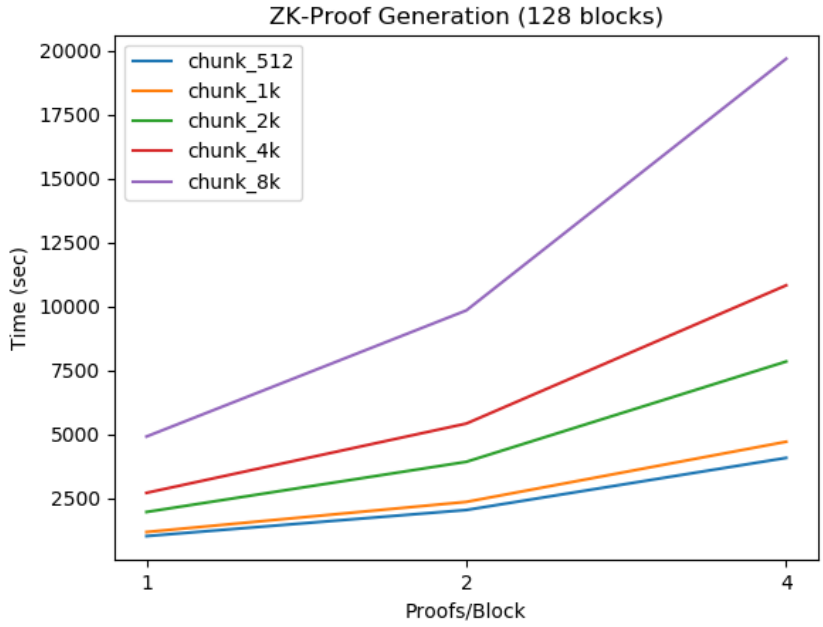
Σχήμα 7.4: *Διάγραμμα διάρκειας μη αποκρίσιμης φάσης του DataNode κατά το blockreport*

στην κατασκευή των Merkle trees, και τη χρήση RAID συστοιχίας ως μια προσιτή λύση για ταχύτερη, ταυτόχρονη πρόσβαση και ανάκτηση των επιμέρους μπλοκ από το τοπικό σύστημα αρχείων.

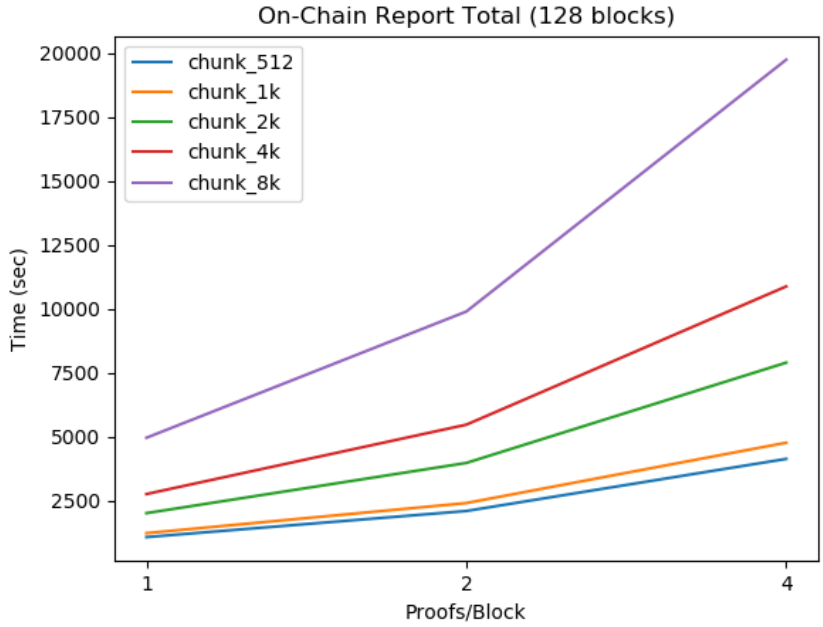
Η χρονική διάρκεια κατασκευής των zk-SNARK αποδείξεων παρουσιάζονται στο Σχήμα 7.5. Αν και δεν περιορίζεται η φυσιολογική λειτουργία του συστήματος, είναι φανερό πως απαιτείται σημαντικά περισσότερος χρόνος για την κατασκευή τους. Το χρονικό διάστημα κατασκευής μιας zk-SNARK απόδειξης για ένα Merkle proof καθορίζεται κυρίως από το πλήθος των hashes που αποδεικνύονται, τόσο των ενδιάμεσων του μονοπατιού όσο και του περιεχομένου του εν λόγω chunk. Πιο συγκεκριμένα, η Blake2s χωρίζει την αυθαίρετου μήκους είσοδο σε επιμέρους τμήματα μήκους 512bits, στα οποία και λειτουργεί διαδοχικά. Δηλαδή το κόστος υπολογισμού της εξαρτάται από το μήκος της εισόδου. Έτσι είναι αναμενόμενο να παρατηρούμε σημαντικά αυξημένο χρόνο κατασκευής των αποδείξεων για μεγαλύτερο chunk size. Όμως για σταθερό μέγεθος μπλοκ, η μείωση του chunk size δε συνεπάγεται γραμμική μείωση του χρόνου κατασκευής των αποδείξεων, καθώς αυξάνεται το ύψος του Merkle tree, οπότε και το μήκος του μονοπατιού που αποδεικνύεται.

