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

Αποκεντρωμένη Αποθήκευση με χρήση **Smart Contracts**

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΜΙΧΑΛΟΠΟΥΛΟΣ ΦΩΤΙΟΣ

Επιβλέπων : Νεκτάριος Κοζύρης
Καθηγητής ΕΜΠ

Εργαστήριο Υπολογιστικών Συστημάτων

Αθήνα, Ιούλιος 2021

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

Αποκεντρωμένη Αποθήκευση με χρήση **Smart Contracts**

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΜΙΧΑΛΟΠΟΥΛΟΣ ΦΩΤΙΟΣ

Επιβλέπων : Νεκτάριος Κοζύρης
Καθηγητής ΕΜΠ

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 23η Ιουλίου 2021.

.....	.....	.....
Νεκτάριος Κοζύρης	Διονύσιος Πνευματικάτος	Γεώργιος Γκούμας
Καθηγητής ΕΜΠ	Καθηγητής ΕΜΠ	Αν. Καθηγητής ΕΜΠ

Εργαστήριο Υπολογιστικών Συστημάτων
Αθήνα, Ιούλιος 2021

. . . . . .

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

Ευχαριστίες

Αρχικά, θα ήθελα να ευχαριστήσω τον Καθηγητή Νεκτάριο Κοζύρη, ο οποίος μέσω των διαλέξεών του μου μετέδωσε το ενδιαφέρον του για τον κόσμο των υπολογιστικών συστημάτων.

Επίσης, ένα μεγάλο ευχαριστώ πρός τη Διδάκτορα Κατερίνα Δόκα, καθώς, χωρίς τη βοήθεια και τις επιστημονικές προσεγγίσεις της, η διπλωματική αυτή, δε θα μπορούσε να έχει υλοποιηθεί.

Οφείλω επίσης ένα μεγάλο ευχαριστώ στο φίλο και συνάδελφό μου Χρήστο Κατσακιώρη, ο οποίος με το πάθος που τον διακατέχει για την επιστήμη των υπολογιστών, με ωθούσε συνεχώς στην εξέλιξή μου ως μηχανικός αλλά και ως ατόμο. Τεράστιο ευχαριστώ οφείλω και στην οικογένεια του, για την αμέριστη στήριξη τους σε όλο το διάστημα της εκπόνησης της διπλωματικής.

Τέλος, θα ήθελα να ευχαριστήσω από τα βάθη της καρδιάς μου, τον αδερφό μου , Δημήτρη Μιχαλόπουλο για όλη την αγάπη και τη στήριξη που μου παρείχε καθόλη τη διάρκεια της διπλωματικής μου και γενικότερα όλων των χρόνων της φοίτησής μου.

Φώτιος Μιχαλόπουλος
Ιούλιος 2021

Περίληψη

Τα τελευταία χρόνια , το cloud computing τείνει να γίνει το κυρίαρχο μέσο ανάπτυξης εφαρμογών. Μερικοί από τους λόγους που οδήγησαν σε αυτό, είναι ότι το cloud computing, έρχεται να αντικαταστήσει το υψηλό κόστος συντήρησης του hardware, καθώς τώρα πια κανείς μπορεί ακόμα και να νοικιάσει απομακρυσμένα hardware. Ένα άλλο πλεονέκτημα άξιο αναφοράς, είναι η κλιμακωσιμότητα την οποία προσφέρει το cloud computing. Παραδείγματος χάριν, μια εφαρμογή η οποία θα χρειαστεί μια αναβάθμιση, η αναβάθμιση αυτή, γίνεται εύκολα υλοποιήσιμη εαν το hardware δεν είναι άρρηκτα συνδεδεμένο με αυτή, δηλαδή είναι υλοποιημένη με τη βοήθεια των cloud services, τα οποία δύναται να τρέχουν σε απομακρυσμένα μεταξύ τους μηχανήματα. Το γεγονός αυτό, αποτέλεσε το έναυσμα για πολλές εταιρείες να οδηγηθούν στην αντικατάσταση των τοπικών υποδομών τους , οι οποίες είναι δύσκολα διαχειρίσιμες απο απομακρυσμένες υπηρεσίες, τόσο για την αποθήκευση, τον υπολογισμό δεδομένων και γενικότερα για την υλοποίηση εφαρμογών, επενδύοντας μεγάλα ποσά. Βέβαια, λόγω της οικονομικής δύναμης κάποιων λίγων τον αριθμό εταιρειών, το μεγαλύτερο μέρος του cloud computing συγκεντρώνεται σε αυτές. Αυτό με τη σειρά του, οδηγεί στη δημιουργία επιπλέον προβλημάτων. Ενδεικτικά αναφέρουμε ζητήματα αξιοπιστίας, καθώς λίγες τον αριθμό εταιρείες διαχειρίζονται μεγάλο όγκο δεδομένων, τα οποία είναι σε θέση να τα εκμεταλλευτούν όπως εκείνες επιθυμούν. Επίσης, οι εταιρείες αυτές λόγω του μονοπωλείου τους στο cloud computing, καθορίζουν οι ίδιες τις τιμές που απαιτούν για την προσφορά υπηρεσιών, όπως είναι η αποθήκευση δεδομένων.Η τεχνολογία του blockchain αποτελεί μια ιδανική εναλλακτική για την ανάπτυξη αποκεντρωμένων εφαρμογών. Πιο συγκεκριμένα, το ethereum χρησιμοποιεί τη τεχνολογία του blockchain και μέ τη βοήθεια των smart contracts, δημιουργεί ευνοικές συνθήκες για την ανάπτυξη αποκεντρωμένων εφαρμογών.

Στόχος μας στην παρούσα διπλωματική, ήταν να δημιουργήσουμε μια αποκεντρωμένη αγορά για τη αποθήκευση δεδομένων, χρησιμοποιώντας τη τεχνολογία του ethereum και πιο συγκεκριμένα τα smart contracts. Με αυτό το τρόπο οι χρήστες οι οποίοι θέλουν να νοικιάσουν αποθηκευτικό χώρο , θα μπορούν να το πετύχουν σε προσιτές τιμές, ενώ οι χρήστες οι οποίοι έχουν περισσευούμενο αποθηκευτικό χώρο, θα μπορούν να κερδίζουν χρήματα προσφέροντάς τον. Εμείς, χρησιμοποιήσαμε τη τεχνολογία του blockchain σα μεσάζοντα ο οποίος διατηρεί και επικυρώνει ενέργειες εκτός της αλυσίδας (off-chain operations).

Λέξεις-Κλειδιά

blockchain, dapp, Ethereum, Smart Contracts, merkle proof, proof of storage, merkle tree, cloud-computing, solomon-reed, redundancy

Abstract

In recent years, Cloud Computing has become the de facto standard platform for application development and deployment, due to the various advantages that it offers. These include the low costs associated with hardware infrastructure (since the latter is being rented and located off premises), as well as its vast capacity which allows it to scale indefinetely much, theoretically. The prevalence of Cloud Computing paradigms and their adoption by a large portion of the industry, themselves translate to heavy investments in few Cloud Service Provider companies, which nowadays control a huge share of the global market.The gradual development of such power poles is deemed problematic, since it is prone to trust an reliability issues : a handful of companies entrusted with storing and manipulating huge amounts of data, often in an opaque manner, raises concerns about the possibility of their non-consensual exploitation. Moreover, the pricing of Cloud services (such as storage) is affected by their centralization in a manner that is harmful to the end users: the existing oligopoly and the vendor lock-in enables providers to effectively set their prices almost unilaterally.

In this thesis, we attempt to create the infrastructure for a decentralized storage market, by leveraging Ethereum and the principles of Blockchain technology that underpin it. Blockchain constitutes an ideal alternative to the Cloud for developing decentralized services and applications. Ethereum revolutionizes Blockchain by establishing the notion and the mechanics of Smart Contracts, which we make great use of: in our system, users who need storage are enabled to rent it in affordable prices, while users who have excess storage space are allowed to earn money by providing it. To ameliorate the reliability and trust issues entailed by its decentralized nature, our system incorporates a mechanism that allows users to verify that providers actually store the data, by leveraging the distributed ledger to keep track and validate off-chain operations.

Key-Words

blockchain, dapp, Ethereum, Smart Contracts, merkle proof, proof of storage, merkle tree, cloud-computing, solomon-reed, redundancy

Contents

List of Figures

Chapter 1

Ανάλυση προβλήματος και κίνητρα

1.1 Πρόβλημα

Τα τελευταία χρόνια, το cloud computing χρησιμοποιείται κατα κόρον σε πολλους τομείς της βιομηχανίας. Εκτός αυτού, το cloud computing χρησιμοποιείται και για αποθήκευση δεδομένων (data storage)[5]. Πράγματι, οι έρευνες δείχνουν ότι το 90% των ήδη υπάρχοντων εταιρειών αποθηκεύουν τα δεδομένα τους στο cloud. Αξίζει να τονίσουμε ότι η διαχείριση του cloud storage αποτελεί μονοπώλιο λίγων εταιρείων οι οποίες απολαμβάνουν τα κέρδη απο αυτό το μονοπώλειο. Αναλυτικότερα, δύο εταιρείες απολαμβάνουν το 49% της συνολικής αγοράς του cloud[6]. Το γεγονός αυτό, εγείρει ερωτήματα ως προς την αξιοπιστία καθώς και το τρόπο διαχείρισης των δεδομένων αυτών καθε αυτών απο τις εταιρείες αυτές. Στην προκειμένη περίπτωση οι εταιρείες αυτές αποτελούν το μοναδικό μέσο στο οποίο εμπιστεύεται η εκάστοτε εταιρεία/χρήστης τα δεδομένα του. Τα τελευταία χρόνια οι εταιρείες αυτές αποτελούν στόχο για επιθέσεις τρίτων. Οι επιθέσεις αυτές βρίσκουν πρόσφορο έδαφος καθώς εκμεταλλεύονται κενά ασφαλείας (security breaches), τα οποία έχουν οι συγκεκριμένες εταιρείες.

Έρευνες δείχνουν, ότι το 70% των εταιρείων που ασχολούνται με την αποθήκευση δεδομένων έχουν πέσει θύμα επίθεσης απο χάκερ με αποτέλεσμα τη διαρροή δεδομένων (data leaks) [7]. Το αποτέλεσμα είναι να αμφισβητείται η διασφάλιση τόσο της ιδιωτικότητας (data privacy). Συν τοις άλλοις, πολλές φορές τα δεδομένα των χρηστών χρησιμοποιούνται για λόγους εκμετάλλευσης ή δίνονται σε τρίτους όπως πχ διαφημιστικές εταιρείες. Εν ολίγοις, το γεγονός ότι τα δεδομένα αποθηκεύονται σε λίγους τον αριθμό παρόχους οδηγεί στο να αποτυγχάνει η αποκέντρωση (decentralization), η οποία αποτελεί και ένα βασικό σκοπό που επιτελεί η ύπαρξη του cloud. Επιπλέον, οι χρήστες θεωρούν δεδομένο ότι τα κριτήρια αυτών των λίγων παρόχων δεν πληρούν σκοπιμότητες. Ένας άλλος αποτρεπτικός παράγοντας του μονοπωλίου αυτού, είναι ότι οι πάροχοι είναι οι μόνοι οι οποίοι καθορίζουν τις τιμές καθότι δεν υπάρχει ανταγωνισμός. Άλλος ένας αποτρεπτικός παράγοντας είναι το

γεγονός, ότι τα δεδομένα συνήθως αποθηκεύονται ως έχουν απο τους παρόχους δεδομένων (data centers), χωρίς να υφίστανται κάποια κρυπτογράφηση (data encryption), κάτι που κάνει τα δεδομένα των χρηστών να είναι εύκολα προσβάσιμα.[13]

1.2 Εναλλακτική Προσέγγιση

Μια ενδιαφέρουσα λύση , η οποία μπορεί να δώσει μια άλλη προσέγγιση στο ζήτημα είναι το blockchain. Όπως αναφέρεται και στην περίληψη της διπλωματικής, στόχος της παρούσης, είναι η δημιουργία μιας αποκεντρωμένης αγοράς για την αποθήκευση δεδομένων. Μέσω της αγοράς αυτής, οι χρήστες οι οποίοι θέλου να νοικιάσουν αποθηκευτικό χώρο(renter), θα είναι σε θέση να το επιτύχουν σε προσιτές τιμές. Αντίστοιχα, οι χρήστες οι οποίοι έχουν περισσευοέμενο αποθηκευτικό χώρο, θα είναι σε θέση να κερδίζουν χρήματα προσφέροντάς τον. Γία τη δημιουργία αυτής της αποκεντρωμένης αγοράς, χρησιμοποιήσαμε την τεχνολογία του blockchain. Στο σημείο αυτό αξίζει να τονιστεί, ότι η το blockchain, δε μπορεί να χρησιμοποιηθεί αυτούσιο ως αποθηκευτικό μέσο, για λόγους οιυ θα εξηγήσουμε παρακάτω. Για τον ανωτέρω λόγο, το blockchain, χρησιμοποιείται ως μεσάζοντας ο οποίος έχει ώς κύρια λειτουργία να διατηρεί και να επικυρώνει ενέργειες εκτός της αλυσίδας (off-chain operations).Για την υλοποίηση του συστήματος μας κάναμε χρήση του Ethereum , το οποίο είναι ένα κρυπτονόμισμα που χρησιμοποιεί τη τεχνολογία του blockchain και πρόκειται να περιγράψουμε στο επόμενο υποκεφάλαιο. Αρχικά, θα αναφέρουμε λίγα πράγματα για την τεχνολογία του blockchain.
Η τεχνολογία του blockchain αναφέρθηκε για πρώτη φορά το 2008 από τον Nakamoto[8] .

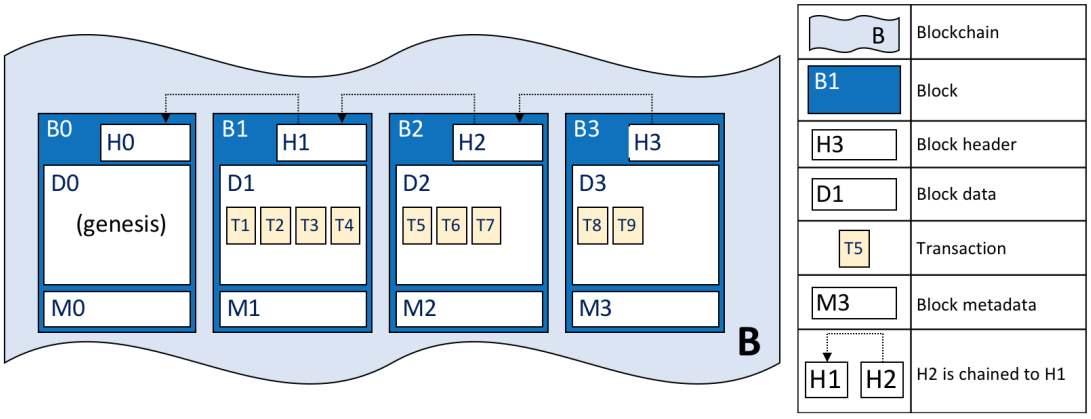

FIGURE 1.1: Blockchain

Ο Nakamoto περιέγραψε ένα κρυπτονόμισμα το bitcoin το οποίο στηρίζεται στην τεχνολογία του blockchain. Το blockchain ουσιαστικά είναι μια αμετάβλητη (immutable) βάση δεδομένων, η οποία αποτελείται από κομμάτια (blocks) συνδεδεμένων μεταξύ τους μέσω δεικτών κατακερματισμού (hash pointers), στην περίπτωση του bitcoin τα blocks αυτά αποτελούνται από οικονομικές συναλλαγές (cryptocurrency transactions) μεταξύ διαφορετικών χρηστών. Τα κομμάτια αυτά των cryptocurrency transactions γίνονται mine κάθε περίπου δεκαπέντε λεπτά απο κάποιο κόμβο (node) ή από ομάδα κόμβων (pool of nodes) μέσω του proof of work, που πολύ απλά λέγοντας αποτελεί έναν brute force αλγόριθμο για την εύρεση ενός hash το οποίο έχει ορισμένες προδιαγραφές. Η διαδικασία αυτή είναι διαφανής καθώς μετέπειτα για να μπεί πράγματι το μπλοκ αυτό στην μόνιμη αλυσίδα θα πρέπει

να επικυρωθεί απο τους υπόλοιπους κόμβους ότι το αποτέλεσμα πληροί τις προδιαγραφές. Έτσι επικρατεί η διαφάνεια καθώς και η έννοια του proof of work είναι ότι ο κόμβος ή οι κόμβοι που κατέληξαν στο αποτέλεσμα κατανάλωσαν υπολογιστικούς πόρους με ό,τι αυτό συνεπάγεται (κόστος σε ρεύμα), χωρίς να χρειάζεται η ύπαρξη κάποιας κεντρικής αρχής. Το τελευταίο μας οδηγεί στο συμπέρασμα ότι το blockchain είναι όντως πλήρως αποκεντρωμένο (fully decentralized). Συνοπτικά, θα μπορούσαμε να πούμε ότι το blockchain είναι μια κατανεμημένη αλυσίδα (distributed ledger) που αποτελείται απο κρυπτο συναλλαγές (cryptocurrency transactions), οι οποίες αυτές συναλλαγές είναι ταξινομημένες σε χρονολογική σειρά και αμετάβλητες. Το bitcoin αποτελείται απο accounts, όπου κάθε account έχει ενα public Key το οποίο είναι φανερό και ενα private Key το οποίο το γνωρίζει μόνο το ίδιο το account.

