Εθνικό Καποδιστριακό Πανεπιστήμιο Αθηνών

Τμήμα Μαθηματικών

Μεταπτυχιακό Πρόγραμμα στη Λογική και Θεωρία Αλγορίθμων και Υπολογισμού

μΠλ∀

Έξυπνα συμβόλαια και πληρωμές χρησιμοποιώντας Bitcoin και Ethereum

---

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αικατερίνη-Παναγιώτα Στούκα

Επιβλέπων: **Αριστείδης Παγουρτζής,** Αναπληρωτής Καθηγητής ΕΜΠ

Αθήνα, Μάιος 2015

Η παρούσα Διπλωματική Εργασία
εκπονήθηκε στα πλαίσια των σπουδών
για την απόκτηση του
Μεταπτυχιακού Διπλώματος Ειδίκευσης
στη
Λογική και Θεωρία Αλγορίθμων και Υπολογισμού
που απονέμει το
Τμήμα Μαθηματικών
του
Εθνικού και Καποδιστριακού Πανεπιστημίου Αθηνών

Εγκρίθηκε την …… από Εξεταστική Επιτροπή
αποτελούμενη από τους:

Ονοματεπώνυμο	**Βαθμίδα**	**Υπογραφή**
1. ……...	…….	……
2. ……...	…….	……
3. ……...	……..	……

ΠΕΡΙΛΗΨΗ

Στην παρούσα διπλωματική μελετάμε το ψηφιακό νόμισμα Bitcoin ,το οποίο προτάθηκε από τον Satoshi Nakamoto [3] το 2008 ,ως προς τον τρόπο λειτουργίας του και ως προς την ασφάλεια του [1],[2] . Το Bitcoin αποτελεί ένα κρυπτοσυνάλλαγμα το οποίο χαρακτηρίζεται για τον αποκεντρωτικό του χαρακτήρα ,δεδομένου ότι δεν υπάρχει κάποια κεντρική αρχή κατά τη διάρκεια κοπής νομισμάτων και συναλλαγών .Στη συνέχεια περιγράφουμε ορισμένα συμβόλαια που χρησιμοποιούν το Bitcoin [12] ,για να λύσουν προβλήματα εμπιστοσύνης . Επίσης μελετάμε μία κατασκευή (με την ασφάλεια που μας παρέχει) ,το Zerocoin [22], η οποία μπορεί να εφαρμοστεί στο Bitcoin ,προκειμένου να επιτευχθεί ανωνυμία (προτάθηκε από τους Ian Miers, Christina Garman, Matthew Green, Aviel D. Rubin το 2013), καθώς και τον dynamic accumulator που δημοσιεύθηκε από τους J. Camenisch and A. Lysyanskaya [23],που χρησιμοποιείται στην παραπάνω κατασκευή. Τέλος αναλύουμε τον τρόπο λειτουργίας του Ethereum ,ενός αποκεντρωμένου συστήματος που χρησιμοποιεί μία Turing complete γλώσσα , με σκοπό την διευκόλυνση δημιουργίας συμβολαίων ,όπως έχει προταθεί από τους Vitalik Buterin και Gavin Wood στο [20],[21] το 2013 και το 2014 αντίστοιχα.

ABSTRACT

This Thesis focuses on Bitcoin ,a digital currency, which was proposed by Satoshi Nakamoto [3] in 2008 . We describe both its function and its security [1],[2]. Bitcoin is a cryptocurrency, which is decentralized, as there is not any trusted authority that mints coins or controls transactions. Consequently, we describe some smart contracts [12], which use Bitcoin to solve problems of trust. In addition, we study a construction (and its security) , Zerocoin [22], which can be implemented in Bitcoin in order to achieve anonymity (it was proposed by Ian Miers, Christina Garman, Matthew Green, Aviel D. Rubin in 2013) ,as well as a dynamic accumulator published by J. Camenisch and A. Lysyanskaya [23] that is used in the above construction . Finally, we analyze Ethereum 's function .Ethereum is a decentralized system that uses a Turing complete language in order to make the contract creation easier and was proposed by Vitalik Buterin and Gavin Wood [20],[21] in 2013-2014.

Ευχαριστίες

Αρχικά θα ήθελα να ευχαριστήσω την τριμελή επιτροπή ,
τον κ .Ευστάθιο Ζάχο, τον κ. Άγγελο Κιαγιά και τον
κ. Αριστείδη Παγουρτζή , οι οποίοι με βοήθησαν να διαμορφώσω το γνωστικό μου υπόβαθρο στην κρυπτογραφία ,καθώς και να γνωρίσω τον ερευνητικό τρόπο σκέψης.
Ειδικότερα ευχαριστώ τον επιβλέποντα της διπλωματικής εργασίας ,κ. Αριστείδη Παγουρτζή, για τις συμβουλές και το πολύ καλό κλίμα συνεργασίας.
Ακόμη ήθελα να ευχαριστήσω όλους τους καθηγητές του ΜΠΛΑ που με ενέπνευσαν με τη διδασκαλία τους.
Τέλος ευχαριστώ το Νίκο και την οικογένεια μου για την ανεκτίμητη συμπαράσταση καθ'όλη τη διάρκεια των σπουδών μου .

Περιεχόμενα

1. BITCOIN

Το bitcoin είναι ένα ψηφιακό νόμισμα οι συναλλαγές του οποίου γίνονται σε ένα peer to peer δίκτυο. Σαν κύριο χαρακτηριστικό αυτού του νομίσματος είναι ότι οι συναλλαγές που έχουν πραγματοποιηθεί από την αρχή δημιουργίας του είναι δυνατό να επιβεβαιωθούν και είναι αποθηκευμένες σε μία αλυσίδα από blocks(blockchain),η οποία ουσιαστικά είναι η ιστορία του νομίσματος και είναι υπολογιστικά δύσκολο να τροποποιηθεί από ένα κακόβουλο παίκτη που έχει λιγότερο από 51% CPU power.(Το κάθε block μαζί με άλλα περιέχει καινούριες συναλλαγές.)

Επίσης ένα από τα χαρακτηριστικά το οποίο το κατέστησε άξιο μελέτης ήταν ότι σε όλη την διαδικασία κοπής νομισμάτων και συναλλαγών δεν υπάρχει κάποια κεντρική αρχή .Αντίθετα οποιοσδήποτε μπορεί να κερδίσει ψηφιακά νομίσματα κάνοντας proof of work χρησιμοποιώντας την CPU power του υπολογιστή του (mining).Επιπλέον αξιοσημείωτο είναι ότι όποιος θέλει να κάνει mining για να κερδίσει χρήματα βοηθάει στο να ενσωματωθούν στην blockchain οι συναλλαγές που γίνονται μέσω των transactions και πιστοποιούνται με ψηφιακές υπογραφές.