Τέλος, παρατίθεται το διάγραμμα του συνολικού χρονικού διαστήματος του blockreport, από τη στιγμή της εκκίνησης έως την υποβολή και τις τελευταίας zk-SNARK απόδειξης (Σχήμα 7.6).

Αν και από τα παραπάνω καταλήγουμε πως μικρότερο chunk size οδηγεί σε ταχύτερη παραγωγή των zk-SNARK αποδείξεων, αξίζει να ερευνήσουμε το ποσοστό των δεδομένων που τελικά ελέγχεται σε κάθε περίπτωση, καθώς αυτό κρίνει και τη βεβαιότητα του χρήστη ως προς την ακεραιότητα των δεδομένων του. Διατηρώντας, σταθερό το πλήθος των δεδομένων που αποδεικνύονται στο 1 MByte από το σύνολο των 8 GByte που διαθέτει ο DataNode (δηλαδή μόλις το 0.0122% του συνόλου των δεδομένων), παρουσιάζουμε το χρόνο κατασκευής των



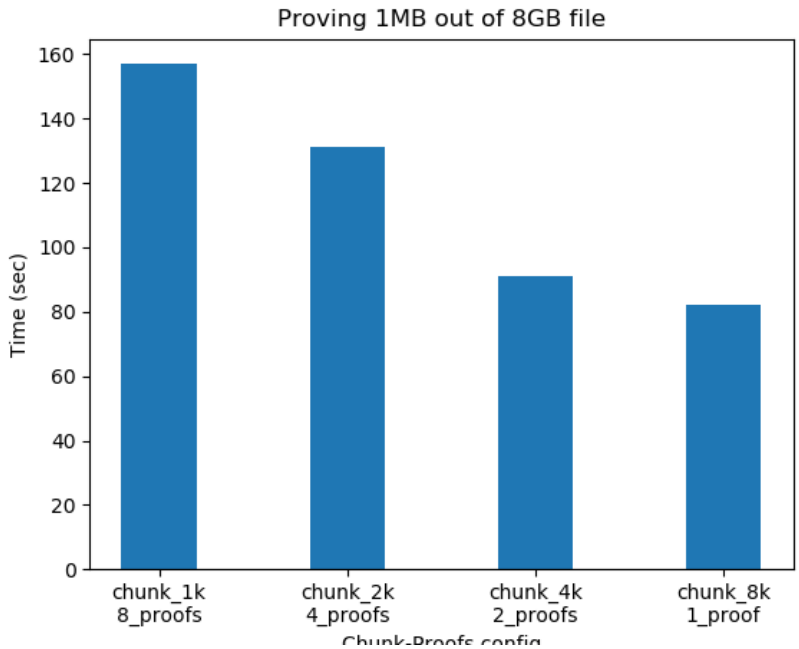
Σχήμα 7.5: *Διάγραμμα διάρκειας κατασκευής των zk-SNARK αποδείξεων*