1.2.1 Ethereum

Το bitcoin, αποτέλεσε τη βάση για την εκμετάλλευση της τεχνολογίας αυτής του blockchain. Σε αυτό στηρίχτηκε το ethereum [12]. Το κύριο χαρακτηριστικό του είναι ότι είναι state-ful , δηλαδή, η προσθήκη ενός block το οδηγεί σε μια και μοναδική κατάσταση (state). Ειναι χαρακτηριστική η φράση που αναφέρεται στο τεχνικό έγγραφο του ethereum (yellowpaper).[14]

> Ethereum is a transaction-based "state" machine; a technology on which all transaction-based state machine concepts may be built.

Κάθε κατάσταση (state) περιγράφεται από τα δύο είδη λογαριασμών (accounts) που περιέχει

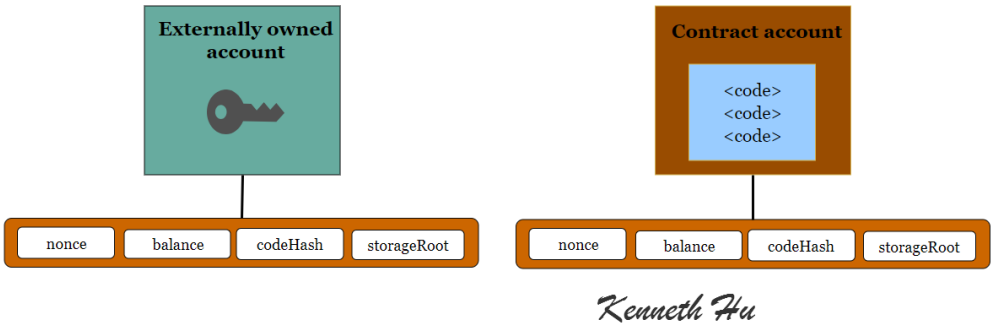

FIGURE 1.2: Ethererum Accounts

Είναι φανερό και από το σχήμα, ότι και τα δύο είδη λογαριασμών περιέχουν τον ίδιο αριθμό πεδίων. Η μόνη διαφορά έγκειται στο τύπο account (contract accounts), το οποία περιέχει το bytecode ,το οποίο είναι αποτέλεσμα compile μιας high level γλώσσας, συνήθως χρησιμοποιείται μια εκ των solidity ή serpent . Η solidity θυμίζει τη γλώσσα προγραμματισμού javascript, ενώ η serpent θυμίζει την python. Παραθέτουμε ενδεικτικά μια εικόνα ενός smart contract γραμμένου σε solidity.

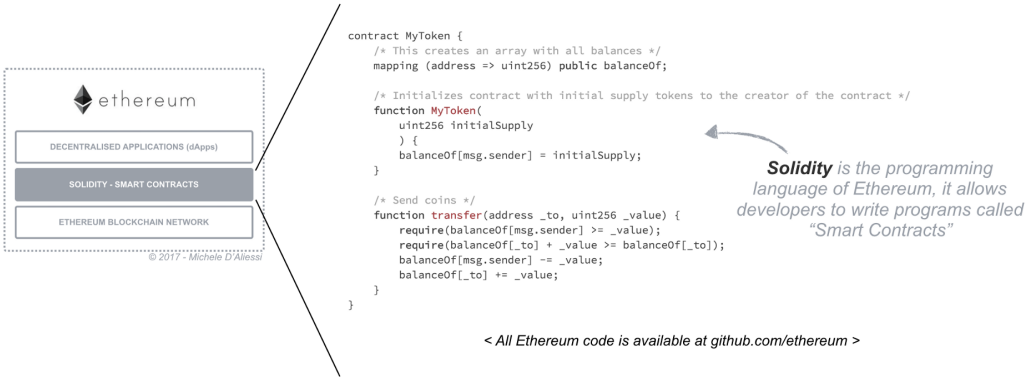

FIGURE 1.3: Solidity Example

Ο κώδικας αυτός αποτελεί τα smart contracts, τα οποία είναι κομμάτια Turing complete κώδικα, που εκτελούνται αύτοματα με κατανεμημένο τρόπο (in a distributed manner).Η

δημιουργία ένος contract είναι ένα transaction, όπου το receiver address είναι κενό και το data field περιέχει το compiled bytecode του smart contract που πρόκειται να γίνει deploy.[11]. Το bytecode του smart contract εκτελείται από το EVM (ethrereum virtual machine).

> Ethereum has its own virtual machine, a stack based machine to push/pop instructions as in a regular computer with Intel/ARM/AMD CPU. The purpose of the virtual machine is to execute smart contract code.

[12]

Το EVM χρησιμοποιεί ένα σύνολο απο εντολές (opcodes).Αυτή τη στιγμή υπάρχουν 140 opcodes. Κάθε εντολή αποτελείται απο 1 byte, επομένως μπορούμε να έχουμε ένα σύνολο απο 256 opcodes.

$$2^8 = 256 \tag{1.1}$$

> At any given block in the chain, Ethereum has one and only one 'canonical' state, and the EVM is what defines the rules for computing a new valid state from block to block.

[3]

Κάθε φορά που ένα block γίνεται mine και κατ'επέκταση περιέχει κάποια transactions, έχει ως αποτέλεσμα την αλλαγή του state στο ethereum.[10]. Ένα από αυτά τα transaction μπορεί να περιέχει κάποια κλήση προς ένα συγκεριμένο contract account. Κάθε τέτοιο contract call που γίνεται mine έχει ως αποτέλεσμα την αλλαγή του state στο ethereum blockchain. Επομένως, τα smart contracts αποτελούν μια καινοτόμο τεχνολογία προσφέροντας θεωρητικά τη δυνατότητα δημιουργίας πάρα πολλών αποκεντρωμένων εφαρμογών (decentralized applications ή αλλιώς dapp), ενώ συγχρόνως δίνει μια νέα εναλλακτική στη διαφάνεια κάποιων εφαρμογών οπώς θα ήταν για παράδειγμα μια ψηφοφορία.Ανοίγει νέους ορίζοντες στην προσέγγιση μιας αποκεντρωμένης εφαρμογής και κλονίζει πολλά πεδιά όπως είναι IoT, finance και άλλα.[9].

Βέβαια στο σημείο αυτό αξίζει να ξεκαθαρίσουμε ότι, μια αποκεντρωμένη εφαρμογή που στηρίζεται εν προκειμένω στο ethereum, δεν μπορεί να αντικαταστήσει πλήρως το cloud computing, καθώς υπάρχουν περιορισμοί που το καθιστούν δύσχρηστο σα μια μηχανή επεξεργασίας ή αποθήκευσης μεγάλου όγκου δεδομένων. Το πιο χαρακτηριστικό παράδειγμα είναι ότι ένα smart contract πριν γίνει deploy ή αν γίνει ένα call προς αυτό ο sender θα πρέπει να πληρώσει σε gas(ether) την αξία για την εκτέλεση του. Αυτό συμβαίνει επειδή κάθε opcode του bytecode κοστίζει. Επίσης, όταν ένα smart contract γίνει deploy ή κληθεί ένα contract call αλλάζει το state και αυτός είναι ο λόγος που υπάρχει το κόστος αυτό. Πρέπει να τονιστεί ότι ένα smart μπορεί να έχει κάποιες συναρτήσεις οι οποίες δεν οδηγούν

στην αλλαγή του state του ethereum, οι συναρτήσεις αυτές δεν έχουν κάποιο κόστος.Ένα δεύτερο μειονέκτημα είναι ότι η τιμή αυτή, είναι ιδιαίτερα υψηλή όσον αφορα την αποθήκευση δεδομένων από το Ethereum. Το γεγονός αυτό εξηγείται ότι τα δεδομένα αυτά θα πρέπει να μείνουν μόνιμα αποθηκευμένα στην αλυσίδα (blockchain) του ethereum.Συμπερασματικά, καταλήγουμε ότι οι αποκεντρωμένες αυτές εφαρμογές με τη χρήση της τεχνολογίας του blockchain και των smart contracts , μπορεί να χρησιμοποιηθεί σαν μεσάζοντας ώστε να επικυρώνει συναλλαγές εκτός της αλυσίδας του blockchain (off chain validation operations).

Chapter 2

Sia Coin

Υπάρχουν κάποια συστήματα τα οποία έχουν ως στόχο την υλοποίηση του storage απο απλούς users. Τέτοια είναι το Siacoin.

Προκειμένου να υλοποιήσουμε την ιδέα μελετήσαμε το SiaCoin το οποίο είναι ένα cryptocurrency το οποίο χρησιμοποιείται για την αποθήκευση storage απο users. Χρησιμοποιεί τα fileContracts προκειμένου να επικυρώνει ότι πράγματι οι providers αποθηκεύουν τα data για τα οποία έχουν συμφωνήσει, καθώς και αναλαμβάουν να αποδείξουν ότι πράγματι οι providers έχουν τα δεδομένα στη διάθεσή τους.

Παρακάτω, περιγράφεται αναλυτικά η λειτουργία του SiaCoin και δίνεται έμφαση στο κομμάτι που μας αφορά, δηλαδή αυτό του storage.

Το Sia αποτελείται απο 8 modules

- *consensus*
- *gateway*
- *host*
- *renter*
- *wallet*
- *transactiopool*
- *miner*
- *explorer*

Εμάς μας ενδιαφέρει η λειτουργία των modules host και renter. Το Sia , όπως και το ethereum είναι stateful, δηλαδή αλλάζει το state των modules όταν ένα block από transactions γίνει mine (κάθε δέκα λεπτά περίπου) Στο Sia, οι hosts είναι υπεύθυνοι για την αποθήκευση δεδομένων των renters. Για το λόγο αυτό, οι renters δημιουργούν συμβόλαια με τους hosts (file contracts)

Κάθε module ειδοποιείται για την αλλαγή του state του blockchain από τό module consensus. Το consensus module ειδοποιεί όλα τα modules που είναι subscribed σε αυτό (host, renter, wallet). Τα modules (implement subscriber interface) με τη σειρά τους υλοποιούν την ProcessConsensusChange.

2.1 File Contract

Ένα συμβόλαιο περιέχει ένα collateral απο τό host σαν εγγύηση για όλα τα δεδομένα που αποθηκεύει. Αν ο host στο διάστημα που αποθηκεύει τα δεδομένα πέσει και χάσει τό χρονικό διάστημα στο όποίο πρέπει αν ανεβάσει τό storage proof, χάνει όλο το colateral και παίρνει μόνο κάποια από τα χρήματα. Ο renter βάζει κάποια siacoins τα οποία είναι για ολη τη χρονική διάρκεια του file contract. Στο blockchain ανεβαίνουν με τη μορφή των transactions μόνο η αρχική συμφωνία μεταξύ renter και host για τό μέγεθος των δεδομένων που θέλει ο host να αποθηκεύσει καθώς και η κοστολόγηση του host τόσο για αποθήκευση με τιμές ενος byte, η κοστολόγηση του bandwidth. Επίσης η κοστολόγηση για την αποθήκευση πολλαπλασιάζεται επι τον συνολικό αριθμό των blocks που ο host θα έχει τα δεδομένα αυτά. Έπιπλέον, ανεβαίνει και ένα transaction κάποια blocks πριν το storage proof το οποίο περιέχει το τελευταίο file contract revision, αν δεν γίνει renew το contract δε γίνονται άλλα data upload στο contract. Η επικοινωνία μεταξύ renter και host γίνεται μέσω RPCs (remote procedure calls) offchain Έτσι, γίνεται και η αρχική συμφωνία για τη δημιουργία ενός file contract μεταξύ host και renter. Στο μεσοδιάστημα απο την αρχίκή συμφωνία μεταξύ host και renter ο renter έχει τις εξής δυνατότητες όσον αφορά την κίνηση των δεδομένων.

- *Insert new data*
- *Delete Data*
- *Renew Contract (after former file contract has expired)*
- *Download Data*

2.2 Renter

Αρχικά παρουσιάζουμε εποπτικά τη δομή του Renter.

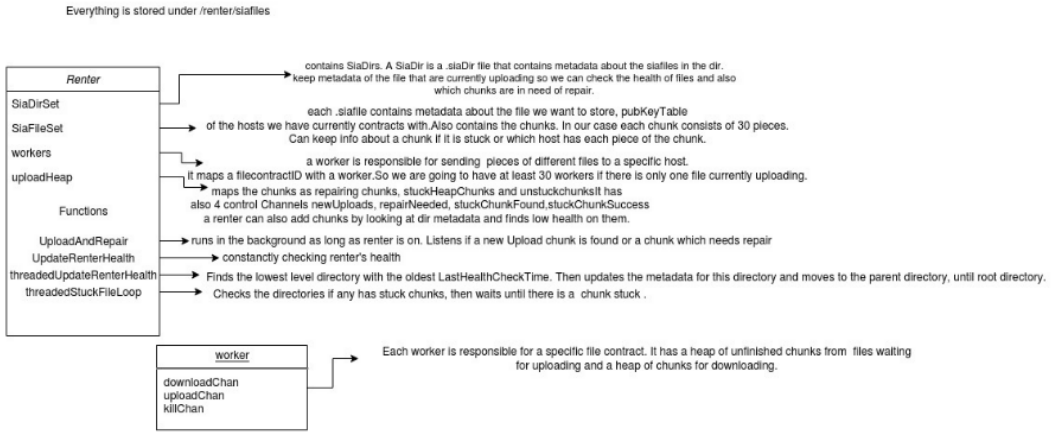

FIGURE 2.1: Δομή και Λειτουργία Renter

HostDB

Δομή που περιλαμβάνει τους hosts τους οποίους γνωρίζει ο renter . Διατηρεί τις εγγραφές για κάθε host σε μια δομή δυαδικού δένδρου, η οποία περιέχει τις παραμέτρους που διαφημίζονται από τον ίδιο τό host. Τους αποδίδει βάρη, τα οποία είναι από τα settings του host οι τιμές για αποθήκευση. Τα χρησιμοποιεί ως κριτήριο για την επιλογή τυχαίων host για τη δημιουργία νεων file contracts. Επίσης ο hostDB είναι subscriber στο consensus set, δηλαδή ενημερώνεται σε κάθε αλλαγή κατάστασης . Σε κάθε αλλαγή κατάστασης γίνεται προσθήκη νέων host, οι οποίοι γίνονται announced μέσω των blocks. Ο HostDB ταξινομεί αυτούς τους hosts μέσω του allowance που έχει ο renter. Αν ο user δεν έχει κάνει set τό allowance , τότε χρησιμοποιείται τό default allowance. Το default allowance περιλαμβάνει 50 hosts, και expected redundancy 3 (θα αναλύσουμε μετά τι σημαίνει αυτό.). Επίσης ο HostDB κάνει ανά τακτά χρονικά διαστήματα scan έτσι ώστε να διαπιστώνει ποιοί hosts από αυτούς που γνωρίζει ειναι online και άρα διαθέσιμοι για upload. Επίσης για κάθε host περίεχει στατιστικά, δηλαδή πόσες φορές είναι offline. Ο Renter κάνει συνεχή scan ανά κάποια λεπτά ώστε να γνωρίζει σε τι κατάσταση βρίσκονται οι hosts.

HostContractor

Υπεύθυνος για τη δημιουργία των file contracts. Τα contracts που δημιουργεί ο renter αποθηκεύονται στο directory contracts ως ¡ filename ¿.contract. Από τα αρχεία αυτά δημιουργούνται τα safe contracts τα οποία αποτελούν μιά δομή του ContractSet την οποία περιέχει ο host contractor.Επομένως ο contractor περιέχει τα old contracts που είχε παλιά ο renter

και έχουν ολοκληρωθεί, όσα έγιναν renew και όσα είναι σε ισχύ . Το pubKeystoContractID αντιστοιχίζει όλα τα public keys των hosts σε FileContractID παλιά και καινούρια.Ο Contractor είναι επίσης subscriber στο consensus set.

WorkerPool
Map από workers , όπου κάθε worker αναλαμβάνει να διαχειριστεί ένα file contract δηλαδή αντιστοιχίζεται σε ένα host .

Ένας renter είναι υπεύθυνος για uploads, tracks, repairs και downloads ενός file. Τα αρχεία ενός renter αποθηκεύονται στο renter/siafiles .Στην αρχή ο renter έχει ένα αρχικό allowance τό οποίο καθορίζει με πόσους hosts θα δημιουργήσει file contracts, καθώς και άλλες αρχικές παραμέτρους που αφορούν τα συμβόλαια αυτά. Έτσι, αρχικά ένας renter που δημιουργείται έχει ως αρχικό allowance 50 hosts.