Ένα από τα σημαντικά προβλήματα που αντιμετωπίστηκε με τη μορφή του bitcoin έτσι όπως προτάθηκε στο **[3]**είναι η αποφυγή το double spending ,δηλαδή να υπάρχουν δύο transactions που στέλνουν τα ίδια νομίσματα σε διαφορετικά άτομα , χωρίς να μπορεί να προσδιοριστεί πιο από τα δύο είναι έγκυρο **.** Η γενική ιδέα του πως αντιμετωπίστηκε είναι ότι η σειρά των blocks της αλυσίδας δεν μπορεί να αλλάξει αφού κάθε block συνδέεται με το προηγούμενο του και αναδρομικά με όλα τα προηγούμενα .

Η μελέτη του Bitcoin λοιπόν έχει πολλά να προσφέρει ,αφού πέρα από ένα ψηφιακό νόμισμα , ο τρόπος λειτουργίας του αποτελεί έμπνευση για πραγματοποίηση και άλλων ενεργειών (συμβολαίων) αποκεντρωμένα.

1.1 TRANSACTION

1.1.1 ΔΟΜΗ TRANSACTION

Το transaction ουσιαστικά αντιπροσωπεύει μία συναλλαγή ,δηλαδή τη μεταφορά ψηφιακών χρημάτων από έναν αποστολέα Α σε έναν παραλήπτη Β.

Η ***γενική μορφή*** ενός transaction **[10],[5]** περιέχει το previous tx,δηλαδή την πηγή των χρημάτων του Α , τα χρήματα που θέλει να στείλει στον Β (VALUE),καθώς και το ζευγάρι scriptsig , scriptpubkey,που αποτελεί το script .Επίσης περιέχει τον αριθμό των inputs και των outputs και την version του transaction. Τέλος περιέχει το locktime και το index.

PREVIOUS TX: περιέχει τα ID των transactions(referenced transactions) στα οποία ήταν ο Α παραλήπτης και λάμβανε τα χρήματα που επιθυμεί τώρα να ξοδέψει .Το ID ενός transaction είναι το hash του υπογεγραμμένου transaction .

***INDEX*:**δείχνει από πιο output του referenced transaction έχει ο Α τα χρήματα

SCRIPTSIG: περιέχει α) την υπογραφή ECDS(ELLIPTIC CURVE DIGITAL SIGNATURE) του αποστολέα Α πάνω σε ένα simplified version του transaction που προφανώς δεν περιέχει τη συγκεκριμένη υπογραφή και β)το public key του Α .Με αυτό τον τρόπο αποδεικνύει ο Α ότι είναι παραλήπτης του transaction που εμφανίζεται στο PREVIOUS TX και ταυτόχρονα αποτρέπει την τροποποίηση του κομματιού που υπέγραψε.

VALUE:Το ποσό που επιθυμεί ο Α να στείλει στο Β σε Satoshi που είναι υποδιαίρεση του BTC(BITCOIN).Ισχύει 1BTC=100000000 SATOSHI

SCRIPTPUBKEY : περιέχει την εντολή που καθορίζει τον τρόπο με τον οποίο μπορεί ο Β να εξαργυρώσει τα χρήματά του και να τα στείλει σε κάποιον άλλον

παραλήπτη με ένα transaction T'. Αυτή η εντολή για παράδειγμα μπορεί να περιέχει μεταξύ άλλων το hash του public key του Β και να επιτρέπει στον Β να εξαργυρώσει τα χρήματά του αν ο Β υπογράψει το νέο transaction T' με το αντίστοιχο private key.

LOCKTIME:δείχνει μετά από πόσο χρονικό διάστημα επιτρέπεται να μπει στο block(χρησιμεύει για συμβόλαια)

Μία μορφή ενός transaction είναι η παρακάτω:

INPUT

Previous tx

Index

Scriptsig

OUTPUT

Value

scriptpubkey

ΠΑΡΑΤΗΡΗΣΕΙΣ

1)Κάθε φορά ο αποστολέας ενός transaction μπορεί να αλλάζει ζευγάρια publc key,secret key .Αυτό βοηθάει στο να μη φαίνεται πόσα bitcoin έχει συνολικά . Βέβαια με κάθε transaction φαίνεται πόσα bitcoin λαμβάνει κάποιος με το συγκεκριμένο public key και όχι το όνομα του κατόχου, αλλά αν με κάποιο τρόπο ,συσχετιστεί το όνομα με το public key τότε παύει η ανωνυμία .Αντίθετα αν κάποιος έχει πολλά public keys είναι πιο δύσκολο να φανερωθεί ο συνολικός αριθμός bitcoin του. Βέβαια θέλει προσοχή ο αποστολέας όταν υπογράφει το transaction που στέλνει να χρησιμοποιεί το private key που αντιστοιχεί στο public key στο οποίο είχαν σταλεί τα χρήματα μέσω του transaction που υπάρχει στο input**.[5]**

2)Σε περίπτωση που ένας χρήστης χάσει το private key του, λόγω της ανωνυμίας και της αποκέντρωσης ,δε μπορεί να εξαργυρώσει τα χρήματα που έχουν σταλεί στο αντίστοιχο public key και έτσι τα χρήματα παραμένουν εγκλωβισμένα **.[5]**

3)Εάν τα χρήματα που υπάρχουν στο output των transactions που έχουμε βάλει για input είναι περισσότερα από τα χρήματα που στέλνουμε τότε η διαφορά (fees) θα σταλεί στο miner που θα συμπεριλάβει το συγκεκριμένο transaction στο block που θα βγάλει ,όπως θα δούμε στη συνέχεια .Αυτό είναι ένα επιπλέον κίνητρο για τους miners να συμπεριλάβουν το transaction ,αν και συνήθως τα fees είναι πολύ μικρό ποσό**.[5]**

4)Σε περίπτωση που τα χρήματα που θέλουμε να στείλουμε είναι λιγότερα από τα χρήματα που παίρνουμε από το transaction του input και δε θέλουμε να δώσουμε fees στο miner τότε βάζουμε ένα ακόμα output που περιέχει το hash του δικού μας public key και τα χρήματα που θέλουμε να μας επιστραφούν**.[5] ,[10]**