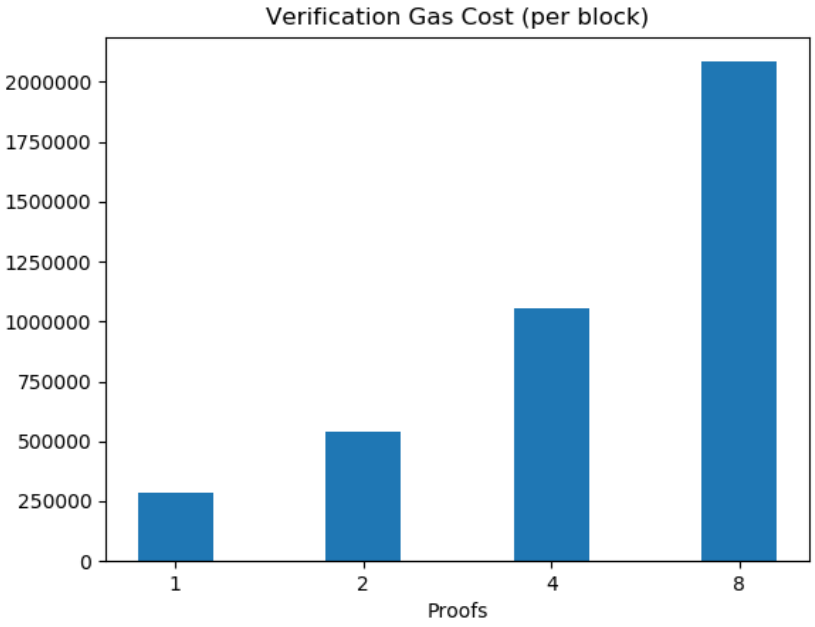
Σχήμα 7.6: *Διάγραμμα συνολικής διάρκειας blockreport*

zk-SNARK αποδείξεων για τα διαφορετικές παραμετροποιήσεις του chunk size, ρυθμίζοντας κατάλληλα των πλήθος των αποδείξεων που υποβάλλονται ανά μπλοκ (Σχήμα 7.7).

Είναι λοιπόν φανερό πως η αύξηση του chunk size είναι αισθητά αποδοτικότερη μέθοδος κάλυψης περισσότερου πλήθους δεδομένων σε αντίθεση με την αύξηση των αποδείξεων. Η θέση αυτή ενισχύεται τελικά και από το κόστος επαλήθευσης των αποδείξεων στο ethereum blockchain (Σχήμα 7.8), το οποίο εξαρτάται αποκλειστικά από το πλήθος τους.



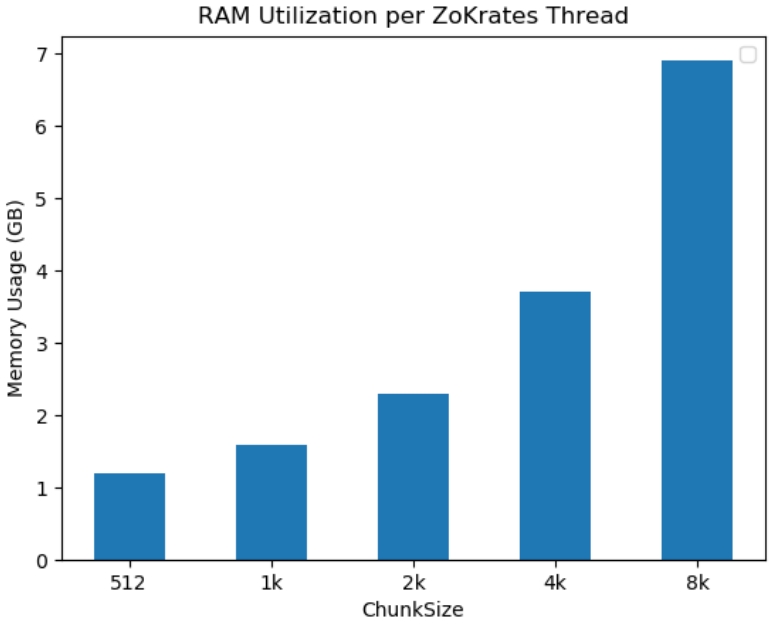
Σχήμα 7.7: *Διάγραμμα κατασκευής zk-SNARK αποδείξεων για σταθερό πλήθος δεδομένων*