Εν συνεχεία, θα δούμε τι κάνει ο contractor του renter σε κάθε αλλαγή κατάστασης(consensus change) Σε κάθε αλλαγή κατάστασης, ο contractor ελέγχει τα file contracts που έχει ο renter να είναι όσα έχει στο allowance που ρυθμίζεται από τον user (default 50, εαν υπάρχουν οι hosts βέβαια και αν ο renter έχει τα χρήματα για κάθε contract).

> In each consensus change the contractor checks the set of contracts the renter has against the allowance, renewing any contracts that need to be renewed, dropping contracts which are no longer worthwhile, and adding contracts if there are not enough.

Αξίζει να σημειώσουμε, ότι τα παραπάνω γίνονται εφόσον ο renter έχει unlocked to wallet του και έχει hosts στο allowance του(πιο συγκεκριμένα πρέπει να κάνει set το allowance του). Το κάθε wallet μπορεί να παράξει ντετερμινιστικό αριθμό unlock hashes τα οποία μπορούν να δεχτούν siacoins or siafunds .

Το allowance περιέχει config για τα fund που θα δώσει ο renter τόσο σε bandwidth όσο και σε storage.Περιέχει τον αριθμό των hosts με τους οποίους θέλει να δημιουργήσει file contracts, καθώς και τη διάρκεια του file contract, καθώς και expected redundancy, και άλλες ρυθμίσεις(expected Download, Expected Upload, ExpectedStorage).

Ένα usecase της συνάρτησης maintenance είναι ότι ο contractor Iterate through the contracts again, figuring out which contracts to renew and how much extra funds to renew them with.Γενικά η συνάρτηση αυτή είναι υπεύθυνη να βρίσκει όσα contracts χρειάζονται refill ως προς το funding καθώς όσα contracts μπορούν να γίνουν renew . Έτσι, προσπαθεί να δημιουργήσει τόσα file contracts όσα έχει το allowance του contractor.

Επομένως, όταν δημιουργείται αρχικά ένας renter και κάνει set το allowance του και έχει το wallet unlock, προσπαθεί να δημιουργήσει x file contracts με x διαφορετικούς hosts, ανάλογα με τό allowance του .Το allowance γίνεται set από τον user.Εμείς υποθέτουμε ότι έχουμε τό default allowance σύμφωνα με τό Sia (hosts : 50, redundancy : 3). Έτσι, αρχικά υπολογίζεται εξ αρχής τό αρχικό σύνολο σε currency που δίνει ο renter για κάθε

file contract :

$$total/50(hosts)/3(redundancysolomonReed) \tag{2.1}$$

Έτσι ο contractor έχει τό allowance που περιέχει κάποιες αρχικές παραμέτρους για τό file contract (διάρκεια, contract funding)

> An allowance dictates how much a renter is allowed to spend in a given period. Ο contractor ξεκινά να δημιουργεί 50 file contracts.

Ο contractor ελέγχει πόσα contracts διαθέτει εκείνη τη στιγμή. Σκοπός είναι να έχει συνεχώς όσα contracts έχει στο allowance του τα οποία τα δημιουργεί με διαφορετικό host το καθένα. Ο Contractor δημιουργεί μια blacklist προκειμένου να μη συμπεριλάβει κάποιον host με τον οποίο έχει ήδη συμβολαιο, κοιτάζοντας τα ήδη υπαρχοντα συμβόλαια

Ο hostDB επιστρέφει ένα πλήθος από τυχάιους hosts (μεγαλύτερο του 50) .Ο hostDB επίσης περιέχει για κάθε host τα settings που ο ίδιος ο host διαφημίζει.

Ο contractor ξεκινά να δημιουργεί file contract με κάθε έναν από αυτούς εως ότου έχει τον απαιτούμενο αριθμό file contracts. Δηλαδή στην περίπτωση που δέν έχει κανένα file contract δημιουργεί 50 file contracts .

Αρχικά ο renter ελεγχει για τον εκάστοτε host τα settings του να δεί ότι όντως ικανοποιούν τις συμβάσεις. Τα settings αυτά του host περιλαμβάνουν ενδεικτικά.

- *Collateral*:amount that the host will put up as an assurance to the renter that the host is really commited to keeping the file.
- *Contract Price*:The number of coins the renter needs to pay to the host just to open a contract with him.(Host wants these funds for siacoin fees for submitting the revision and storage proof to the blockchain)
- *UploadBandwidthPrice*:is the cost per byte for uploading data to the host

Ο renter κανει unlock το wallet του με το primary seed του και παράγει με τη βοήθεια του seed μια νέα διεύθυνση στην οποία θα πληρωθεί. Αυτή η διεύθυνση προκύπτει από το hashing του unlock conditions . Hash(unlockConditions) Πιο συγκεκριμένα, απο το primary seed του wallet παράγεται ένα spendableKey (unlockConditions, secretKey).Το primary seed ενός wallet μπορεί να παράγει ντετερμινιστικό αριθμό spendablekeys έτσι ένα wallet γνωρίζοντας το primary seed μπορεί να αναπαράγει όσες φορές θέλει τα spendablekeys και να βρεί τα SiacoinOutputs τα οποία αναφέρονται σε κάθε ένα απο αυτά. Το ζευγάρι (sk, pk) παράγεται με entropy H(seed —— index) .

Το unlockConditions αποτελείται από Timelock(μέχρι τότε δεν μπορεί να μπεί στο blockchain), PublicKeys(που συμμετέχουν στο transaction, εκτός αν είναι SiacoinInput ή SiaCoinOutput θα έχει πιθανώς μόνο ένα), Signatures*(αντιστοίχιση ένα προς ένα με τα public keys).

Τα unlock conditions που έχει ένα wallet δεν αλλάζουν το timelock. Το timelock έχει νόημα και αλλάζει μόνο όταν το block γίνεται mine.

Έπειτα, ο renter δημιουργεί τό transaction το οποίο αρχικά είναι κενό //συμπληρώνοντας μόνο κάποια από τα πεδία του, όπως είναι τό δικό του allowance, τό αρχικό funding του στο file contract.// Στο αρχικό allowance η προτεινόμενη περίοδος ενός filecontract είναι 3 μήνες που αντιστοιχούν σε συγκεκριμένο αριθμό blocks δεδομένου ότι ανά δέκα λεπτά γίνεται mine ένα block.

Επομένως στην αρχή υπολογίζεται τό αρχικό funding του renter που καλυπτει το αρχικό κόστος της δημιουργίας filecontract και του κόστους δημιουργίας που έχει ζητήσει ο host. Δημιουργούνται στο transaction δύο ειδών outputs missed proof και τα valid proof. Τα οποία αρχικοποιούνται κατά τη δημιουργία του transactions απο το renter και το host. Αυτά θα αλλάζουν μεταξύ τους ποσά κατά τη διάρκεια του file contract ανάλογα με το τι χρήση γίνεται από τπ renter πχ insert,download data αλλά και από το proof που καλείται να δώσει ο host.

Προκειμένου να δημιουργηθεί το SiacoinOutput του renter επι παραδείγματι, μπορεί να χρειαστούν περισσότερα από ένα SiacoinInput από το renter.Δημιουργείται ένα parent transaction το οποίο μπορεί να έχει ένα η περισσότερα SiacoinInputs(όσα χρειάζονται ώστε να έχουμε το funding του file contract) και δύο SiacoinOutput ένα με το ακριβές funding το οποίο θα χρησιμοποιηθεί ως SiacoinOutput στο transaction και άλλο ένα που θα επιστρέφει τα υπόλοιπα σε ένα pay address(unlockHash) που θα ανήκει στο renter. Ο renter υπογράφει κάθε SiacoinInput της parent transaction.

Πριν ξεκινήσει η αρχική rpc για τη δημιουργία του file contract με τον host, ο renter βάζει σαν input τό funding του στο transaction που ξεκινά να δημιουργεί.

Επίσης ο renter δημιουργεί το file contract . Παράγει filecontractseed τα οποία βάζει στο arbitrary data της transaction ώστε να μπορεί ο renter να βρίσκει τα transactions εκείνα τα οποία περιέχουν filecontracts που τον αφορούν. Δημιουργεί το unlockConditions του fileContract το οποίο πρέπει να γίνει sign τόσο από το renter όσο και από το host. Τα sk,pk του renter προκύπτουν GenerateKeyPair(params.RenterSeed,fcTxn). Δημιουργούνται τα SiacoinOutputs του file contract τα οποία περιέχουν 1 valid output και 1 missed output για το renter και αντίστοιχα για το host.Τέλος, υπάρχει και ένα Output για την κενή διεύθυνση σε περίπτωση που ο host έχει κάποιο penalty τα χρήματα που χάνει να καίγονται ώστε ο renter να μην έχει κάποιο incentive να χάνει ο host χρήματα.

2.3 Δημιουργία File Contract

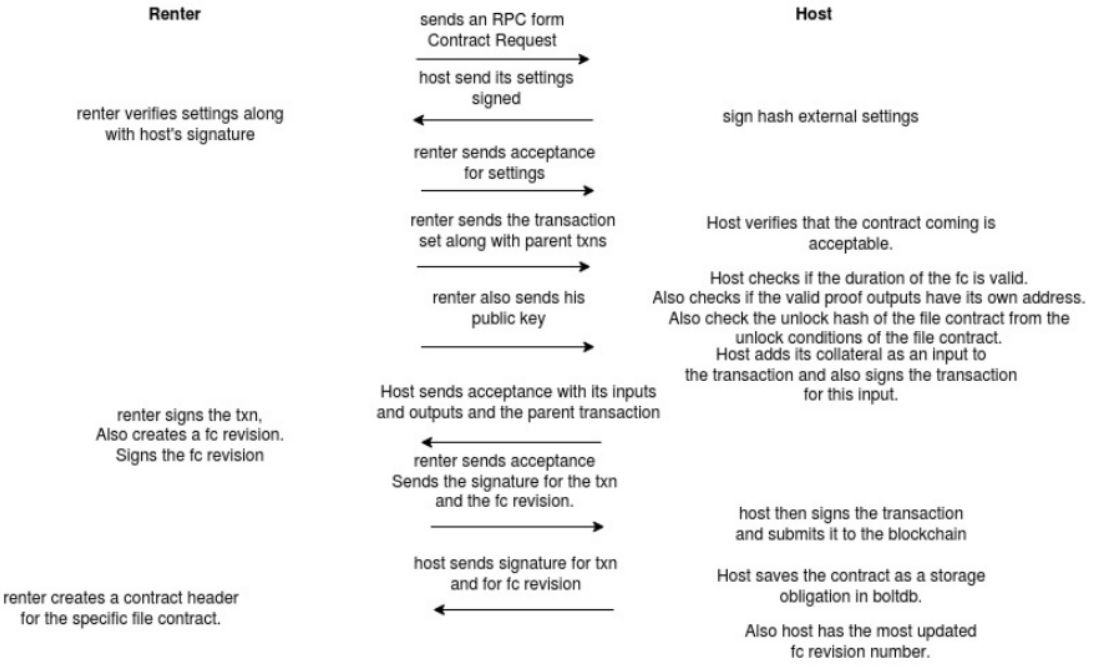

FIGURE 2.2: Δημιουργία File Contract

Στην παραπάνω εικόνα φαίνεται η διαδικασία επικοινωνίας μεταξύ renter και host για τη δημιουργία του file contract.

2.4 Αποθήκευση File Contract από Renter και Host

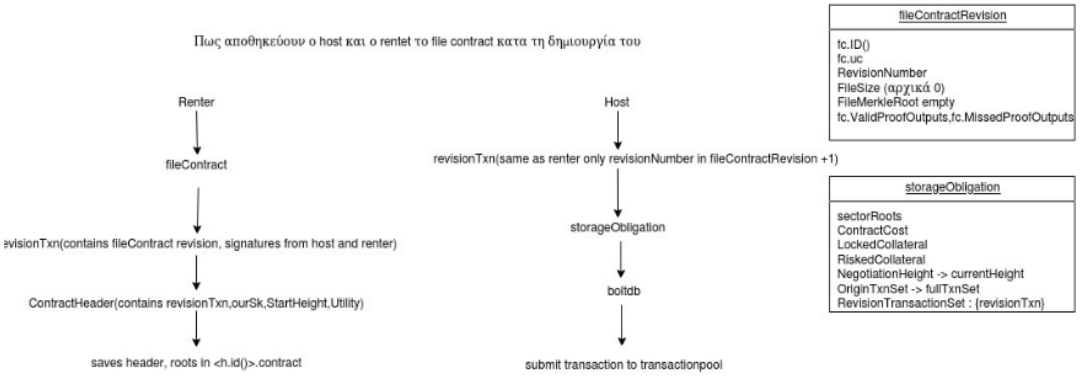

FIGURE 2.3: Αποθήκευση File Contract από Host και Renter

Ο Renter αποθηκεύει τα contracts που δημιουργεί στο δίσκο και πιο συγεκριμένα στο directory renter/contracts/ . Έχουν την κατάληξη .contract . Στο file αυτό αποθηκεύεται το contract header και τα merkle roots.

Σε κάθε file contract revision αναγράφεται το updated fileContract size και το updated merkle root.

Ο host αποθηκεύει τα storageObligations στη boltdb, σε key,value pairs όπου το key του so είναι το fileContract id .

2.5 Upload File

Παρακάτω θα δείξουμε με σχήματα πώς τι συμβαίνει απο τη στιγμή που καλείται ένας renter να κάνει upload ενα file , καθώς και δείχνεται πως ο renter αντιμετωπίζει το αρχείο δηλαδή πως το σπα σε μικρότερα κομμάτια, αλλά και τις δομές πυ χρησιμοποιεί.

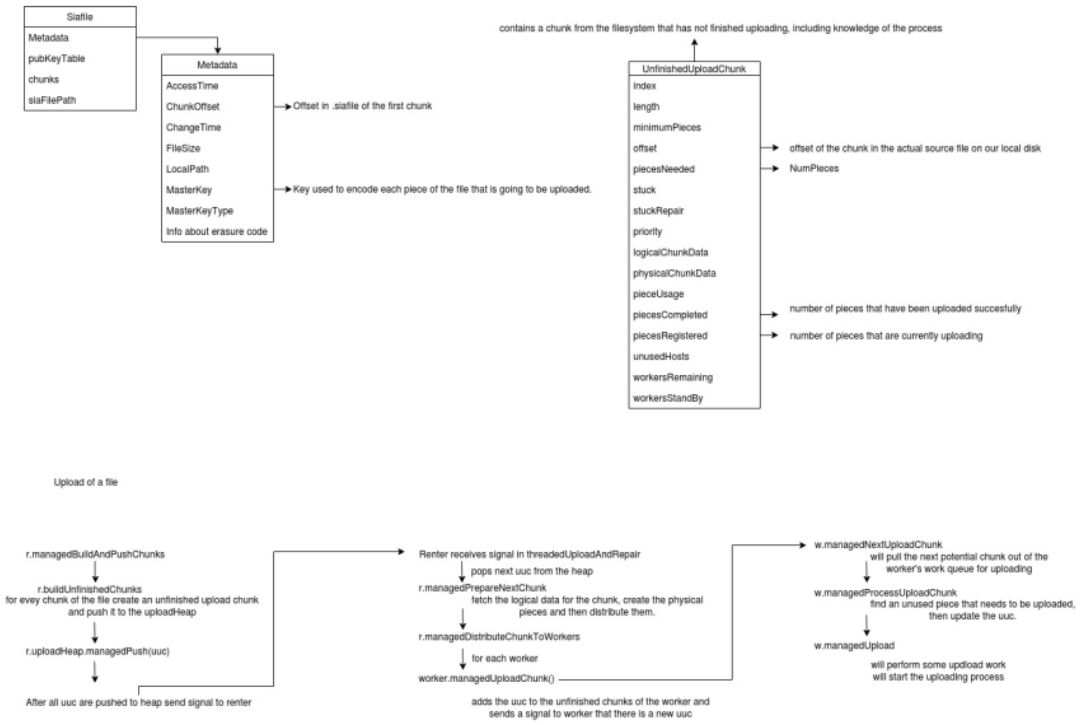

FIGURE 2.4: Πως ο renter αντιμετωπίζει ένα file

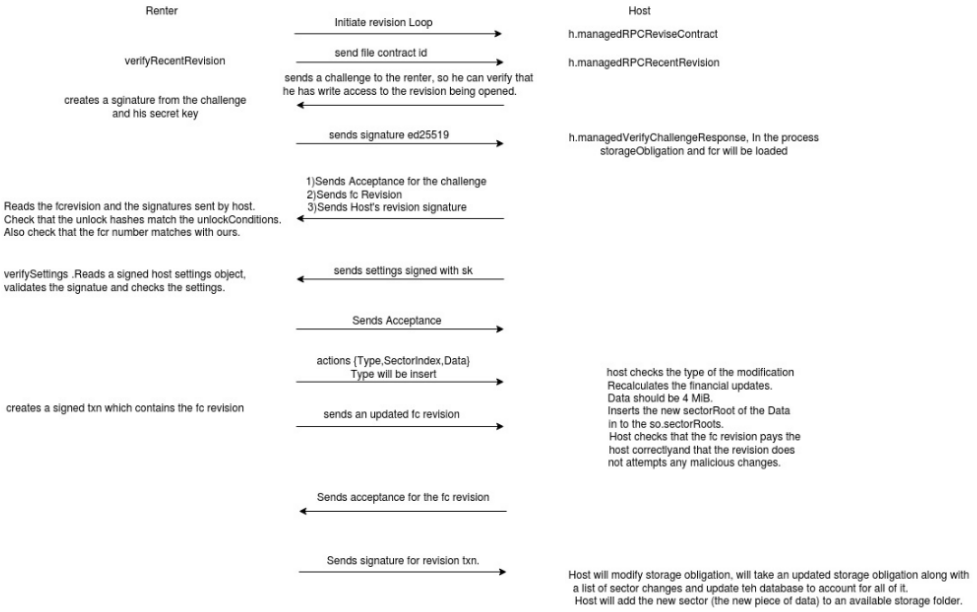

FIGURE 2.5: Επικοινωνία Renter και Host κατά το Upload

Κατά την αρχικοποιήση του ο renter, ελέγχει αν υπάρχουν file, τα οποία γίνονται upload και τα κάνει load απο το renter.json . Πιο αναλυτικά, όλα τα files του τα οποία γίνονται upload βρίσκονται κάτω απο το /renter, μπορεί να είναι είτε files είτε ακόμα και directories. Γενικά τα files που γίνονται upload έχουν την κατάληξη .siafile και περιέχουν τα metadata .Στα metadata υπάρχουν όλες πο πληροφορίες για έναfile, όπως πόσο είναι το erasure coding, τι μέγεθος έχει ένα chunk, αλλά και πληροφορίες σχετικά με το κάθε chunk άν έχει γίνει upload έαν είναι stuck ή εαν θέλει repair.Επίσης κάθε chunk περιέχει τα pieces τα οποία αποτελούνται απο το merkle root και το pk του host ο οποίος το έχει. Ελεγχει πρωτα απο όλα να υπάρχει ένας worker για κάθε file contract που ο renter έχει , επίσης αφαιρεί κάποιον worker ο οποίος μπορεί να αντιστοιχεί σε ένα file contract το οποίο δεν υπάρχει πια. Η αντιστοιχία είναι ένα πρός ένα. Ένας worker έχει ως συνεχή δουλειά να ακούει έαν υπάρχει κάποιο upload ή download.