5)Σε περίπτωση που τα χρήματα που έχει ο αποστολέας από το transaction του input είναι λιγότερα από αυτά που έχει δηλώσει ότι θα στείλει το transaction είναι άκυρο **.[5] ,[10]**

1.1.2 ΕΞΑΡΓΥΡΩΣΗ ΕΝΟΣ TRANSACTION

Στη δημιουργία των transactions χρησιμοποιείται μία γλώσσα που χρησιμοποιεί script και ουρά LIFO .Είναι πολύ εύχρηστη ,αλλά δεν είναι Turing complete. Έτσι δεν μπορεί να χρησιμοποιηθεί ‘’ FOR’’ ,το οποίο είναι θετικό επειδή δε χρειάζεται να εξετάζουμε αν υπάρχει ατέρμων βρόγχος ,αλλά έχει το μειονέκτημα ότι μας περιορίζει ως προς τα contracts που θα μπορούσαμε να δημιουργήσουμε όπως θα δούμε αργότερα και μας αναγκάζει να γράψουμε μεγαλύτερο κώδικα με πολλά “if” .Παρόλα αυτά μας προσφέρει τη δυνατότητα στο scriptPubkey να έχουμε πολλές παραλλαγές που μας δίνουν διαφορετικούς τρόπους εξαργύρωσης**.[5],[20]**

Ουσιαστικά η απόδειξη ότι έχουμε κάποια χρήματα είναι ότι υπάρχει ένα transaction μέσα σε ένα block της κύριας αλυσίδας για το οποίο τηρούμε την εντολή εξαργύρωσης .Έτσι εξαργυρώνουμε ουσιαστικά τα χρήματα από ένα transaction Τ όταν το χρησιμοποιούμε σαν input σε ένα άλλο transaction Τ’για να στείλουμε χρήματα σε ένα παραλήπτη που θέλουμε και στο scriptsig βάζουμε την κατάλληλη εντολή .Στη συνέχεια για να θεωρηθεί το transaction έγκυρο από το miner τοποθετείται στο script το “ <scriptsig> <scriptPubkey>”και μία- μία εντολή του script από αριστερά στα δεξιά εκτελείται στο κενό stack ,που ουσιαστικά είναι μία LIFO ουρά μέχρι να βγει η εντολή true.(Το scriptisig βρίσκεται στο Τ’ και το scriptPubkey στο Τ.)Αν βγει η εντολή true τότε το transaction είναι έγκυρο και ο

miner αν επιθυμεί μπορεί να το συμπεριλάβει στο block,αλλιώς θεωρείται άκυρο αφού ο αποστολέας δε πληροί την προϋπόθεση εξαργύρωσης του transaction του input.**[10]**

1.1.3 ΕΙΔΗ ΕΞΑΡΓΥΡΩΣΗΣ

1) **Pay-to-PubkeyHash [10]**

scriptPubKey: OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG

scriptSig: <sig> <pubKey>

Stack	Script
Empty.	<sig> <pubKey> OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
<sig> <pubKey>	OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
<sig> <pubKey> <pubKey>	OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
<sig> <pubKey> <pubHashA>	<pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
<sig> <pubKey> <pubHashA> <pubKeyHash>	OP_EQUALVERIFY OP_CHECKSIG
<sig> <pubKey>	OP_CHECKSIG
true	Empty.

Εικόνα από [10]

OP_DUP :διπλασιασμός στο τελευταίο item που μπήκε στο stack

OP_EQUALVERIFY: εξετάζει αν είναι ίσα τα δύο τελευταία αντικείμενα που μπήκαν στο stack

OP_CHEKSIG:ελέγχει αν η υπογραφή <sig> επαληθεύεται με το <pubkey>,κάτι το οποίο μας εξασφαλίζει ότι ο αποστολέας διαθέτει το private key που αντιστοιχεί στο pubkey και είναι ο παραλήπτης του transaction του input.

Ο κυριότερος τρόπος εξαργύρωσης είναι να εξαργυρώνει τα χρήματα του transaction αυτός που διαθέτει το public key ,του οποίου το hash χρησιμοποιείται στο scriptPubkey στο output του transaction. Σε αυτό τον τρόπο εξαργύρωσης στο scriptsig ≡ <sig> <pubkeyA>,όπου sig είναι η υπογραφή του αποστολέα πάνω στη simpilified version του A και pubkeyA είναι το pubkey του αποστολέα ,το οποίο απαιτείται για να ανακτηθεί το <pubkeyHash>.

2)Pay-to-Script-Hash

A)

scriptPubKey: OP_HASH160 <scriptHash> OP_EQUAL

scriptSig: ..signatures... <serialized script>

Αυτό το είδος εξαργύρωσης **[10]** βοηθάει στο να πραγματοποιούνται διάφορα συμβόλαια. Συγκεκριμένα στο serialized script μπορεί να γραφτεί οτιδήποτε. Η συνθήκη εξαργύρωσης είναι να γνωρίζει κάποιος το script του οποίου το hash βρίσκεται στο scriptPubKey.
Βέβαια μπορεί να απαιτούνται και κάποιες υπογραφές σε περίπτωση που το serialized script περιέχει στην αρχή του κομμάτι σαν OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG.

Δεδομένου ότι το scriptsig και το scriptpubkey γράφονται συνεχόμενα στο script μπορεί να μπει αυτό το κομμάτι και στο scrptsig.Είμαστε σίγουροι ότι αυτός που πάει να το εξαργυρώσει δε θα το παραλείψει γιατί θα αλλάξει το hash του serialized script που απαιτείται κατά τη διάρκεια εξαργύρωσης .

B) m-of-n multi-signature transaction:

scriptSig: 0 <sig1> ...

scriptPub key: OP_m <pubKey1> ... OP_n OP_CHECKMULTISIG

Με αυτό το είδος **[10]** εξαργύρωσης ελέγχεται αν τουλάχιστον m από τις n υπογραφές γίνονται verify με τα αντίστοιχα n public keys .