Σχήμα 7.8: *Διάγραμμα κόστους επαλήθευσης των αποδείξεων ενός μπλοκ (σε gas)*

Όμως η αύξηση του chunk size οδηγεί σε σημαντικά αυξημένες απαιτήσεις σε μνήμη RAM, κάτι που περιορίζεται αποκλειστικά από το υλισμικό των DataNodes (Σχήμα 7.9). Δηλαδή, η παραμετροποίηση του συστήματος εξαρτάται από τις προδιαγραφές των υπολογιστικών συσκευών που το συγκροτούν, ενώ παρουσιάζει δυσκολίες κατά την κλιμάκωση ως προς το πλήθος των προς απόδειξη δεδομένων. Συμπεραίνουμε έτσι πως η βέλτιστη παραμετροποίηση του συστήματος με γνώμονα την μέγιστη κάλυψη δεδομένων κατά τον έλεγχο,

στο ελάχιστο δυνατό χρονικό διάστημα, ευνοείται από την επιλογή του μεγαλύτερου εφικτού chunk size που υποστηρίζεται από τη διαθέσιμη RAM των DataNodes.



Σχήμα 7.9: *Διάγραμμα απαιτήσεων σε RAM* ***για κάθε εκτελούμενο νήμα*** *κατά την κατασκευή των zk-SNARK αποδείξεων*

Είναι λοιπόν φανερό πως αν και τα zk-SNARKs αποτελούν ένα ισχυρό αποδεικτικό εργαλείο που μπορεί να εγγυηθεί μηδενική γνώση, διατηρώντας την ιδιωτικότητα των προς απόδειξη ισχυρισμών, προς το παρόν η χρήση τους σε μεγάλης κλίμακας εφαρμογές είναι περιορισμένη λόγω των σημαντικά υψηλών υπολογιστικών απαιτήσεων που τα συνοδεύουν.