Ο renter αποθηκεύει όλα τα αρχεία που έχει προς upload στο /renter/siafiles. Παραδείγματος χάριν, εαν θέλει να κάνει upload το /user/file.pdf . Θα αποθηκευτεί στο /renter/siafiles/user/file.siafile. Στο siafiles θα δημιουργήσει ένα user.siadir, το οποίο θα είναι τα metadata για το directory user και θα περιέχει πληροφορίες όπως το health του directory καθώς και πόσα chunks του file είναι stuck. Αντίστοιχα το file.siafile περιέχει metadata για το file σε ποιους host έχει γίνει upload , καθώς και τα chunks του , αν είναι stuck ή όχι αλλά και το erasure coding που έχει χρησιμοποιηθεί .

- *Η uploadAndRepair*, αναλαμβάνει να ενημερώσει τον αντίστοιχο worker ότι ενα chunk είτε χρειάζεται upload ή repair.

  (is a background thread that maintains a queue of chunks to repair.This thread attempts to prioritize repairing files and chunks with the lowest health).

Η συνάρτηση αυτή δημιουργεί μια heap απο τα directories του renter και κάθε φορά ελέγχει με dfs ένα ένα τα metadata των directories και ελέγχει ποιο δεν είναι healthy. Εαν κάποιο directory δεν είναι healthy τότε ελέγχει όλα τα siafiles που ανήκουν σε αυτό το directory βάζοντας στην uploadHeap όσα chunks δεν έχουν γίνει σωστά upload(δηλαδή δεν είναι healthy, δεν έχουν γίνει upload ολά τα pieces ενός chunk σε εμάς είναι 30 τον αριθμό).Η uploadAndRepair δε δέχεται όσα chunks είναι stuck.Επίσης η uploadAndRepair καλείται κατά την πρώτη φορά που ένα chunk γίνεται upload.Διαφορετικά εαν δε βρεθεί κάποιο directory unhealthy, τότε απλώς περιμένει η uploadHeap ειδοποίηση πιθανώς όταν πρέπει κάποιο chunk να γίνει upload.

- *Η threadedUpdateRenterHealth*, βρίσκει σε ποιο directory έχει γίνει χρονολογικά παλιότερα το update και ανανεώνει τα metadata του directory ανάλογα με τις αλλαγές που έχουν γίνει στο file

  (reads all the siafiles in the renter, calculates the health of each file and updates the folder metadata.Finds the lowest level directory with the oldest LastHealthCheckTime).

Για αυτό το directory υπολογίζει και ανανεώνει όλα τα file που ανήκουν σε αυτό αλλάζοντας τόσο τα file metadata όσο και τα metadata του ίδιου του directory. Για κάθε file ελέγχει το ποιοι hosts είναι online έτσι ώστε να υπολογίζει το κατα πόσον ένα file είναι healthy ή όχι και σε τι ποσοστό . Υπολογίζει το health ενός file, το οποίο είναι το χειρότερο health ενός chunk του. Γενικά τα metadata του directory καθώς και όλων των parent directory αυτού εως το root υπολογίζονται από την managedBubbleMetadata .

ChunkHealth returns the health of the chunk which is defined as the percent of paritypieces remaining.

$$ChunkHealth = 1 - \frac{goodPieces - minPieces}{redundancyPieces} \tag{2.2}$$

$$bestHealth = 0 \rightarrow goodPieces = 30 \tag{2.3}$$

$$worstHealth = \frac{3}{2} \rightarrow goodPieces = 0 \tag{2.4}$$

$$health \leq 1 \rightarrow recoverable \tag{2.5}$$

$$health \geq 1 \rightarrow NeedsRepairFromDisk \tag{2.6}$$

Επομένως

$$0 \leq chunkHealth \leq \frac{3}{2} \tag{2.7}$$

Έτσι, ένα chunk είναι healthy και full redundant εαν ChunkHealth = 0, δηλαδή goodPieces = NumPieces(στην περίπτωσή μας 30)

$$Redundancy = \frac{numPiecesRenew}{MinPieces} \tag{2.8}$$

Εαν το Redundancy <1 και το file δεν υπάρχει στο local disk, τότε το file είναι unrecoverable. Εαν βρεί ότι το λιγότερο healthy chunk >0.25 , τότε ο renter ενημερώνει τη threadedUploadAndRepair.

- *Η threadedStuckFileLoop*, ελέγχει εαν υπάρχει κάποιο directory με stuck chunks , αλλιώς περιμένει κάποιο chunk να δηλωθεί ως stuck και να ενημερωθεί μέσω του channel της uploadHeap.

Κάθε worker έχει τα δικά του downloadChunks και uploadChunks . Από αυτά ο worker επιλέγει ένα ελεύθερο piece και το κάνει upload. Στο download πρέπει να επιλέξει το συγκεκριμένο piece που ανήκει στο host που αντιστοιχεί στο συγκεκριμένο worker.

Τώρα θα αναλύσουμε το flow, προκειμένου ο renter να κάνει upload ένα file στους hosts. Ο Sia client στέλνει ένα post στο /renter/upload/SiaPath όπου στο body του περιλαμβάνει το path του file στο filesystem του renter.

Το redundancy επιτυγχάνεται μέσω του αλγορίθμου Solomon Reed.

> When a file is stored, it is broken into chunks of 40 MiB. Each chunk is broken into 10 pieces, all the same size (4 MiB). Then 20 additional pieces of the same size(4 MiB) are created that hold parity. So the original file can be reconstructed from any 10 out of 30 pieces. Επομένως ένα chunk θα αποτελείται απο 30 pieces(4Mib το καθένα).

Ο renter δημιουργεί το directory για το file. Το directory θα περιέχει το .siafile, το οποίο θα περιέχει τα metadata του siafile , οπώς είναι το size του file, το local path του file στο δίσκο, το key με το οποίο κάθε piece έχει γίνει encode κτλπ, καθώς και τα chunks του file (κάθε chunk αποτελείται από 30 pieces(merkle root, pk)).

Στην αρχή ο renter ελέγχει το file ότι όντως υπάρχει στο δίσκο . Έπειτα, ελέγχει έαν έχει τουλάχιστον

$$\frac{NumPieces + MinPieces}{2} \tag{2.9}$$

contracts έτσι ώστε να υπάρχει τουλάχιστον 1 redundant piece για κάθε piece ενός chunk. (Δηλαδή να έχουμε redundancy 2).

Επειτα, για κάθε file, που θέλουμε να κάνουμε upload, ο renter δημιουργεί μια λίστα απο unfinishedchunks τα οποία θα βάλει στην uploadHeap του renter

> builds the unfinished upload chunks and adds them to the upload heap

Το unfinishedupload chunk είναι ένα chunk του file το οποίο γνωρίζει το progress του κατά τη διάρκεια του upload. Ελέγχει εαν εχουν γίνει upload pieces και για αυτά που βρίσκει ότι έχουν γινει upload ελέγχει εαν το filecontract είναι goodForRenew και κατόπιν θεωρεί ότι αυτό το piece έχει γίνει ήδη upload οπότε δε θα ασχοληθεί κανείς να γίνει αυτό το piece upload.

Στη συνέχεια ελέγχει το health του κάθε chunk του file το οποίο εδώ θα είναι

$$\frac{1}{2} \tag{2.10}$$

, αφού ακόμα κανένα piece του file δεν έχει γίνει upload(σύμφωνα με τον παραπάνω τύπο 3.2). Ένα uuc για να είναι repairable πρέπει το health του να είναι είτε <1 είτε το file να υπάρχει στο δίσκο. Όταν ο renter δημιουργήσει όλα τα uuc (buildUnfinishedChunks) τα κάνει push στην uploadHeap

Εν συνεχεία, η uploadAndRepair καλεί την r.managedRepairLoop, η οποία κάνει pop απο την heap το επόμενο uuc , ελέγχει εαν υπάρχουν τουλάχιστον minimumPieces τον αριθμό (εδώ 10) workers και αναλαμβάνει να το κάνει έτοιμο για upload.

> Fetching the logical data for the chunk from the disk , erasure coding the logical data into the physical data and passing the work onto workers. Fetching the logical data will get the raw for a chunk data from the disk. Logical data consists of 10 pieces(MinPieces) of 4 MiB each. Create the physical data pieces for the chunk, then immediately release the logical data.

‘Οπως είχαμε αναφέρει, χρησιμοποιείται ο αλγόριθμος Solomon Reed για redundancy. Πιο συγκεκριμένα αρχικά έχουμε 40 MiB logical data. Παίρνουμε κάθε φορά 640 Bytes και βάζουμε 64 byte σε κάθε piece. Από τα 10 pieces των 64 byte δημιουργουμε άλλα 20 redundant των 64 byte , τα οποία βάζουμε στα psysical μας NumPieces. Έτσι με αυτό το τρόπο σπάμε τα data μας ανα 640 bytes σε 10 pieces. ‘Επειτα κάθε piece από τα 30 το κάνουμε encrypt με το masterKey του uuc.

Σημείωση : η ίδια συνάρτηση χρησιμοποιείται και εάν ένα file έχει ChunkHealth <1 καί χρειάζεται repair.

Επειτα καλείται η r.managedDistributeChunkToWorkers Κάθε worker ξεκινά να επεξεργάζεται από τη λιστα του το uuc. Αρχικά κάνει κάποιους ελέγχους ότι το fc είναι ακόμα ενεργό, και βάζει το uuc στη λίστα με άλλα uuc που πρόκειται να επεξεργαστεί ο worker για upload. Ο worker αναζητά στο uuc το πρώτο ελεύθερο piece που θα βρεί προκειμένου να το κάνει upload. Checks again if the host and fc is ok. Έπειτα καλείται η w.managedUpload. Έδω δημιουργεί έναν editor οποίος ξεκινάει επικοινωνία με rpc με το host με τον οποίο έχει το file contract.

Στο σχήμα του upload περιγράφεται αναλυτικά το πρωτόκολο εποκοινωνίας. Εν ολίγοις όμως, ο renter στέλνει στο host το fcid. Ο host με τη σειρά του στέλνει ένα challenge στο renter, προκειμένου να δεί εαν είναι όντως ο ίδιος . Έπειτα αν είναι πράγματι ο renter ο host στέλνει το τελευταίο file contract revision καθώς και τις υπογραφές , η οποία η μια είναι του renter και η άλλη του host, οι οποίοι υπογράφουν τα unlock conditions, του file contract revision. 'Επειτα ξεκινάει η διαδικασιά του upload από το renter.

Ο renter αρχίζει να κάνει τους υπολογισμούς και τις αλλαγές στο file contract revision.

- *BlockBytes* = sectorSize(4MiB) * (NewWindowEnd - he.height) byte , επί πόσα block θα είναι uploaded
- *SectorStoragePrice* = host.StoragePrice * BlockBytes
- *SectorBandwidthPrice* = UploadBandwidthPrice * sectorSize
- *sectorCollateral*
- *sectorPrice* = SectorStorageProce + SectorBandwidthPrice
- *sectorRoot*
- *MerkleRoot*

  Renter sends the updated fc revision to the host along with the modifications actions. Then host checks the modifications and recalculates the finances in order to check if the filecontract is well-formed. Host checks all non volatile fields , along with the amount of siacoins

Host updates the fields of the storageObligation and saves them to db. 'Επειτα για κάθε gainedSector(4 MiB of data) h.AddSector AddSector will add a sector to the contract manager.

Ο Host αποθηκεύει τα data, sectors όπως τα ονόμάζει από το renter σε storagefolders που έχει φροντίσει προηγουμένως ο user να δημιουργήσει.Ένα storage folder ουσιαστικά αποτελείται από ένα directory στο filesystem του host το οποίο περιέχει δύο files.

- *Metadatafile* : το οποίο αποτελείται απο ομάδες των 14 byte όπου κάθε τέτοια ομάδα αντιστοιχεί σε ένα sector. Τα πρώτα 12 byte αποτελούν το id του sector, το οποίο προκύπτει απο το hash(cm.salt, sectorRoot). Το salt αρχικοποιείται μία φορά κατα τη διάρκεια που δημιουργείται ο renter.

  > SectorSalt is a persistent security field that gets the first time the contract manager is created,It's used to randomize the location on-disk that a sector gets stored.So that an adversary cannot maliciously add sectors to specific disks, or perform other malicious actions.

- *DataFile* : περιέχει sectors των 4Mib και το index τους αντιστοιχεί με εκείνο στο metadatafile.

Η AddSector αρχικά βρίσκει το id του sector , hashing sector root with the cm.salt. Έπειτα, ελέγχει εαν υπάρχει ήδη ή όχι το sectorlocation για το συγκεκριμένο id που υπολογίστηκε. Έαν δεν υπάρχει καλείται η cm.wal.managedAddPhysicalSector η οποία αναλαμβάνει αρχικά να βρεί ένα τυχαίο storageFolder το οποίο να έχει χώρο να αποθηκεύσει το συγκεκριμένο sector(4MiB).

> Find a commited storage folder that has enough space to receive this sector.

vacancyStorageFolder : returns the storagefolder along with its index. grab a random free sector from the sf.usage array. start from a random usage[i] and check which usage[i] is not completely full.

Then contract manager writes the sector to disk

```
err = writeSector(sf.sectorFile,sectorIndex , data)
err = wal.writeSectorMetadata(sf, su)
```

Τα παραπάνω δε γίνονται εαν για το συγκεκριμένο id υπάρχει ήδη το αντίστοιχο sectorlocation. Ο contract manager προκειμένου να είναι ACID έχει το δικό του wal , το οποίο εξασφαλίζει ότι άν γίνει κάποιο shutdown προτού προλάβει να προστεθεί κάποιο sector θα το δεί κατά την επανεκκίνησή του από το wal.

Ο contract manager προκειμένου όταν επαναλειτουργεί να βρίσκει τα storage folders και τα sector roots που έχει αποθηκευμένα δημιουργεί το αρχείο host/contractmanager/contractmanager.json.Το αρχείο αυτό περιέχει το salt που ο host χρησιμοποιεί για να κάνει hash το sector root και να προκύψει από εκεί το id του sector. Επίσης περιέχει και έναν πίνακα από savedStorafolders . Ένα saved storage folder περιέχει το index του storagefolder, το path του προκειμένου να μπορούν να γίνει ανάκτηση των sectors. Τέλος περιέχει ένα []uint64 , δηλαδή ένα κελί αυτού του πίνακα αντιστοιχεί σε 64 sectors(bitfield). Κάθε bit δείχνει εαν στο συγκεκριμένο index του storage folder υπάρχει κάποιο sector.

> Contract manager is responsible for managing contracts that the host has with renters, including storing the data, submitting storage proofs and deleting the data when the contract ends.

Επίσης όταν ο Host κάνει form ένα contract για το soid αντιστοιχίζει κάποια blockheights στα οποία πρέπει να γίνει κάτι για το filecontract.

```
h.queueActionItem
```

Τα ActionItems αποτελούνται από key,value pairs τα οποία αντιστοιχίζουν σε blockheights soids. Ένα blockheight δηλαδή περιέχει πολλά soids 'Ετσι για κάθε νέο soid δημιουργούνται 3 νέα key,value pairs, τα οποία αντιστοιχίζονται σε τρία διαφορετικά blockheights.