Stack	Script
Empty.	0 <sig1> <sig2> OP_2 <pubKey1> <pubKey2> <pubKey3> OP_3 OP_CHECKMULTISIG
0 <sig1> <sig2> OP_2 <pubKey1> <pubKey2> <pubKey3> OP_3	OP_CHECKMULTISIG
true	Empty

Εικόνα από [10]

3) **Anyone-Can-Spend Outputs [7]**

scriptPubKey: (empty)

scriptSig: OP_TRUE

Σε περίπτωση που ένα transaction έχει για scriptPubkey κενό τότε οποιοσδήποτε μπορεί να πάρει τα χρήματα που στέλνονται με το transaction δημιουργώντας ένα νέο transaction με scriptSig: OP_TRUE και public key παραλήπτη το δικό του. Ουσιαστικά σε περίπτωση που ανακοινωθεί ένα τέτοιο transaction τότε ο miner μπορεί να φτιάξει ένα transaction που παίρνει αυτά τα χρήματα και να το προσθέσει στο block που θα ανακοινώσει(σε περίπτωση βέβαια που είναι αυτός που θα βγάλει το επόμενο block).Αυτό θα μπορούσε να λειτουργήσει σαν μία δωρεά για τους miners**.[7]**

4)Provably Unspendable/Prunable Outputs[7]

scriptPubKey: OP_RETURN {zero or more ops}

Σε περίπτωση που δημιουργηθεί ένα τέτοιο transaction τότε τα χρήματα του transaction δε μπορούν να ξοδευτούν με κανένα scriptsig . Επίσης μπορεί να

χρησιμοποιηθεί για να προσθέσουμε μερικά data που θέλουμε να εμφανίζονται στο transaction.

5)Ένας άλλος τρόπος εξαργύρωσης μπορεί να είναι κάποιος από τους παραπάνω με τη διαφορά στο pubkey να υπάρχει στην αρχή το hash οτιδήποτε θέλουμε και μετά η εντολή OP_DROP ,η οποία διώχνει από το stack το τελευταίο αντικείμενο .Με αυτόν τον τρόπο αν μπει το transaction στο block δεσμευόμαστε στο hash οποιουδήποτε πράγματος επιθυμούμε ή έχουμε μία απόδειξη ότι ξέραμε κάτι τη στιγμή που βγήκε το block, όπως ένα κείμενο του οποίου διεκδικούμε τα πνευματικά δικαιώματα.[5],[7].[10],[12]

1.1.4 TRANSACTION MALLEABILITY

Δεδομένου ότι ο αποστολέας υπογράφει μία simplified version του transaction και όχι το scriptsig μπορεί κάποιος κακόβουλος να προσθέσει κάτι το οποίο να τροποποιεί το hash του transaction ,το ID του δηλαδή χωρίς να το καθιστά άκυρο .Για παράδειγμα θα μπορούσε να προσθέσει οτιδήποτε στο scriptsig και στο τέλος με μία εντολή OP_DROP,η οποία κατά την εξαργύρωση θα το αγνοούσε. Αυτό μπορεί να προκαλέσει επιθέσεις που αφορούν transactions που ακόμη δεν έχουν μπει στο block**. [5]**

Ένα παράδειγμα είναι κάποιος κακόβουλος Α που θέλει να αγοράσει ένα προϊόν από τον Β και πρέπει να στείλει κάποια χρήματα στο public key του Β να δημιουργήσει τα παρακάτω δύο transaction : Το πρώτο να στέλνει χρήματα από ένα public key του Α σε ένα άλλο public key του που έχει και το δεύτερο να έχει για input το προηγούμενο transaction και να στέλνει τα χρήματα στο Β . Αυτά τα δύο transaction ο Α τα δημοσιεύει ως υποψήφια να μπουν στο επόμενο block .Αν ο Β στείλει το προϊόν πριν περιμένει να μπουν στο block τότε μπορεί ο Α να τροποποιήσει το ID του πρώτου transaction κι έτσι το δεύτερο transaction να είναι άκυρο.

Ένας επιπλέον λόγος που υπάρχει αυτό το πρόβλημα είναι ότι αναφερόμαστε στο transaction μέσω του ID.Τέτοια προβλήματα μπορούν να αντιμετωπιστούν αν περιμένουμε να μπουν τα transactions στο block ,καθώς και περιορίζοντας τις επιτρεπτές εντολές OP_στο scriptsig ,όπως ήδη γίνεται σε κάποιες εκδόσεις των transaction .

1.1.5 SIGNATURE HASH TYPES

Όπως είδαμε παραπάνω είναι πολύ σημαντικό πιο κομμάτι υπογράφεται ,επειδή δεν μπορεί να τροποποιηθεί. Έτσι υπάρχουν πολλά είδη ως προς το τι υπογράφει ο αποστολέας ανάλογα με το τι θέλει να αφήσει ελεύθερο να μπορεί να μεταβληθεί. Πιο συγκεκριμένα υπάρχουν τα παρακάτω είδη **[5]:**

SIGHASH_ALL :Είναι η κλασσική υπογραφή που υπογράφεται από τον ένα signer όλο το transaction εκτός από το scriptsig.

SIGHASH_NONE: Υπογράφονται από τον ένα signer όλα τα input και κανένα output ,αφήνοντας ευμετάβλητα τα outputs,που πάνε δηλαδή τα Satoshi,εκτός και αν τα υπογράψει κάποιος άλλος signer.

SIGHASH_SINGLE: Υπογράφει ο signer μόνο το δικό του input και το αντίστοιχο output, επιτρέποντας στους άλλους signer να αλλάξουν το δικό τους κομμάτι του transaction.

SIGHASH_ALL|SIGHASH_ANYONECANPAY: Υπογράφει ο signer μόνο το δικό του input και όλα τα output επιτρέποντας επιπλέον σε οποιοδήποτε να προσθέσει ή να αφαιρέσει inputs.

SIGHASH_NONE|SIGHASH_ANYONECANPAY:Υπογράφει ο signer μόνο το δικό του input και κανένα output ,ενώ ταυτόχρονα επιτρέπει σε οποιοδήποτε να προσθέσει inputs και outputs .

SIGHASH_SINGLE|SIGHASH_ANYONECANPAY: Υπογράφει ο signer μόνο το δικό του input και το αντίστοιχο output και επιτρέπει σε οποιονδήποτε να προσθέσει ή να αφαιρέσει άλλα inputs.

Αν υπάρχουν πολλά είδη inputs σε ένα transaction μπορεί να εφαρμοστεί διαφορετικός τύπος υπογραφής για κάθε input.