Παραρτήματα

Παράρτημα Α΄

Κώδικας έξυπνου συμβολαίου

```
// SPDX-License-Identifier: UNLICENCED
pragma solidity >=0.4.22 <0.9.0;

import "./verifier.sol";


contract Data {

    // Need to be instantiated on deployment
    uint constant num_chall = 1;     // challenges/block
    uint constant num_chunks = 65536;    // chunks/block

    struct bp_struct {
        mapping (uint=>bytes32) roots;
        mapping (address=>uint) seeds;
    }

    mapping (bytes32=>bp_struct) bp_data;
    Verifier verifier;

    event BlockReport(bytes32 indexed blockpool, address datanode, uint time,  uint blockId, bool corrupt
         );


    constructor () {
        verifier = new Verifier();
    }


    function add_digest(bytes32 _bp_id, uint _block_id, bytes32 _root) external {
        bp_data[_bp_id].roots[_block_id] = _root;
    }

    function create_seed(bytes32 _bp_id, uint randomness) external {
        uint seed = block.timestamp;
        // pack both randomness and timestamp in 256bit uint (timestamp on LSBs)
        assembly{
            let shifted_rand := mul(randomness, 0x100000000000000000000000000000000)
            seed := or(seed, shifted_rand)
```

```
39          }
40          bp_data[_bp_id].seeds[tx.origin] = seed;
41      }
42
43
44      function peek_seed(bytes32 _bp_id) external view returns(bytes memory) {
45          return abi.encodePacked(bp_data[_bp_id].seeds[tx.origin], _bp_id, tx.origin);
46      }
47
48      // generate a series of _count random numbers using a seed
49      function gen_challenges(bytes memory _seed, uint _block_id) internal pure returns (uint[num_chall]
            memory) {
50          uint[num_chall] memory challenges;
51          uint tmp = uint(keccak256(abi.encodePacked(_block_id, _seed)));
52          challenges[0] = tmp % num_chunks;
53          for(uint i = 1; i < num_chall; i++) {
54              tmp = uint(keccak256(abi.encodePacked(tmp, _block_id)));
55              challenges[i] = tmp % num_chunks;
56          }
57          return challenges;
58      }
59
60
61      function get_input_vector(bytes32 root) internal pure returns(uint[5] memory) {
62          uint[5] memory input;
63          // implemented with inline assembly for gas efficiency
64          assembly {
65              mstore(add(input, 0x40), and(root, 0xffffffffffffffffffffffffffffffff))
66              mstore(add(input, 0x20), and(div(root, 0x100000000000000000000000000000000), 0
                    xffffffffffffffffffffffffffffffff))
67          }
68          return input;
69      }
70
71
72      function verify(bytes32 _bp_id, uint _block_id, uint[] memory numbers) external{
73          // first get a reference to the blockpool storage
74          bp_struct storage bp_pointer = bp_data[_bp_id];
75          // grab the seed and generate challenges
76          uint seed = bp_pointer.seeds[tx.origin];
77          uint time;
78          assembly{
79              time := and(seed, 0xffffffffffffffffffffffffffffffff)
80          }
81          uint[num_chall] memory challenges = gen_challenges(abi.encodePacked(seed, _bp_id, tx.origin),
                _block_id);
82          // get the input vector for the verification
83          uint[5] memory input = get_input_vector(bp_pointer.roots[_block_id]);
84          input[3] = challenges[0];
85          input[4] = 1;
86          for(uint i = 0; i < num_chall; i++) {
87              input[0] = challenges[i];
```

```
88              Verifier.Proof memory proof;
89              proof.a = Pairing.G1Point(numbers[i*8], numbers[i*8+1]);
90              proof.b = Pairing.G2Point([numbers[i*8+2], numbers[i*8+3]], [numbers[i*8+4], numbers[i*8+5]])
                    ;
91              proof.c = Pairing.G1Point(numbers[i*8+6], numbers[i*8+7]);
92              if(!verifier.verifyTx(proof, input)) {
93                  emit BlockReport(_bp_id, tx.origin, time, _block_id, true);
94                  return;
95              }
96          }
97          emit BlockReport(_bp_id, tx.origin, time, _block_id, false);
98      }
99
100 }
```

Βιβλιογραφία

[1] *Introduction to Distributed File Systems*. http://www.cse.chalmers.se/~tsigas/Courses/DCDSeminar/Files/afs_report.pdf.

[2] *File Systems*. https://en.wikipedia.org/wiki/File_system.

[3] G. Coulouris, J. Dollimore, T. Kindberg και G. Blair. *ΚΑΤΑΝΕΜΗΜΕΝΑ ΣΥΣΤΗΜΑΤΑ ΑΡΧΕΣ ΚΑΙ ΣΧΕΔΙΑΣΗ*. DA VINCI, 2020.

[4] *What is a distributed file system?* https://www.scality.com/topics/what-is-a-distributed-file-system.

[5] *HDFS Architecture Guide*. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.

[6] *What is a Block Scanner in HDFS?* https://data-flair.training/forums/topic/what-is-a-block-scanner-in-hdfs/.

[7] *HDFS Federation*. https://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-hdfs/Federation.html.

[8] *Merkle Tree in Blockchain: What is it, How does it work and Benefits*. https://www.simplilearn.com/tutorials/blockchain-tutorial/merkle-tree-in-blockchain.

[9] *What are cryptographic hash functions?* https://www.synopsys.com/blogs/software-security/cryptographic-hash-functions/.

[10] *Merkle proofs Explained*. https://medium.com/crypto-0-nite/merkle-proofs-explained-6dd429623dc5.