- resubmission: εαν το fc δεν έχει γίνει ακόμα mine από το blockchain.
- υπενθύμιση λίγο πριν γίνει expire το contract ο host να ανεβάσει το fcrevision.
- Ύπενθύμιση ο host να ανεβάσει το storage proof.

2.6 Sia Subscribers to ConsensusChange

Τέλος, κάνουμε μια αναφορά στους subscribers του consensus change.

- *HOSTDB* : Sees each set of blocks and searches for host Announces. Each host founded will be inserted into the set of all hosts and if it is online and responding to requests it will be put into the list of active hosts. Then adds the host to the queue scan

  ```
  hdb.queueScan(Host)
  ```

  Μάλιστα βάζει το host σε τυχαίο position στην scanlist. HostDb every few hours scans the hosts in order to see who is online and keeps statistics for each host in hostdb Έτσι καθορίζεται και το contract utility

- *Contractor(Renter)*
  1. *managedRecoverContracts*
  2. *managedMarkContractsUtility* : Checks every contract and figures out whether the contract is good for upload or whether is good for renew. Finds the minimum score a host is allowed to have to be considered good for upload
  Ανανεώνει το contract utility για κάθε contract

  ```
  If the score of a host is low then !GoodForUpload ‘and‘ !GoodForRenew
  If the host is offline at the time then !GoodForUpload ‘and‘ !GoodForRenew
  If we are close to renew then !GoodForUpload
  If the contract does not have enough funds remaining then !GoodForUpload ‘but
  Else GoodForUpload ‘and‘ GoodForRenew
  ```

3.Iterate through the set of contracts, figuring out which contracts to renew and how much to renew them with. 4.Add any contracts in order to always have the Allowance number , which in our case is 50.

- *Host* Ελέγχει κάθε block το οποίο έγινε mine, αναζητώντας εαν περιέχει συμβόλαια στα οποία είναι ο ίδιος.
  - *Εαν υπάρχει fc* : so.OriginConfirmed = true
  - *Εαν υπάρχει fcRevision* : so.RevisionConfirmed = true
  - *Εαν υπάρχει storageProof* : so.ProofConfirmed = true

  Έπειτα ο host ελέγχει κάθε soid το οποίο βρίσκεται στο ActionItem key value pairs.

```
If !so.Confirmed then resubmit the fc transaction, also queue another ActionItem
If !so.RevisionConfirmed && len(so.RevisionTransactionSet) > 0 then add a miner f
If it is time for storageProof then construct the storage proof.
```

The index of the sector we are going to use for the merkle proof is calculated by appending the fcID to the trigger block (blockheight that the contract started) and taking the hash, then converting the hash to a numerical value and modding it against the number of segments in the file.

Chapter 3

Σχεδιασμός

Στο παρόν κεφάλαιο θα αναλύσουμε το σχεδιασμό μας . Στόχος μας είναι η δημιουργία μιας αποκεντρωμένης αγοράς. Μέσω αυτής θέλουμε, κάποιος χρήστης (renter) ο οποίος θέλει να νοικιάσει ελεύθερο χώρο (storage) να μπορεί μέσω της αγοράς να βρει συμφέρουσες προσφορές απο άλλους χρήστες(hosts), οι οποίοι είναι διατεθειμένοι να προσφέρουν το δικο τους ανεκμετάλλευτο χώρο αποθήκευσης. Επιπλέον, υλοποιούμε και μηχανισμούς ώστε ο host να αποδεικνύει ότι πράγματι διατηρεί στην κατοχή του τα δεδομένα .

3.0.1 Δημιουργία δημοπρασίας και συμβολαίου αποθήκευσης

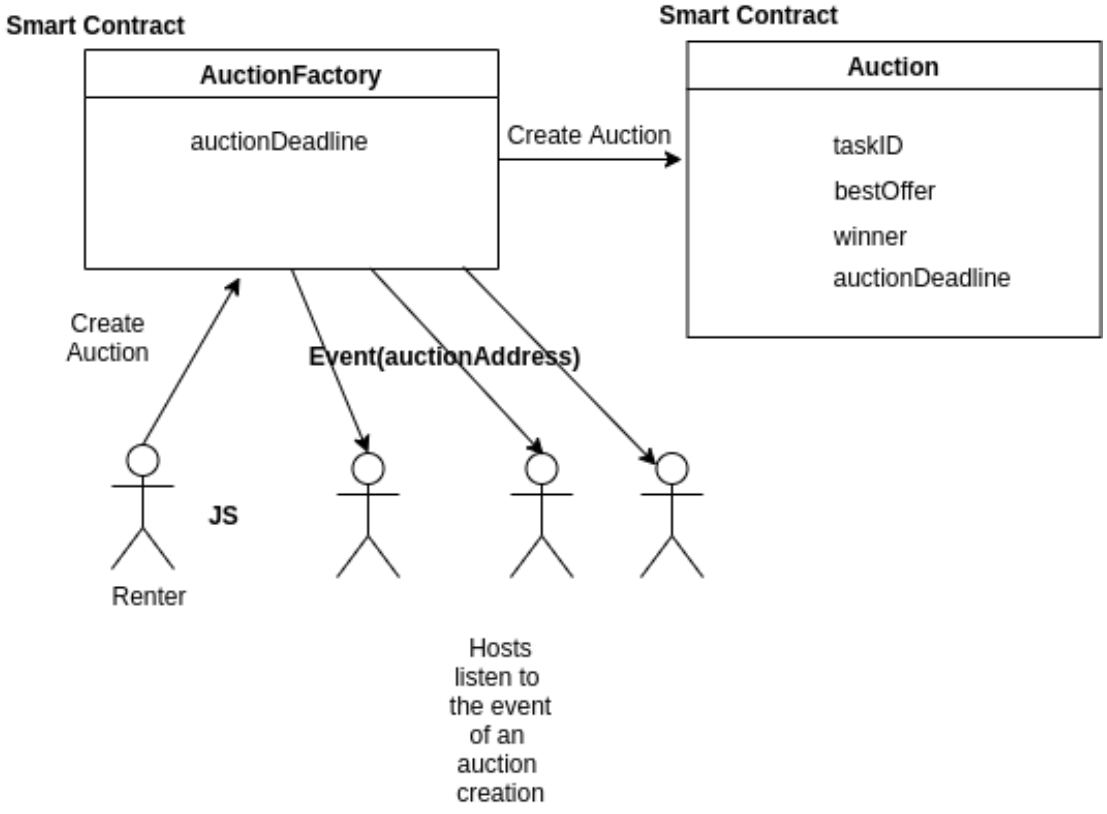

FIGURE 3.1: Auction Create

Στο ανωτέρω σχήμα, το Auction Factory επιτελεί το ρόλο της αγοράς . Το Auction Factory είναι ένα smart contract. Είναι εκείνο, το οποίο δημιουργεί ένα auction δηλαδή τη δημοπρασία, η οποία αποτελεί επίσης ένα smart contract. Επιπλέον, οι actors του σχήματός μας, ο renter και οι hosts , θεωρείται ότι είναι υλοποιημένοι σε node javascript. Η διαδικασία της δημοπρασίας ξεκινάει απο το renter, ο οποίος κάνει ένα contract call create Auction και με τη σειρά του το Auction Factory αρχικοποιεί ένα καινούριο smart contract, αυτό του Auction. Το contract call του createAuction πρέπει να περιέχει τις παραμέτρους που επιθυμεί ο renter για τη δημιουργία της δημοπρασίας.Οι παράμετροι είναι :

- *Auction Deadline*: Η χρονική διάρκεια της δημοπρασίας.
- *Bid*: Η αρχική τιμή απο την οποία εκκινεί η διαδικασία της δημοπρασίας, και και στο τέλος επιλέγεται η πιο συμφέρουσα.

Εν συνεχεία, οι host ενημερώνονται για τη δημιουργία της δημοπρασίας μέσω ενός feature του ethereum , των events. Οι host ακούνε το event επειδή γνωρίζουν την διεύθυνση του smart contract auction Factory , το οποίο είναι εκείνο που καλεί το event.

Στο επόμενο σχήμα παρουσιάζουμε τη διαδικασία που ακολουθεί εως ότου ολοκληρωθεί η δημοπρασία.

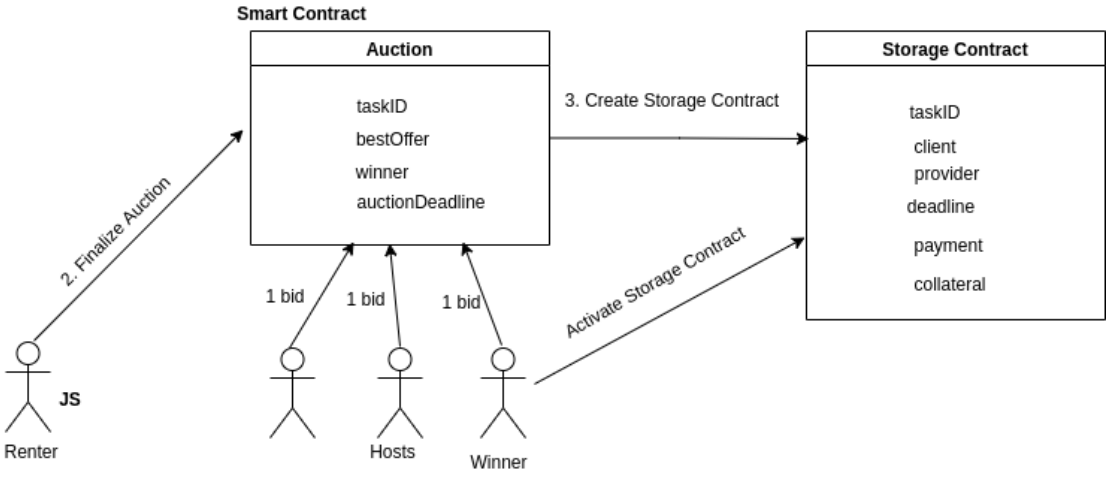

FIGURE 3.2: Auction Finalize

Από το σχήμα φαίνεται ότι κάθε host κάνει ένα bid , το οποίο εαν είναι μικρότερο απο το ήδη υπάρχον ή το αρχικό , θα επικρατήσει έναντι των υπολοίπων. Το smart constract auction μέσω της μεταβλητής winner αποθηκεύει ποιός είναι ο host με την πιο συμφέρουσα προσφορά. Το auction contract μπορεί να ολοκληρωθεί με δύο τροπους

- *Auction Deadline*: Μετά το πέρας του χρονικού πλαισίου που έχει οριστεί απο το renter.
- *Auction Finalize*: Ο renter έχει τη δυνατότητα μέσω του contract call να τερματίσει το auction Contract, οπότε ο host με την πιο συμφέρουσα προσφορά μέχρι εκείνη τη στιγμή θα είναι ο νικητής.

Όταν τερματιστεί το auction με οποιονδήποτε απο τους παραπάνω τρόπους το auction contract προχωρά στη δημιουργία ενος νέου contract αυτό του storage contract, το οποίο θα αποτελεί τη συμφωνία μεταξύ renter και host. Στην παραπάνω εικόνα η δημοπρασία τερματίζεται από τον ίδιο τον renter.

3.0.2 Χωρισμός αρχείου

Παρακάτω παραθετούμε το τρόπο χωρισμού του αρχείου, το οποίο πρόκειται να μοιραστεί στους hosts μέσω της κάτωθι εικόνας.

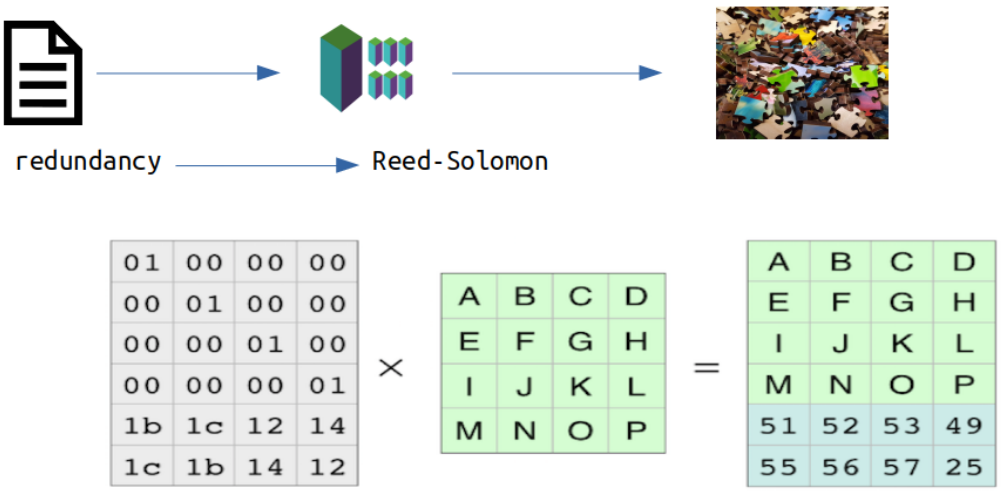

FIGURE 3.3: Χωρισμός Αρχείου και Redundancy

Αρχικά, χωρίζουμε το αρχείο μας , το οποίο προορίζεται για upload, σε μικρότερα κομμάτια τα οποία τα ονομάζουμε chunks. Συνήθως καλούμαστε να επιλέξουμε ένα μέγεθος το οποίο δε θα είναι ούτε πολύ μικρό αλλά ούτε και πολύ μεγάλο ώστε να μην σπαταλούμε πολύ bandwidth. Στην περίπτωση του Siacoin, το μέγεθος του chunk που έχει επιλεχτεί είναι περίπου 40MiB. 'Ενα chunk , όμως χωρίζεται και σε περαιτέρω κομμάτια τα οποία τα ονομάζουμε pieces. Εν προκειμένω, εαν το μέγεθος του chunk θεωρήσουμε ότι είναι 40MiB, τότε το chunk μας , χωρίζεται σε 10 pieces (4 MiB/piece). Επομένως ο αριθμός των chunks αρχικοποιείται ως εξής :

```
numChunks := fileSize / file.ChunkSize()
      if fileSize%file.ChunkSize() != 0 || numChunks == 0 {
              numChunks++
}
```

Από το παραπάνω μπλοκ φαίνετα ότι έαν ο αριθμός των chunks δε διαιρείται ακριβώς τότε προσθέτουμε ένα ακόμη chunk το οποίο δε θα είναι ολόκληρο από pieces, οπότε όσα pieces είναι άδεια θα γεμίζουν με μηδενικά . Στο σημείο αυτό είναι εύλογο το ερώτημα τι γίνεται έαν κάποιος απο τους host ο οποίος είναι υπέυθυνος για κάποιο απο τα piece του chunk για κάποιο λόγο γίνει μη διαθεσιμος ; Για το λόγο αυτό χρησιμοποιούμε redundancy στα pieces το οποίο δημιουργείται με τη βοήθεια του αλγορίθμου Solomon-Reed.[2]. Η λειτουργία του Solomon-Reed αλγορίθμου είναι να δημιουργεί κάποια επαυξημένα pieces , έτσι ώστε να μπορούμε να ανακτήσουμε το αρχέιο μας από τον αρχικό αριθμό pieces. Στην εικόνα, το chunk του αρχείου μας έχει χωριστεί σε τέσσερα pieces (4 bytes/piece). Τα pieces αναπαρίστανται από ένα τετραγωνικό πίνακα 4χ4 , όπου κάθε σειρά του πίνακα αναπαριστά ένα piece. Ο αλγόριθμος solomon-reed δημιουργεί έναν κωδικοποιημένο πίνακα ο οποίος πολλαπλασιάζεται με τον πίνακα data και προκύπτει ο νέος επαυξημένος πίνακας με τις νεες σειρές απο τα pieces τα οποία ονομάζονται parity pieces. Στο συγκεκριμένο παράδειγμα. το αρχείο μας μπορεί να ανακτηθεί εαν έχουμε στη διάθεσή μας τέσσερα απο τα έξι pieces. Μπορούμε να δημιουργήσουμε όσα parity pieces θέλουμε.[4]

Παρακάτω παραθέτουμε ένα παράδειγμα, προκειμένου να εξηγήσουμε το τρόπο με τον οποίο χωρίζουμε το αρχείο μας . Θεωρούμε ότι χωρίζουμε το κάθε chunk μας σε τρία pieces . Με τη βοήθεια του αλγορίθμου solomon-reed, μέσω της διαδικασίας που περιγράψαμε παραπάνω, δημιουργούμε τρία parity pieces που εξυπηρετούν το σκοπό του redundancy. Επομένως έχουμε συνολικά έξι pieces για κάθε chunk του αρχείου μας. Εν συνεχεία, κάθε ένα εκ των έξι pieces θα μοιραστεί σε ένα διαφορετικό host.