1.2 BLOCK

1.2.1 ΔΟΜΗ BLOCK

Το Block ουσιαστικά είναι ένα κομμάτι της αλυσίδας (blockchain) ,δηλαδή είναι ένα κομμάτι της ιστορίας του νομίσματος. Αυτό που το καθιστά πολύ σημαντικό είναι ότι περιέχει τις καινούριες συναλλαγές και για να είναι έγκυρο πρέπει να είναι έγκυρες

και οι συναλλαγές ,δηλαδή να μην ξοδεύονται χρήματα από κάποιον που δεν τα έχει. Επίσης το block είναι ο τρόπος κοπής νομισμάτων αφού κάθε φορά που βγαίνει ένα block δημιουργούνται κάποια νέα νομίσματα και δίνονται σαν ανταμοιβή σε αυτόν που το έβγαλε ,το miner,όπως θα δούμε στη συνέχεια. Η κύρια δομή του είναι η παρακάτω: **[4],[5]**

Magic number: μία τιμή 4 bytes η οποία είναι σταθερά η τιμή 0xD9B4BEF9

Block size : μία τιμή 4 bytes που δείχνει το μέγεθος του block

Block header: είναι το κύριο κομμάτι του block (80 bytes)

Transaction counter: θετικός ακέραιος που δείχνει τον αριθμό των transactions (1-9 bytes)

Transactions : τα transaction που διαλέγει ο miner να συμπεριλάβει στο block που θα βγάλει , είναι από αυτά που έχουν γίνει broadcast και είναι έγκυρα ,δηλαδή έχουν την κατάλληλη μορφή και δε γίνεται μέσω αυτών double spending .Το πρώτο από τα transactions του block είναι το κίνητρο του miner να βγάλει το block και είναι ένα transaction (coinbase transaction ή generation transaction) το οποίο στέλνει κάποιο συγκεκριμένο αριθμό νομισμάτων που δημιουργούνται εκείνη τη στιγμή στο public key του miner .Αυτό το transaction έχει ένα επιπλέον χαρακτηριστικό ότι δε μπορεί να ξοδευτεί από το miner για τα επόμενα 100 blocks.Αυτό αποτρέπει το miner να ξοδέψει τα χρήματα αυτά σε περίπτωση που το block που έβγαλε τελικά ανήκει σε μία αλυσίδα μικρότερη από την κύρια με αποτέλεσμα το συγκεκριμένο coinbase transaction να είναι άκυρο.(όπως θα δούμε στη συνέχεια έγκυρη θεωρείται η μεγαλύτερη αλυσίδα από blocks)**[5],[10]**

1.2.2 BLOCK HEADER

Το block header περιλαμβάνει τα παρακάτω **[11] ,[20]:**

Version: αποτελεί τη version του block

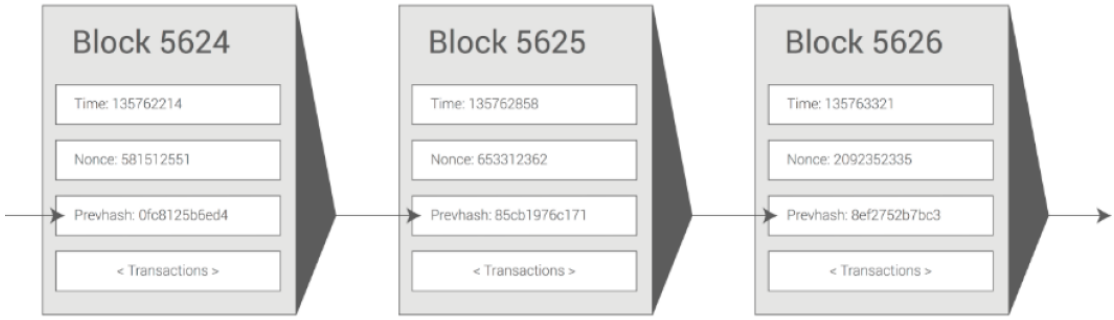
Hashprevblock: περιλαμβάνει το hash του προηγούμενου block header της αλυσίδας και είναι 256 bits

HashMerkleRoot: είναι ένα hash 256 bits που σχετίζεται με τα transactions. Είναι πολύ δύσκολο δύο miner να έχουν το ίδιο ,επειδή διαφέρει το αριστερό φύλλο που είναι το coinbase transaction

Time :είναι η ώρα τη στιγμή που βγαίνει το block .Μπορεί να διαφέρει ανά node του δικτύου

Bits:Σχετίζεται με τη δυσκολία του mining,όπως θα περιγράψουμε παρακάτω

Nonce: Είναι το μεταβλητό κομμάτι του block (32 bit) ,μεταβάλλεται γραμμικά (όχι αναγκαστικά με τον ίδιο τρόπο σε όλους τους miners) και συμμετέχει στο proof of work,έτσι όπως θα το ορίσουμε αργότερα.

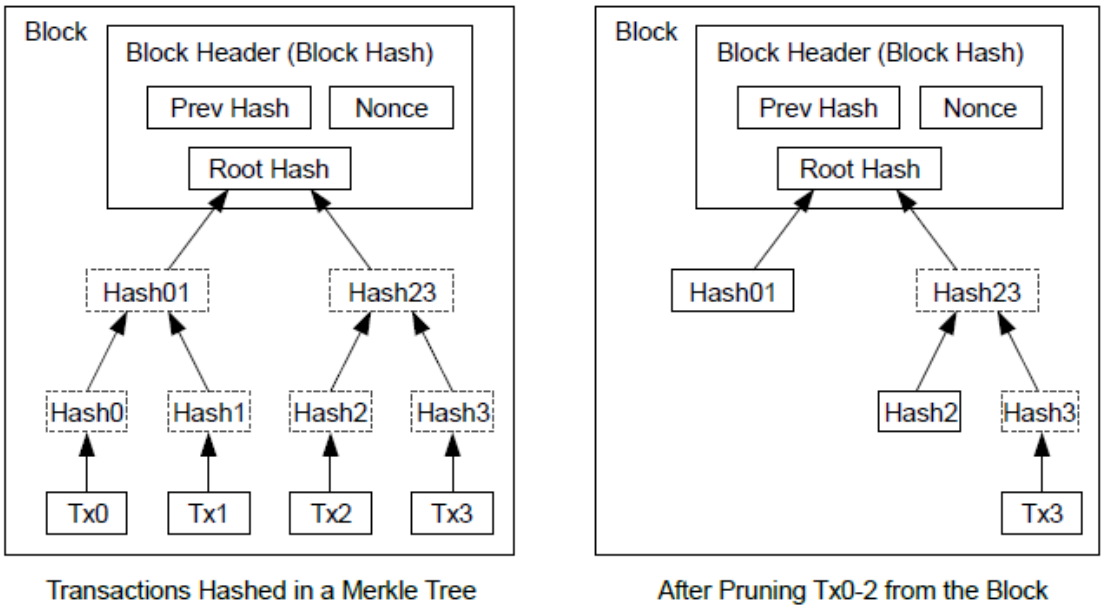


Εικόνα από [20]

1.3 MERKLE TREE

Τα transactions είναι αποθηκευμένα σε μία μορφή δέντρου που είναι το Merkle tree .Αυτό το είδος δέντρου έχει σαν φύλλα του τα(hash των) transactions .Κάθε γονιός προκύπτει αν κάνουμε padding τα δύο παιδιά του, και στην συνέχεια τα κάνουμε hash.Σε περίπτωση που ένας γονιός έχει ένα παιδί κάνουμε padding δύο φορές το ίδιο παιδί. Η ρίζα του Merkle tree αποθηκεύεται στο block header και είναι η πιστοποίηση ότι κανένα transaction δε μπορεί να τροποποιηθεί, γιατί αυτό θα μεταβάλει και τη ρίζα.