[11] *Blockchain Explained*. https://www.investopedia.com/terms/b/blockchain.asp.

[12] Alan T. Sherman, Farid Javani, Haibin Zhang και Enis Golaszewski. *On the Origins and Variations of Blockchain Technologies*. *IEEE Security Privacy*, 17(1):72–77, 2019.

[13] Stuart Haber και W. Scott Stornetta. *How to Time-stamp a Digital Document*. *Journal of Cryptology*, 3:99–111, 1991.

[14] Dave Bayer, W. Scott Stornetta και Stuart Haber. *Improving the Efficiency and Reliability of Digital Time-Stamping*. *Sequences II: Methods in Communication, Security and Computer Science*, σελίδες 329–334. Springer-Verlag, 1993.

[15] Zibin Zheng, Shaoan Xie, Hong Ning Dai, Xiangping Chen και Huaimin Wang. *An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends*. σελίδες 557–564, 2017.

[16] Don Johnson, Alfred Menezes και Scott A. Vanstone. *The Elliptic Curve Digital Signature Algorithm (ECDSA)*. *Int. J. Inf. Sec.*, 1(1):36–63, 2001.

[17] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker και Stefan Savage. *A fistful of bitcoins: Characterizing payments among men with no names*. *IMC 2013 - Proceedings of the 13th ACM Internet Measurement Conference*, Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC, σελίδες 127–139, 2013. 13th ACM Internet Measurement Conference, IMC 2013 ; Conference date: 23-10-2013 Through 25-10-2013.

[18] Jaume Barceló. *User Privacy in the Public Bitcoin Blockchain*. 2014.

[19] Alex Biryukov, Dmitry Khovratovich και Ivan Pustogarov. *Deanonymisation of Clients in Bitcoin P2P Network*. *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014.

[20] Leslie Lamport, Robert Shostak και Marshall Pease. *The Byzantine generals problem*. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[21] *What is Ethereum?* https://aws.amazon.com/blockchain/what-is-ethereum/.

[22] *Intro to Ethereum*. https://ethereum.org/en/developers/docs/intro-to-ethereum.

[23] Μ. Burmester και Σ. Γκρίτζαλης και Σ. Κάτσικας και Β. Χρυσικόπουλος. *Σύγχρονη Κρυπτογραφία: Θεωρία και Εφαρμογές*. Παπασωτηρίου, 1η έκδοση, 2011.

[24] *Demonstrate how Zero-Knowledge Proofs work without using math*. https://www.linkedin.com/pulse/demonstrate-how-zero-knowledge-proofs-work-without-using-chalkias.

[25] *What are zk-SNARKs?* https://z.cash/technology/zksnarks/.

[26] *What are zk-SNARKs?* https://minaprotocol.com/blog/what-are-zk-snarks.

[27] *Introduction to zk-SNARKs*. https://consensys.net/blog/developers/introduction-to-zk-snarks/.

[28] *Quadratic Arithmetic Programs: from Zero to Hero*. https://medium.com/@VitalikButerin/quadratic-arithmetic-programs-from-zero-to-hero-f6d558cea649.

[29] Damgard και Ivan. *Towards Practical Public Key Systems Secure Against Chosen Ciphertext attacks*. *Advances in Cryptology — CRYPTO '91* Feigenbaum και Joan, επιμελητές, σελίδες 445–456, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[30] Maksym Petkus. *Why and How zk-SNARK Works*. *CoRR*, αβσ/1906.07221, 2019.

[31] *ZoKrates*. https://zokrates.github.io/.

[32] *How to Generate a Groth16 Proof for Forgery*. https://medium.com/ppio/how-to-generate-a-groth16-proof-for-forgery-9f857b0dcafd.

[33] *BLAKE2 – fast secure hashing*. https://www.blake2.net/.

[34] *Go Ethereum*. https://geth.ethereum.org/.

[35] *OkHttp 4.x Change Log*. https://square.github.io/okhttp/changelogs/changelog_4x/#version-450-rc1.