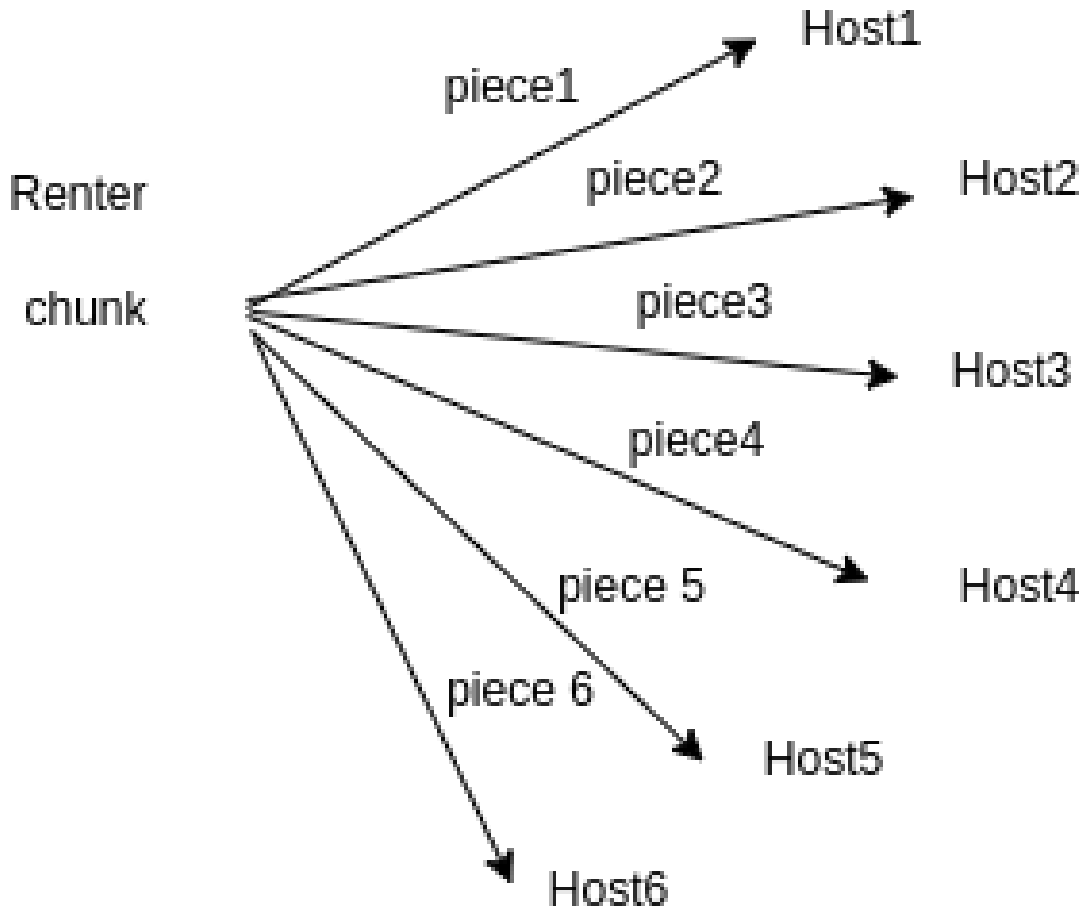

FIGURE 3.4: Διαμοιρασμός pieces σε hosts

3.0.3 Απόδειξη κατοχής δεδομένων απο host

Στη συνέχεια του κειμένου μας μπορούμε να αναφερόμαστε σε κάθε piece και ως sector. Όπως αναλύσαμε και παραπάνω ένας host είναι υπεύθυνος για την αποθήκευση ενός piece(sector). Ένας host εκτός από το να αποθηκέυσει το εκάστοτε piece , έχει ως λειτουργία τον υπολογισμό του merkle Root για το συγκεκριμένο piece.

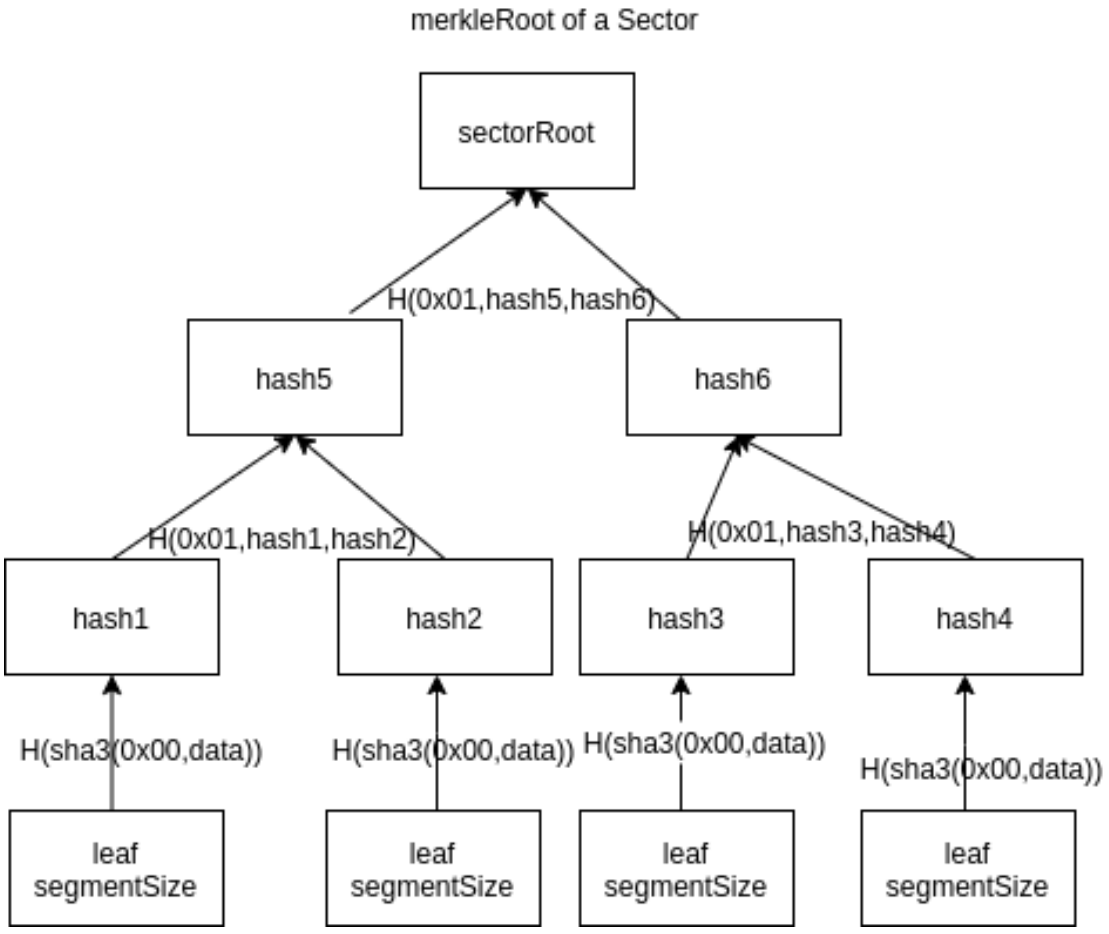

Figure 3.5: Merkle Tree of a sector

Το παραπάνω σχήμα αποτελεί το merkle Tree , δηλαδή τη δομή μέσω της οποίας υπολογίζουμε το merkle Root. Ο υπολογισμός του merkle Root, βασίζεται στην κρυπτογραφική δομή του merkle tree. [1].Το merkle tree αποτελεί ένα δυαδικό δένδρο. Το piece μας χωρίζεται και αυτό με τη σειρά του σε μικρότερα κομμάτια , τα οποία ονομάζουμε segments. Τα segments μας θέλουμε να είναι ίσα σε μέγεθος(bytes). Όπως γίνεται φανερό και μέσω του σχήματος , τα segments αποτελούν τα φύλλα του δέντρου μας (merkle tree). Στο σχήμα γίνεται φανερή η διαδικασία που ακολουθείται για τον υπολογισμό του merkle root του sector.Αρχικά υποβάλλουμε κάθε φύλλο από μια συνάρτηση κατακερματισμού (hash function).

```
H(sha3(0x00,data))
```

> A hash function is any function that can be used to map data of arbitrary size to fixed-size values.

Hash function

Εμείς χρησιμοποιήσαμε τη hash function sha3-256 η οποία έχει πάντα σαν έξοδο 256 bits. Επιλέξαμε τη συγκεκριμένη, επειδή χρησιμοποιείται και απο τη solidity και τα smart contracts κατ'επέκταση.SHA-3. Αφού υποβάλλουμε κάθε φύλλο από τη συνάρτηση κατακερματισμού μας SHA-3 ο αλγόριθμος έχει ως εξής . Κάθε κόμβος ο οποίος δεν είναι φύλλο, είναι το αποτέλεσμα του hash των δύο κόμβων παιδιών του σύμφωνα με τον τύπο :

```
H(sha3(0x01,left sibling sum, right sibling sum))
```

Εφαρμόζοντας την παραπάνω διαδικασία, καταλήγούμε σε ένα root 256 bits για το sector το οποίο είναι μοναδικό .

Στην παραπάνω κρυπτογραφική δομή του merkle tree βασίζεται η διαδικασία του proof of storage, δηλαδή η απόδειξη απο πλευράς του host στο renter ότι πράγματι διατηρεί τα δεδομένα του αποθηκευμένα, όπως είχε συμφωνηθεί. Παραθέτουμε μια εικόνα η οποία περιέχει ένα παράδειγμα δημιουργίας ένος τέτοιου proof, την οποία και εξηγούμε.

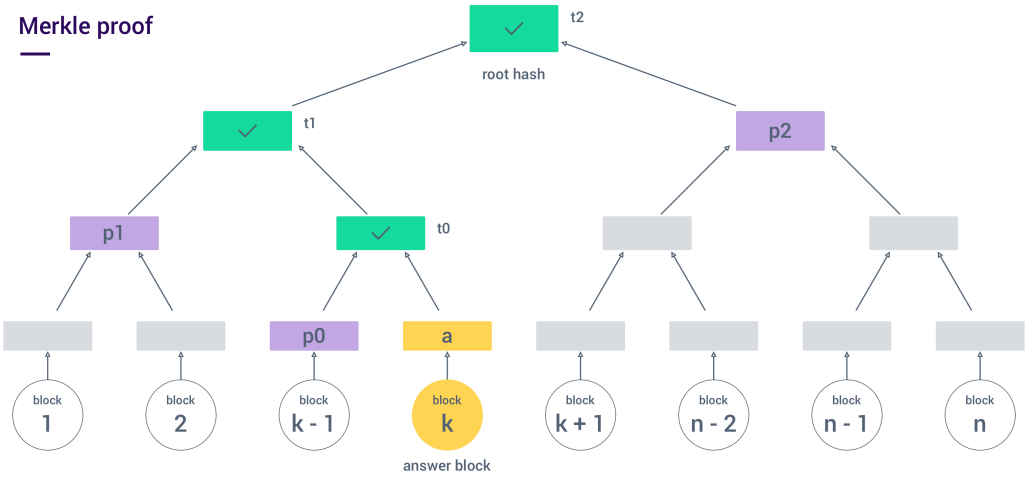

FIGURE 3.6: Merkle Proof of a sector

source

To proof of storage επιτυγχάνεται με τη βοήθεια των merkle proof challenges. Στην παραπάνω εικόνα, τα φύλλα του δέντρου αποτελούν τα sements ενός piece , τα οποία ο host έχει αποθηκευμένα. Επομένως κάθε φύλλο (segment) αποτελείται απο segment size αριθμό bits. Όπως είναι φανερό από την εικόνα, από το host ζητείται να αποδείξει ότι κατέχει στη διάθεσή του το κ-οστό φύλλο (segment) του piece. Επομένως το merkle proof θα αποτελείται απο τα στοιχεία k, p0, p1, p2.

```
H(sha3(0x00, k-data)) -> a

H(sha3(0x01,p0,a)) -> t0

H(sha3(0x01,p1.t0)) -> t1

H(sha3(0x01,t1,p2)) -> t2
```

Επομένως μέσω του merkle proof καταλήξαμε στο t2 το οποίο είναι το root του sector μας και είναι μοναδικό, επομένως αυτή είναι μια απόδειξη ότι ο host πράγματι έχει το δεδομένο

κ αποθηκευμένο. Αξίζει να τονίσουμε στο συγκεκριμένο σημείο ότι κάποιος κακόβουλος host θα μπορούσε να έχει διαγράψει το δεξί μέρος του δέντρου και να έχει αποθηκέυσει μόνο τα roots που του χρειάζονται για να δημιουργήσει το ζητούμενο merkle proof. Το ζήτημα αυτό μπορεί να αντιμετωπιστεί μόνο επιλέγοντας συνεχώς διαφορετικά φύλλα προς απόδειξη τα οποία να είναι καλά απλωμένα για να καλύπτουν μεγάλο εύρος του δέντρου.

Επομένως στην περίπτωση που ένας host διατηρεί αποθηκευμένα ν(έστω για το παράδειγμά ότι ν = 3) τον αριθμό sectors και του ζητηθεί merkle proof ένα συγκεκριμένο segment του δεύτερου sector, το merkle proof θα εχέι την παρακάτω εικόνα

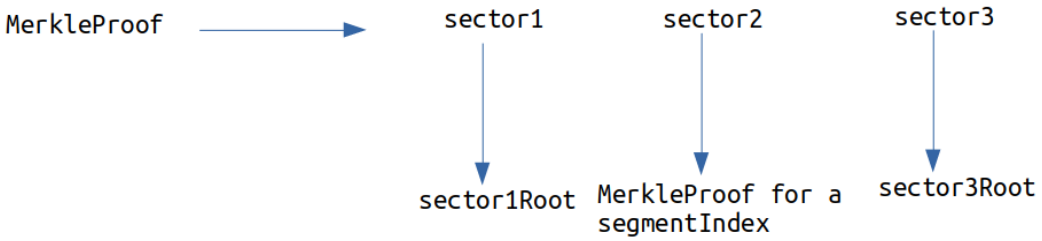

FIGURE 3.7: Merkle Proof Εποπτικά

Παρακάτω παραθέτουμε τις καταστάσεις στισ οποίες μπορεί να περιέλθει ένα storage contract στη διάρκεια ενός κύκλου ζωής.

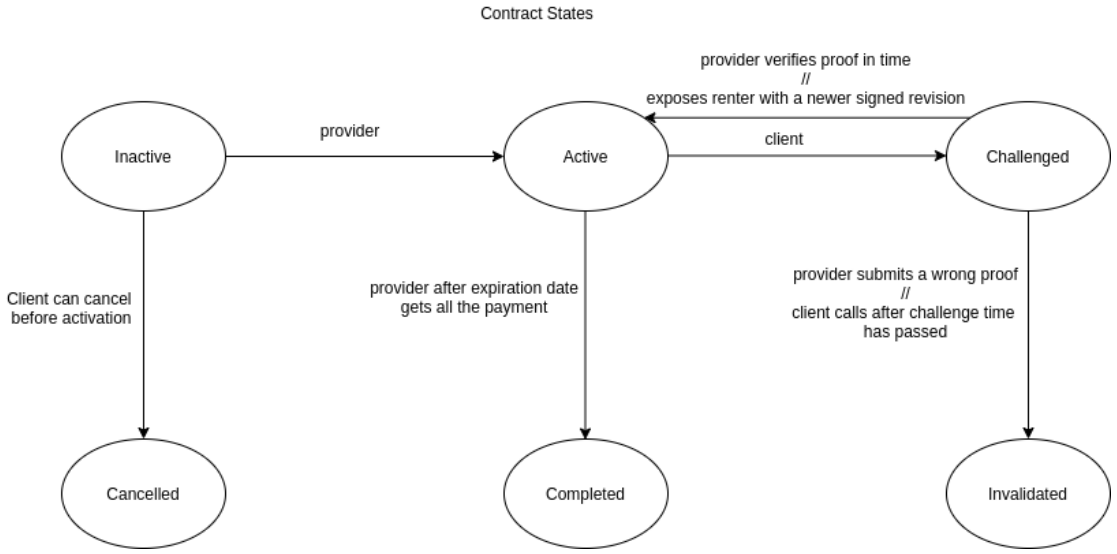

FIGURE 3.8: Διάγραμμα καταστάσεων ενός Storage Contract

Αξίζει να σταθούμε στην μετάβαση από Active σε Challenged και τι γίνεται μετέπειτα απο Challenged . Ένα storage contract μπορεί να μεταβεί από active σε challenged μέσω του renter , ο οποίος μέσω του contract call challenge provider κάνει challenge το host να αποδείξει ότι κατέχει ένα συγκεκριμένο segment. Από challenge ένα storage contract μπορεί να μεταβεί σε δύο καταστάσεις τις οποίες και εξηγούμε :

- *Challenged - Active*: Μπορεί να μεταβεί με δύο τρόπους

  1) Εαν ο host υποβάλλει εντός του προβλεπόμενου χρόνου το merkle proof.
  2) Εαν ο host υποβάλλει μια συμφωνία η οποία είναι μεταγενέστερη και αποδεικνύει πως ο renter έχει χρησιμοποιήσει προγενέστερη συμφωνία για τη δημιουργία του merkleProof challenge.