Η δομή του Merkle tree **[13],[5]** επιτρέπει σε κάποιους nodes (light nodes) να μην έχουν αποθηκευμένα όλα τα transactions ,να έχουν αποθηκευμένη μόνο τη ρίζα . Αν χρειαστεί να επαληθεύσουν ότι ένα transaction είναι αποθηκευμένο σε ένα block μπορούν να ζητήσουν από ένα full node (που έχει αποθηκευμένο όλα τα transactions)τις κορυφές που χρειάζεται (branch) για να ελέγξει ότι βρίσκει την ίδια ρίζα ,με αποτέλεσμα να κάνει verify ότι το transaction συμπεριλαμβάνεται στο δέντρο του block.Αυτή η διαδικασία ονομάζεται ***SIMPLIFIED PAYMENT VERIFICATION(SPV) [3].***



Εικόνα από [3]

1.4 DOUBLE SPENDING

Η δομή του block ,έτσι όπως περιγράφηκε παραπάνω συμβάλλει στο να μην υπάρχει double spending.Με τον όρο double spending**[3]** εννοούμε να είναι έγκυρες δύο συναλλαγές που στέλνουν τα ίδια νομίσματα σε δύο διαφορετικούς παραλήπτες ,χωρίς να μπορούμε να συμφωνήσουμε ποιος είναι ο πραγματικός κάτοχος των νομισμάτων.

Έτσι όπως λειτουργεί το bitcoin δεν έχουμε double spending ,επειδή έτσι κι έχει συμπεριληφθεί σε ένα block ένα transaction μετά οποιοδήποτε transaction στέλνει τα ίδια νομίσματα (ίδιο input) σε άλλον παραλήπτη δε μπορεί να μπει σε block ,επειδή δε θα θεωρηθεί έγκυρο από το miner.Αν ο miner το συμπεριλάβει παρόλο που είναι άκυρο οι υπόλοιποι nodes δε θα θεωρήσουν το δικό του block έγκυρο και δε θα το συνεχίσουν ,με αποτέλεσμα να μην ανήκει στην κύρια αλυσίδα.

1.5 MINING

Mining **[3],** ονομάζεται η διαδικασία κατά την οποία όσοι nodes(miners) του peer to peer δικτύου επιθυμούν συμμετέχουν στο proof of work με σκοπό να εκπληρώσουν την προϋπόθεση που απαιτείται για να βγάλουν το επόμενο έγκυρο block της αλυσίδας(blockchain) και να κερδίσουν τα αντίστοιχα bitcoin μέσω του coinbase transaction που αναφέραμε παραπάνω.

Η αμοιβή ξεκίνησε στα 50 bitcoin ,και υποδιπλασιάζεται κάθε 210.000 blocks,δηλαδή περίπου κάθε 4 χρόνια **.[5]**

1.6 PROOF OF WORK

Το proof of work **[3]** είναι ουσιαστικά η δουλεία που απαιτείται από τους miners μέχρι να βγει το επόμενο block .Συγκεκριμένα οι miner κάθε φορά δημιουργούν το block header,αφού έχουν ελέγξει ότι τα transactions που θα συμπεριλάβουν σε αυτό είναι έγκυρα, το περνούν από hash και ελέγχουν αν το αποτέλεσμα είναι μικρότερο από κάποιο συγκεκριμένο αριθμό που προσδιορίζεται από το difficulty ,έτσι όπως είναι διαμορφωμένο εκείνη τη στιγμή. Εάν είναι, τότε ο miner που το πέτυχε αυτό ανακοινώνει το block και αν οι υπόλοιποι nodes ,ελέγξουν ότι είναι έγκυρο συνεχίζουν την αλυσίδα που ανήκει αυτό το block.Εάν δεν είναι μικρότερο ο miner αλλάζει το timestamp και το nonce και ξαναδοκιμάζει μέχρι να το πετύχει.

Επειδή αυτό ουσιαστικά που απαιτεί το proof of work **[5],[6]** είναι υπολογισμός hash πολλές φορές έχει αναπτυχθεί ειδικό hardware που το πραγματοποιεί αυτό(ASICS) .Αυτό έχει σαν αποτέλεσμα το γεγονός ότι για να είναι ένας miner ανταγωνιστικός να πρέπει να διαθέσει αρκετά χρήματα.

Το difficulty**[5]** μεταβάλλεται ανά 2.016 και μπορεί να υπολογιστεί από τον κάθε node.Σε περίπτωση που ένας miner χρησιμοποιήσει άλλο difficulty το block του θεωρείται άκυρο. Ουσιαστικά για να υπολογιστεί το difficulty ελέγχεται μέσω timestamps αν ο χρόνος που απαιτήθηκε για να βγουν τα 2.016 blocks ήταν περίπου 1.209.600 sec(δύο εβδομάδες περίπου. Αν απαιτήθηκε λιγότερο χρονικό διάστημα το difficulty θα ανέβει αναλογικά ,ενώ αν απαιτήθηκε περισσότερο το difficulty θα μειωθεί έτσι ώστε ο αναμενόμενος χρόνος των επόμενων 2.016 blocks να είναι δύο εβδομάδες.

Η εγκυρότητα του block **[5],[20]** έγκειται στο ότι :1) έχει τη σωστή μορφή 2)στο γεγονός ότι το hashprevblock αναφέρεται σε έγκυρο block 3) στο ότι το timestamp του είναι μεγαλύτερο από το timestamp του προηγούμενου block και μικρότερο από το timestamp του τελευταίου block αυξανόμενο κατά δύο ώρες 4) στο ότι τα

transactions είναι έγκυρα και δεν έχουμε double spending,δηλαδή το output των referenced transactions που βρίσκονται στο input των καινούριων transactions δεν έχει ξοδευτεί unspent transaction output UTXO(παίζει ρόλο η σειρά των transactions στο block) 5) είναι σωστό το proof of work

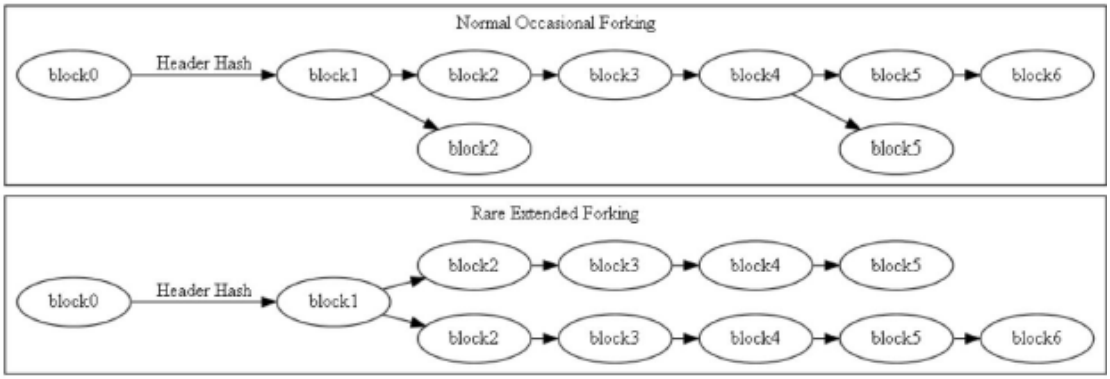
1.7 BLOCKCHAIN

Το blockchain είναι **[3],[5]** ουσιαστικά η κύρια αλυσίδα από blocks ,η οποία είναι αποδεκτή από τους nodes του peer to peer δικτύου και περιγράφει το πώς μεταφέρονται τα bitcoin από public key σε public key μέσω transactions. Η αλυσίδα ξεκινάει με το genesis block και επεκτείνεται συνεχώς περίπου κάθε 10 λεπτά. Σε περίπτωση που δημιουργηθούν παραπάνω από μία αλυσίδες με έγκυρα blocks αποδεκτή είναι η μεγαλύτερη δεδομένου ότι για να δημιουργηθεί έχει γίνει περισσότερο proof of work.

Ένας από τους τρόπους με τους οποίους μπορεί να δημιουργηθεί παραπάνω από μία αλυσίδες είναι σε παρόμοια χρονική στιγμή να βγουν δύο blocks τα οποία είναι έγκυρα, δηλαδή να δημιουργηθεί ένα fork και κάποιοι nodes να συνεχίσουν το ένα block και κάποιοι άλλοι nodes το άλλο. Κάποια στιγμή μία από τις δύο αλυσίδες θα γίνει μεγαλύτερη, θα θεωρηθεί κύρια και η άλλη δε θα συνεχιστεί. Τα τελευταία blocks της αλυσίδας (όπως φαίνεται παρακάτω **[5]**)η οποία δεν έγινε αποδεκτή λόγω μήκους λέγεται ***orphan.***Ο πατέρας ενός orphan block μπορεί να ανήκει ή όχι στην κύρια αλυσίδα.**[5]**

Η αρνητική πλευρά του να συμβαίνει το παραπάνω φαινόμενο πολύ συχνά είναι ότι το proof of work πολλών miner που έχουν βγάλει έγκυρο block δεν έχει ανταπόκριση, επειδή τελικά το block τους βρίσκεται στη συνέχεια εκτός κύριας αλυσίδας.

Ένας άλλος τρόπος να έχουμε δύο αλυσίδες είναι κάποιος που θέλει να αλλάξει την'' ιστορία'' και έχει την ανάλογη cpu power να εργάζεται μόνος του βγάζοντας blocks ,χωρίς να τα ανακοινώνει στο δίκτυο και κάποια στιγμή να τα ανακοινώσει όλη την αλυσίδα .Αν η αλυσίδα του αποτελείται από έγκυρα blocks και είναι μεγαλύτερη, αυτή είναι η κύρια ιστορία.



Εικόνα από [5]

1.8 ΑΣΦΑΛΕΙΑ ΑΝ Ο MALICIOUS ΕΧΕΙ ΛΙΓΟΤΕΡΟ ΑΠΟ 51% HASH POWER

Ένα πολύ σημαντικό χαρακτηριστικό της block chain είναι ότι κάθε block περιέχει το hash του προηγούμενου block με αποτέλεσμα να εξαρτάται από αυτό ,καθώς και από όλα τα προηγούμενα αναδρομικά. Αυτό υποχρεώνει έναν node που θέλει να αλλάξει ένα transaction που έχει υπογράψει ο ίδιος μέσα σε ένα block A της κύριας αλυσίδας να πρέπει όχι μόνο να δημιουργήσει ένα έγκυρο block B που να περιέχει το τροποποιημένο transaction , αλλά να δημιουργήσει περισσότερα σε πλήθος έγκυρα blocks από όσα υπάρχουν στην κύρια αλυσίδα μετά το A, έτσι ώστε η δική του αλυσίδα να γίνει κύρια. Αυτό λόγω του proof of work απαιτεί υπολογιστική ισχύ και θα δειχθεί στη συνέχεια ότι αν οι malicious nodes δεν ξεπερνούν το 50% της συνολικής hash power που διατίθεται για mining έχουν πολύ μικρή πιθανότητα να το πραγματοποιήσουν ,όπως περιγράφεται στο **[3]**

Για να δειχθεί αυτό θα χρειαστεί να περιγραφεί το **Gambler's Ruin Problem.[2],[1]**

1.8.1. GAMBLER'S RUIN PROBLEM

Στο παραπάνω πρόβλημα **[2] ,[1]** έχουμε έναν παίκτη, ο οποίος έχει αρχική περιουσία i $ και σε κάθε γύρο κερδίζει 1$ με πιθανότητα p ή χάνει 1$ με πιθανότητα 1-p=q

ανεξάρτητα από τους προηγούμενους γύρους .Το παιχνίδι τελειώνει μόλις ο παίκτης έχει συνολικά N $,όπου κερδίζει ή μόλις μείνει με 0 $,όπου χάνει.

Αυτό μοντελοποιείται ως εξής **[1] :**

Έστω R_n, $n\geq 0$ μία τυχαία ακολουθία αποτελεσμάτων γύρων ενός παίκτη συμπεριλαμβανομένου και του i. Έχουμε $R_n=K_0+K_1+K_2+\dots K_n$, όπου $K_i=\pm 1$ για $0<i\leq n$ τα κέρδη κάθε γύρου και $R_o=i=K_0$.