- *Challenged - Invalidated*: Επίσης δυνάται να μεταβούμε με δύο τρόπους

  1) Εάν ο host υποβάλλει μια λάθος merkle proof , τότε χάνει όλα τα χρήματα που έχει κερδίσει και το storage contract γίνεται invalidated.
  2) Ο host δεν απαντάει καθόλου στο χρονικό πλαίσιο που του έχει ζητηθεί και ο renter καλεί το contract call το οποίο το επιβεβαιώνει και το storage contract , γίνεται invalidated.

Chapter 4

Υλοποίηση

Στο παρόν σύστημα υλοποιήσαμε ένα node, ο οποίος μπορεί να είναι είτε renter, είτε host . Ο node αυτός αποτελεί ένα deamon, ο οποίος είναι υλοποιημένος σε golang. Τόσο ο host όσο και ο renter μπορεί να επικοινωνεί με τον αντίστοιχό του server (Host-Server, Renter-Server) ο οποίος είναι γραμμένος σε nodejs. Η υλοποίηση αυτου του renter σε node javascript, εξυπηρετεί τη χρήση του web3 api, το οποίο αποτελεί μια συλλογή βιβλιοθηκών για επικοινωνίά με ένα remote ή τοπικο blockchain με τη χρήση htppRequests, ipcs, websockets. web3-api.
Παρακάτω παραθέτουμε μια εικόνα η οποία περικλείει τα παραπάνω.

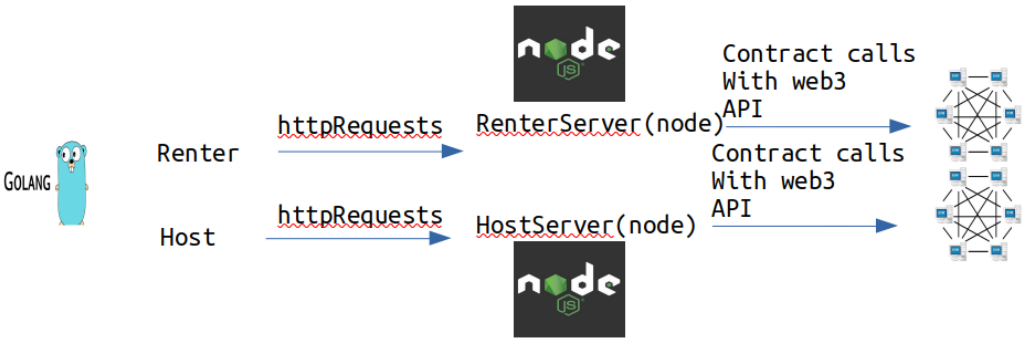

FIGURE 4.1: Υλοποίηση Κόμβου

4.0.1 Τεχνολογίες

4.0.1.1 Truffle

Για την υλοποίηση και το deploy των smart contracts στο τοπικό blockchain , χρησιμοποιήσαμε το truffle, το οποίο είναι το πιο διαδεδομένο framework για το deploy και το compile των smart contracts. Το truffle χρησιμοποιείται από πολλές εταιρείες , όπως είναι η amazon, microsoft και άλλες.

> Truffle takes care of managing your contract artifacts so you don't have to. Includes support for custom deployments, library linking and complex Ethereum applications.

truffle suite

Το truffle όταν κάνει compile ένα smart contract , παράγει εκτός από το bytecode και ενα json αρχείο.

4.0.1.2 Ganache

Επιπλέον, χρησιμοποιήσαμε το Ganache , προκειμένου να στήσουμε ένα τοπικό ethereum blockchain. Το Ganache μάλιστα περιέχει και γραφικό περιβάλλον προκειμένου να μπορούμε να δούμε την κατάσταση των account συμπεριλαμβανομένων του address, private key, καθώς και τα transactions αλλά και το balance κάθε account. Επιπροσθέτως είναι δυνατόν να δούμε κάθε μπλοκ που έχει γίνει mine τι ακριβώς περιέχει απο transactions, καθώς και εσωτερικά απο τι αποτελούνται τα transactions. Επιπροσθέτως μπορούμε να δούμε όλα τα smart contracts που έχουν γίνει deploy στο blockchain, καθώς και τι events έχουν παραχθεί από κάθε smart contract. ganache.Παρακάτω παραθέτουμε μια εικόνα απο το γραφικό περιβάλλον (gui) , που προσφέρει το ganache σε ένα χρήστη.

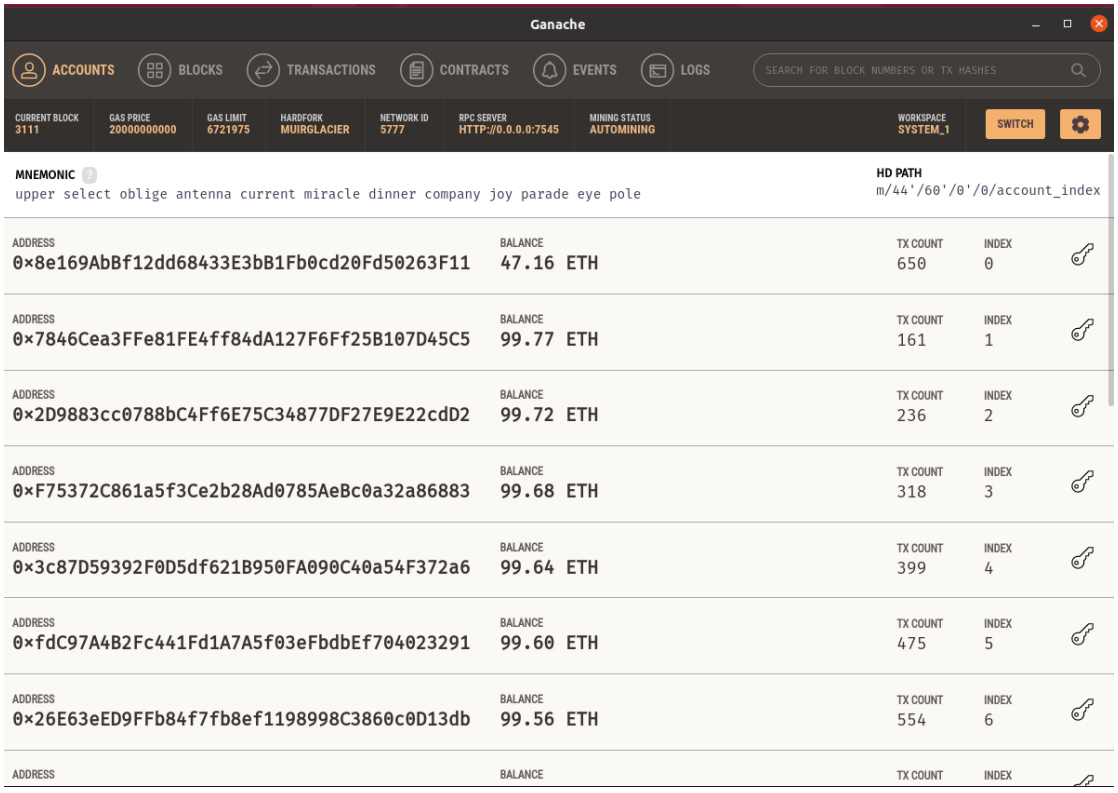

FIGURE 4.2: Γραφικό Περιβάλλον Ganache

4.0.2 Υλοποίηση - Σενάρια Χρήσης Συστήματος

4.0.2.1 Βάση Χρηστών

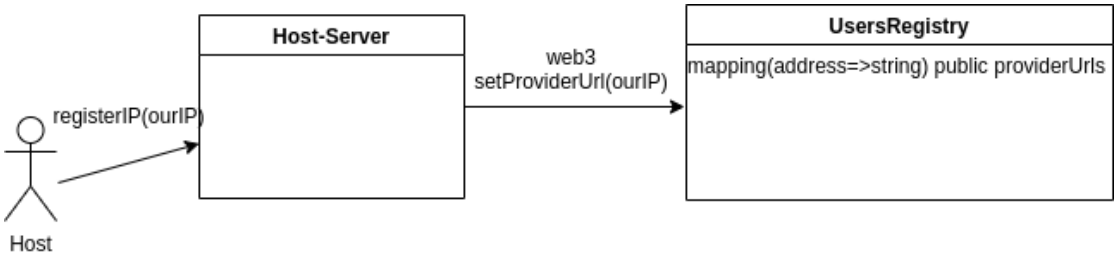

FIGURE 4.3: Βάση Χρηστών

Κάθε host θα πρέπει να εγγραφεί στο users Registry το οποίο αποτελεί ένα smart contract. Στην εικόνα φαίνονται οι κλήσεις που γίνονται, αρχικά ο host στέλνει ένα httpRequest στο Host-Server στο οποίο περιλαμβάνεται η IP του host . Εν συνεχεία ο Host-Server καλεί μέσω του web3 api το contract call setProvider(ourIP), προκειμένου να προστεθεί το public key του host μαζί με την IP του, στο map providerURls το αντιστοιχίζει ένα public key ενός host με μία IP.

4.0.2.2 Δημιουργία Δημοπρασίας

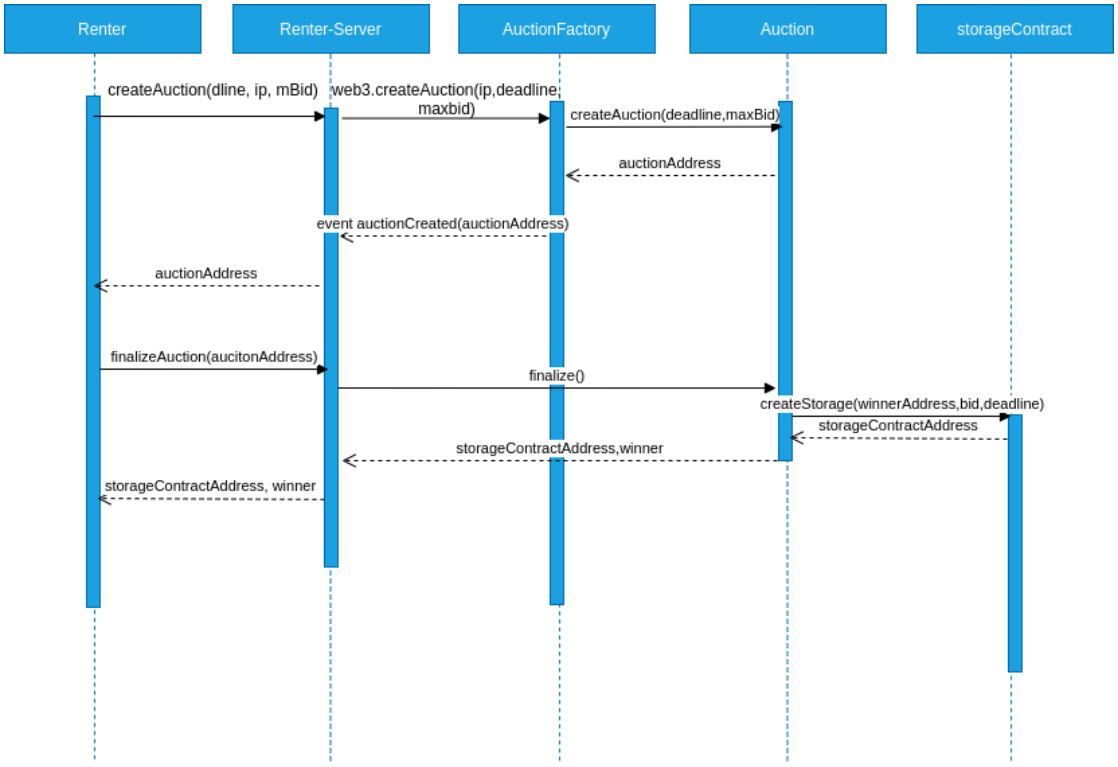

FIGURE 4.4: Δημιουργία Δημοπρασίας

Η διαδικασία της δημιουργίας μιας δημοπρασίας ,συμβαίνει όπως αναφέραμε προηγουμένως και τα βήματα αναλυτικά φαίνονται στην παραπάνω εικόνα. Η διαδικασία εκκινείται απο το Renter . Ο Renter έχει ως στόχο τη δημιουργία τόσων δημοπρασιών και κατ'επέκταση storage contract όσο είναι το σύνολο των pieces μαζί με το erasure coding. Από το σχήμα φαίνονται οι κλήσεις που γίνονται από το renter node μας στον αντίστοιχο Renter-Server, καθώς και τα αντίστοιχα contract calls τα οποία κάνει ο Renter-Server. Από το Renter node μας στέλνεται με httpRequest το create Auction το οποίο περιλαμβάνει στο url τα deadline, maxBid . Τα ορίσματα αυτά είναι η χρονική διάρκεια που θέλει ο renter να διαρκέσει ένα storage contract και το maximum Bid, που είναι διατεθειμένος να δεχτεί ως προσφορά από κάποιον host. Έπειτα, ο Renter-Server καλεί το contract call createAuction(deadline, maxBid) του smartContract Auction Factory και εκείνο με τη σειρά του αρχικοποιεί ένα νέο smart contract το Auction με τα ορίσματα deadline και maxBid. Έπειτα ο Renter-Server, ο οποίος πάιρνει την διεύθυνση του auction που δημιούργησε και την επιστρέφει στο node Renter μέσω ενός http Response. Ο node Renter κάνει finalize το auction , στέλνοντας με httpRequest στο Renter-Server και εκείνος καλεί το αντίστοιχο contract call auctionFinalize στο contract auction , το οποίο επιστρέφει στο Renter-Server το

public key του νικητή host, δηλαδή εκείνου με την πιο συμφέρουσα προσφορά μέχρι εκείνη τη στιγμή. Η παραπάνω διαδικασία από τη μεριά του renter έχει παραλληλοποιηθεί για τη δημιουργία όλων των δημοπρασιών.

4.0.2.3 Bid Δημοπρασίας

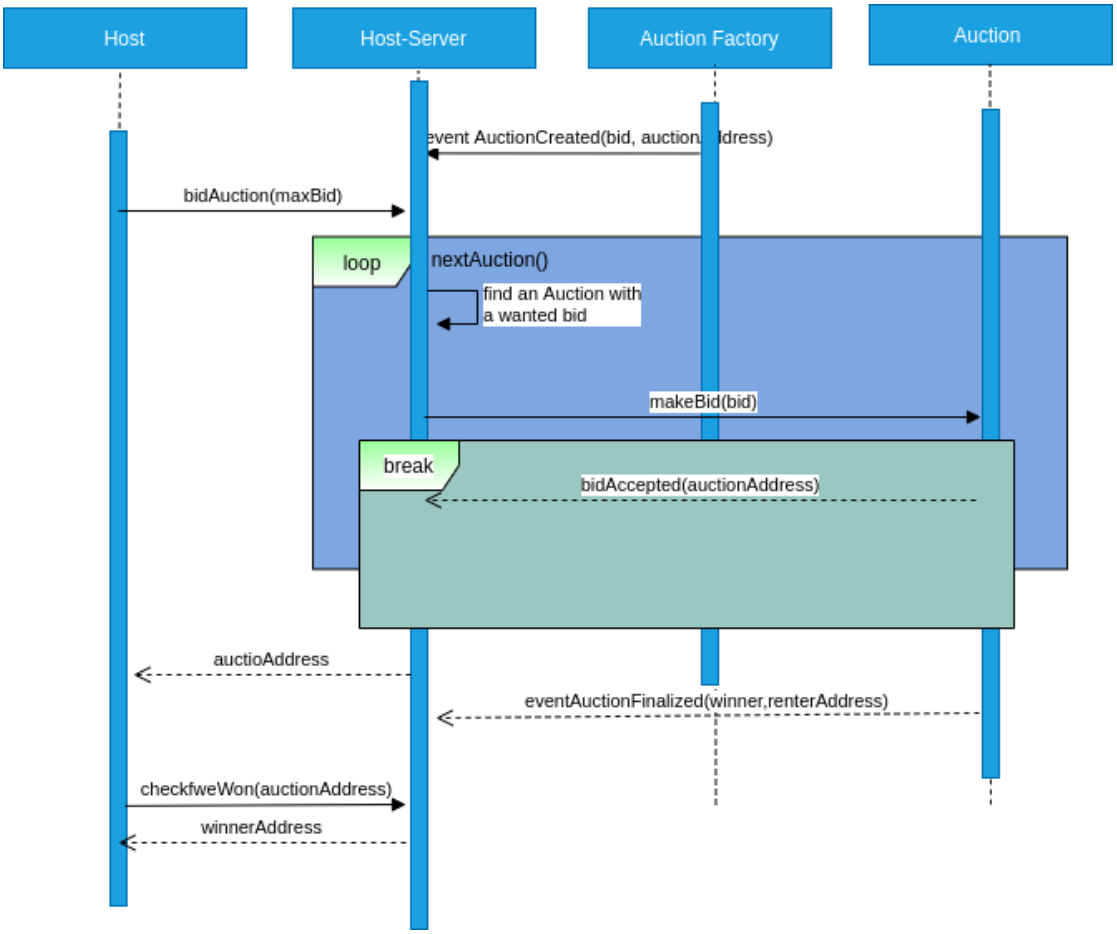

FIGURE 4.5: Bid Δημοπρασίας

Στο παραπάνω διάγραμμα , φαίνεται πως ο host κάνει bid σε κάποιο τυχαίο auction. Πρέπει να αναφέρουμε ότι ο Host-Server από τη στιγμή της δημιουργίας του, ακούει όλα τα events από το auction Factory, τα οποία περιέχουν κάθε καινούρια δημοπρασία που δημιουργείται και το bid το οποίο ορίζει ως τιμή εκκίνησής της. Ο Host εκκινεί τη διαδικασία στέλνοντας ένα httpRequest στο Host-Server , προκειμένου να κάνει bid σε ένα auction, ορίζοντας στο url του HttpRequest το bid που είναι διατεθειμένος να κάνει. Ο Host-server μπαίνει σε ένα loop στον οποίο αναζητά μια auction που να περιλαμβάνει το επιθυμητο bid. Οταν τη βρεί κάνει το contract call σε αυτή makeBid(). Εαν το bid γίνει accepted τότε βγαίνει απο το loop, διαφορετικά συνεχίζει την αναζήτηση για την εύρεση δημοπρασίας. Όταν

τελικά καταφέρει να κάνει bid σε μια δημοπρασία βγαίνει απο το loop και ενημερώνει το host με httpResposonse στο οποίο περιλαμβάνει τη διεύθυνση του auction. Στο μεταξύ εαν ο host όπως στο παράδειγμά μας δεν επιλέξει να κάνει κάτι, στο τέλος της δημοπρασίας ο Host-Server ενημερώνεται απο το event auctionFinalized(winner) για το ποιος είναι ο νικητής της δημοπρασίας. Ο host επιλέγει μετά το πέρας του auction deadline να στείλει μέσω httpRequest στο Host-Server προκειμένου να δει ποιος κέρδισε τη δημοπρασία. Ο Host-Server , του απαντάει μέσω HttpResponse , ποιος είναι ο νικητής της δημοπρασίας (winner Address).

4.0.2.4 Upload Αρχείου

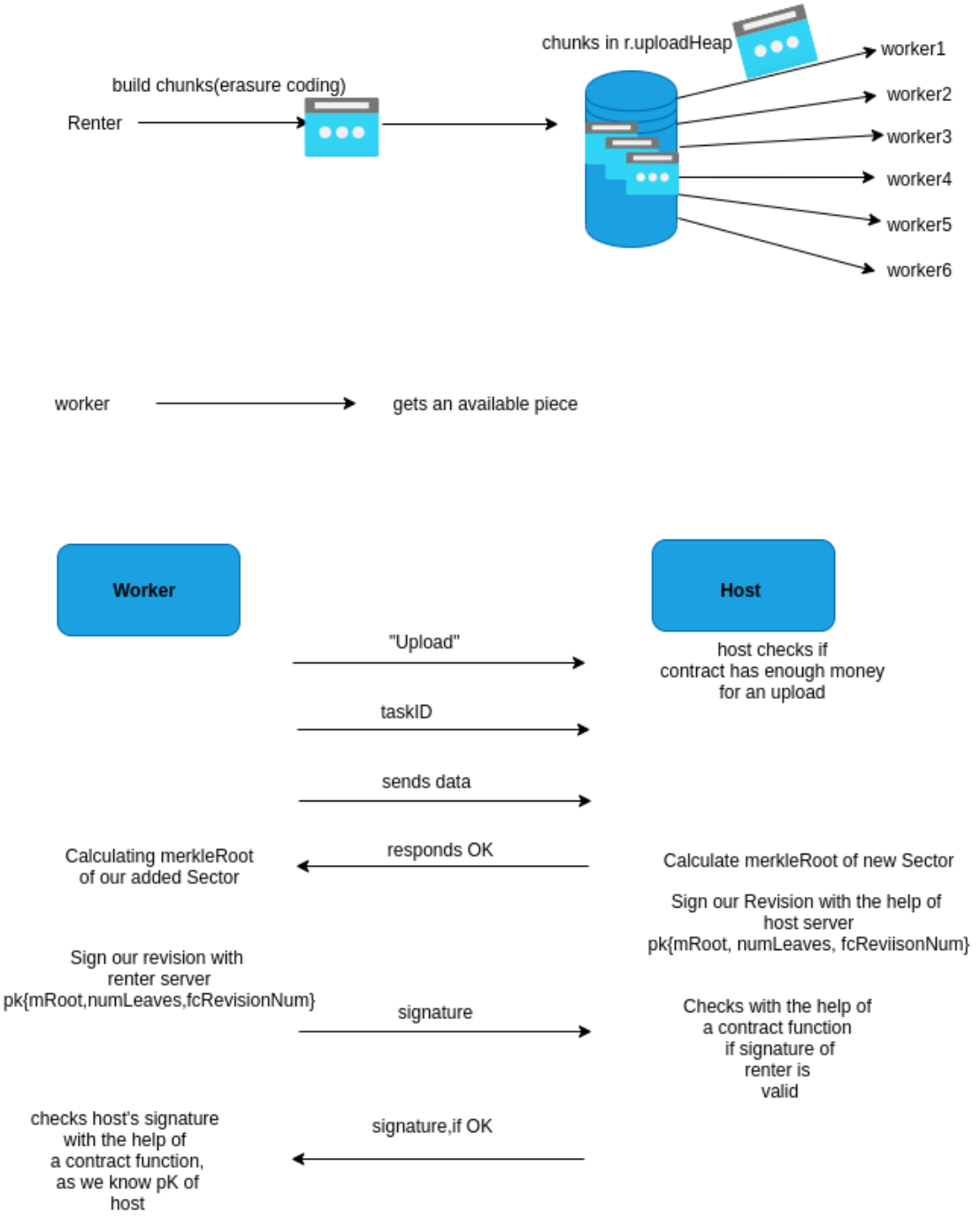

Figure 4.6: Upload Αρχείου

Η διαδικασία του upload ενός αρχείου, φαίνεται από την παραπάνω εικόνα, την οποία και θα εξηγήσουμε αναλυτικά. Αρχικά, ο renter χωρίζει το file σε chunks. Για κάθε chunk , φτιάχνει τα pieces του συμπεριλαμβανομένων και των pieces που προκύπτουν απο το redundancy. Στην εικόνα θεωρούμε ότι κάθε chunk αποτελείται απο 3 pieces , ενω με το redundancy, προσθέτουμε άλλα τρία pieces, με αποτέλεσμα να καταλήγουμε να έχουμε έξι pieces συνολικά. Έπειτα, κάθε chunk που δημιουργείται στέλνεται στη heap του. Από εκεί κάθε chunk μοιράζεται στους workers, οι οποίοι είναι έξι, δηλαδή όσα pieces έχουμε συνολικά. Οι workers ξεκινούν παράλληλα να επεξεργάζονται το chunk, παίρνοντας ο κάθε ένας από ένα διαθέσιμο piece, απο τα ήδη υπάρχοντα. Κάθε worker μιλά με έναν συγκεκριμένο host και αναλαμβάνει τη διαδικασία του upload του piece αυτού στο συγκεκριμένο host. Εν συνεχεία παρουσιάζεται το πρωτόκολλο του upload ενός piece μεταξύ ενός worker και ενός host.
Αρχικά, ο worker στέλνει μήνυμα "upload", δηλώνοντας την πρόθεση του στο host να του στείλει κάποιο piece προς αποθήκευση. Ο host , εν συνεχεία μέσω ενός contract call με τη βοήθεια του Host-Server ελέγχει εαν το StorageContract διαθέτει τα χρήματα προκειμένου να γίνει το upload. Εν συνεχεία ο renter στέλνει στο host το taskID και το piece . Εάν ο host επιβεβαιώσει ότι το storageContract περιέχει τα χρήματα που χρειάζονται για το upload, ο host στέλνει ok στο renter. Έπειτα υπολογίζει το merkle Root για το συγκεκριμένο sector, καθώς και το ως τώρα mekleRoot όλων των sectos που έχει αποθηκευμένα. Έπειτα υπογράφει το νεό fcRevision , το οποίο αποτελείται απο :

- *merkle Root*: Είναι το merkle root όλων των sectors εως εκείνη τη στιγμή , συμπεριλαμβανομένου του νέου sector.
- *num Leaves*: Περιλαμβάνει όλα τα φύλλα , δηλαδή τα semgments τα οποία έχει αποθηκευμένα.
- *fcRevisionNum* : Αποτελεί ένα ενδεικτικό για την αρίθμηση των fc revisions.

Ο Renter αντίστοιχα υπογράφει και εκείνος το fc revision με τα ίδια ακριβώς πεδία και στέλνει την υπογραφή του στο host. Ο host ελέγχει με τη βοήθει του Host-Server και πιο συγκεκριμένα μιας μεθόδου του storage contract (η οποία επειδή δεν αλλάζει το state του Ethereum , δεν έχει κάποιο κόστος), εαν πράγματι η υπογραφή αυτή ανήκει στο Renter. Εαν πράγματι ανήκει στο Renter αποθηκεύει το sector και στέλνει και εκείνος με τη σειρα του την υπογραφή του στο Renter. Ο Renter με τη σειρά του απευθυνόμενος στο Renter-Server και εκείνος εκτελώντας την ίδια μέθοδο ελέγχει εαν η υπογραφή είναι πράγματι του host , δηλαδή αντιστοιχεί στο public key του.

4.0.2.5 Challenge

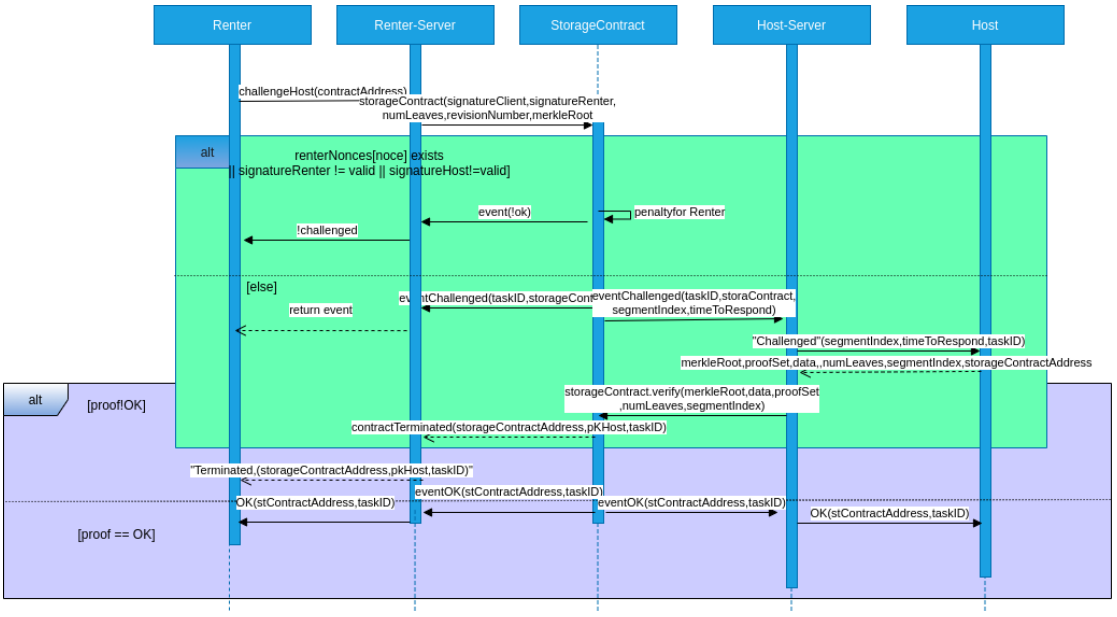

FIGURE 4.7: Challenge

Στην παραπάνω εικόνα, φαίνεται πως ένας renter υποβάλλει σε challenge ένα host. Αρχικά ο renter στέλνει στο Renter-Server ένα httpRequest όπου στο url challengeHost συμπεριλαμβάνει την υπογραφή του, την υπογραφή του host, τον αριθμό των συνολικών φύλλων, τον αριθμό του fcRevision καθώς και το merkleRoot των συνολικών sectors εως εκείνη τη στιγμή. Ο Renter-Server καλεί με ένα contract call με τη βοήθεια του web3 api την challengeProvider() με τα αντίστοιχα ορίσματα που αναφέραμε προηγουμένως. Αρχικά απο το storage Contract ελέγχεται εαν έχει ήδη ζητηθεί το συγκεκριμένο nonce από το renter. Εαν έχει ήδη ζητηθεί τότε το challenge δε γίνεται δεκτό και επιβάλλεται penalty στο renter. Επιπλέον ελέγχονται οι υπογραφές των renter και host εαν πράγματι έχουν υπογράψει το συγκεκριμένο file contract revision. Στην περίπτωση που είναι λάθος έχουμε επίσης απόρριψη του challenge απο το contract. Διαφορετικά , η συνάρτηση challengeProvider υπολογίζει ένα segment Index και κάνει public το event του Challenge . Το event αυτό το ακούει ο Renter-Server ο οποίος με τη σειρά του ενημερώνει το renter. Επιπλέον το ίδιο event ακούει ο Host-Server του συγκεκριμενου host(ο οποίος ακούει για event στη συγκεκριμενη διεύθυνση του storage contract από τη δημιουργία του contract), ο οποίος ενημερώνει το host για το συγκεκριμένο challenge.Εδώ θεωρούμε ότι ο host απαντάει εντός χρονικού ορίου του challenge. Ο host στέλνει το merkle proof στο host-server και εκείνος καλει με contract call τη verify για το συγκεκριμένο storageContract. Σε περίπτωση που η verify αποφανθεί ότι το merkle proof του host δεν είναι έγκυρο το storage contract τερματίζεται και ανακοινώνεται μέσω event το οποίο ακούει ο Renter-Server

και κατ'επέκταση ο ίδιος ο renter, ενώ ο host χάνει το collateral που είχε βάλει στην αρχή , καθώς και όσα χρήματα είχε συγκεντρώσει ως τότε απο τα uploads των data. Αντίθετα στην περίπτωση που το merkle proof του host είναι έγκυρο, μεταφέρονται χρήματα για τη σωστή απόδειξη από το renter στο host, ενώ ανακοινώνει το event verified. Το event αυτό το ακούει τόσο ο Renter-Server, όσο και ο Host-Server (αντίστοιχα οι nodes Renter και Host).

Chapter 5

Μελλοντικές Προεκτάσεις

Στο κεφάλαιο αυτό αναλύουμε μελλοντικές προσθήκες που μπορουν να γίνουν στο σύστημά μας.

- *Ανανέωση Storage Contract* : Θα ήταν ενδιαφέρον να υλοποιηθεί μελλοντικά ένας μηχανισμός, όπου πριν λήξει το storage contract , να μπορεί ο renter να το ανανεώνει επικοινωνόντας με το host off chain.
- *Μηχανισμός εύρεσης νέων host* :
- *Προσθήκη περισσότερων κριτηρίων κατά τη διάρκεια της δημοπρασίας* : Θα μπορούσε να υπάρχει μια βαθμολογία για κάθε host, η οποία θα ήταν το αποτέλεσμα της ψηφοφορίας κάθε renter, μετά το πέρας του storage contract. Εάν ο renter ήταν ικναοποιημένος από τις υπηρεσίες του host, τότε θα τον αξιολογούσε θετικά, αυξάνοντας την αξιοπιστία του host.

Βιβλιογραφία

[1] Merkle tree. https://en.wikipedia.org/wiki/Merkle_tree.

[2] Reed-solomon error correction. https://en.wikipedia.org/wiki/Reed\OT1\textendashSolomon_error_correction.

[3] Ethereum virtual machine. https://ethereum.org/en/developers/docs/evm/, 2021.

[4] Backblaze. Backblaze open-sources reed-solomon erasure coding source code, 2015.

[5] Joe. Baguley. *How cloud computing is changing the world ... without you knowing*. https://www.reportlinker.com/p05749258/Cloud-Computing-Market-by-Service-Deployment-Model-Organization-Size-Workload-Ver html, September 2013.

[6] canalys. Cloud infrastructure spend grows 46% in Q4 2018 to exceed US$80 billion for full year. https://canalys-com-public-prod.s3.eu-west-2.amazonaws.com/static/press_release/2019/pr20190204.pdf.

[7] Jeff. Elder. 70% *Companies Storing Data have been Hacked*. https://www.businessinsider.com/cloud-computing-hacked-cybersecurity-sophos-amazon-2020-7, 9 July 2020.

[8] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf, 2008.

[9] Melanie Swan. Blueprint for a new economy, 2015.

[10] vasa. Getting deep into ethereum : How data is stored in ethereum. https://hackernoon.com/getting-deep-into-ethereum-how-data-is-stored-in-ethereum-e3f669d96033?ref=hackernoon.com, 2018.

[11] vasa. Getting deep into evm: How ethereum works backstage. https://medium.com/swlh/getting-deep-into-evm-how-ethereum-works-backstage-ab6ad9c0d0bf, 2019.

[12] Buterin Vitalik. Ethereum whitepaper. https://whitepaper.io/document/5/ethereum-whitepaper, 2013.

[13] Hongyi Wang, Qingfeng Jing, Bingsheng He, Zhengping Qian, and Lidong Zhou. *Distributed systems meet economics: pricing in the cloud*, 2010.

[14] Dr. Gavin Wood. Ethereum yellow paper. https://ethereum.github.io/yellowpaper/paper.pdf, 2021.