Ισχύει $Pr(K_i=+1)=p$, $Pr(K_i=-1)=1-p=q$ και το παιχνίδι τελειώνει όταν $R_i=0$ ή $R_i=N$,όπου ο παίκτης χάνει ή κερδίζει ανάλογα.

Ο γύρος που τελειώνει το παιχνίδι αν $R_0=i$ είναι $\tau_i=min\{n\geq o:R_n \in \{0,N\} /R_0=i\}$

$P_i=Pr(R_{\tau i}=N)$ είναι η πιθανότητα ο παίκτης να νικήσει και $1-P_i$ η πιθανότητα να χάσει.

Ισχύει ότι $P_0=0$,αφού από τον πρώτο γύρο ο παίκτης έχει *0$* με αποτέλεσμα να χάνει ανεξαρτήτως τι γίνεται στους επόμενους γύρους και $P_N=1$,αφου ξεκινάει ο παίκτης κατευθείαν με *N$,* οπότε νικάει.

Θα υπολογιστεί **[1]** το P_i χρησιμοποιώντας δέσμευση ως προς το K_1.

$P_i = Pr(R_{\tau i}=N/K_1=+1)\cdot Pr(K_1=+1)+ Pr(R_{\tau i}=N/ K_1=-1)\cdot Pr(K_1=-1)= P_{i+1}\cdot p+ P_{i-1}\cdot q$ *(1)*

$p+q=1\rightarrow P_i=p\cdot P_i+q P_i \rightarrow$ *(1)*

$P_{i+1}\cdot p+ P_{i-1}\cdot q = p\cdot P_i+q P_i \rightarrow P_{i+1}- P_i = {}^{q}\!/_{p}\cdot (P_i - P_{i-1}) \rightarrow$

$P_{i+1}- P_i = {}^{q}\!/_{p}{}^{i}\cdot P_1, \quad i>0 \quad και \quad i<N$

$P_{i+1}- P_1= P_{i+1}- P_i + P_i +\dots.-P_1 = \sum_{k=1}^{i} {}^{q}\!/_{p}{}^{k}\cdot P_1 \rightarrow$

$P_{i+1}= \sum_{k=0}^{i} {}^{q}\!/_{p}{}^{k} P_1 = \frac{1-{}^{q}\!/_{p}{}^{i+1}}{1-{}^{q}\!/_{p}}\cdot P_1$ *αν* $p\neq q$ *,αλλιώς* $P_{i+1}=(i+1)\cdot P_1$,*αν* $p=q=0.5$.,από εφαρμογή τύπου γεωμετρικής σειράς.

Ξέρουμε ότι $P_N=1$,έτσι αν εφαρμοστεί ο παραπάνω τύπος για $i=N-1$ έχουμε $\frac{1-{}^{q}\!/_{p}{}^{N}}{1-{}^{q}\!/_{p}}\cdot$

$P_1=1\rightarrow \quad P_1=\frac{1-{}^{q}\!/_{p}}{1-{}^{q}\!/_{p}{}^{N}}$ *αν* $p\neq q$ *και* $P_1=\frac{1}{N}$ *αν* $p=q=0.5$

Έτσι προκύπτει ο τύπος $\boldsymbol{P_i=\frac{1-{}^{q}\!/_{p}{}^{i}}{1-{}^{q}\!/_{p}}\cdot \frac{1-{}^{q}\!/_{p}}{1-{}^{q}\!/_{p}{}^{N}}=\frac{1-{}^{q}\!/_{p}{}^{i}}{1-{}^{q}\!/_{p}{}^{N}}}$ ***αν*** $\boldsymbol{p\neq q}$ ***και*** $\boldsymbol{P_i=i\cdot\frac{1}{N}}$ ***αν*** $\boldsymbol{p=q=0.5.}$

Εάν *p>0.5*, δηλαδή p>q → $^{q}/_{p}$<1 ,ισχύει $\lim_{N\to\infty} Pi = \lim_{N\to\infty} \frac{1-{}^{q}/_{p}{}^{i}}{1-{}^{q}/_{p}{}^{N}} = 1 - {}^{q}/_{p}{}^{i}$ >0 ενώ αν p≤q , δηλαδή $^{q}/_{p}$≥1 , τότε ισχύει, *αν* $p \neq q$, $\lim_{N\to\infty} Pi = \lim_{N\to\infty} \frac{1-{}^{q}/_{p}{}^{i}}{1-{}^{q}/_{p}{}^{N}}$ =0 και *αν* $p=q$ $\lim_{N\to\infty} Pi = \lim_{N\to\infty} \cdot \frac{1}{N}\ i$=0. **[1]**

Αυτό πρακτικά συμβαίνει ότι αν ο παίκτης παίζει το παιχνίδι μέχρι να γίνει οσοδήποτε πλούσιος ή μέχρι να χάσει δηλαδή για N =∞, αν p>q τότε η πιθανότητα να νικήσει και να γίνει απεριόριστα πλούσιος είναι θετική, ενώ αν p≤q η πιθανότητα να νικήσει είναι 0. Αυτή η ιδιότητα θα φανεί πολύ χρήσιμη για την απόδειξη **[1],[3]** που θα περιγραφεί στη συνέχεια.

1.8.2 ΠΙΘΑΝΟΤΗΤΑ ΕΠΙΤΥΧΙΑΣ MALICIOUS NODE ΜΕ CPU POWER <51%

Έστω κάποιος malicious ,ο οποίος υπέγραψε ένα transaction με το οποίο μεταβιβάζει έναν αριθμό bitcoin σε ένα παραλήπτη Β για κάποιο αγαθό ή υπηρεσία που του προσέφερε και το transaction συμπεριλαμβάνεται σε ένα έγκυρο block της κύριας blockchain ,ενώ ταυτόχρονα έχουν βγει άλλα z blocks μετά από αυτό .Θα εξετάσουμε την περίπτωση που ο malicious αφού παρέλαβε αυτό που επιθυμούσε από τον Β δημιούργησε ένα νέο transaction με το οποίο τα ίδια χρήματα που έστελνε στον Β ,τα στέλνει στο δικό του public key και προσπαθεί αφενός να βγάλει ένα έγκυρο block που να το περιέχει ,αφετέρου να βγάλει και τον απαιτούμενο αριθμό επόμενων blocks,έτσι ώστε μόλις ανακοινώσει τη δικιά του αλυσίδα να γίνει εκείνη αποδεκτή. Με υπόθεση ότι ο malicious έχει λιγότερη hash power από 51% θα περιγραφεί η πιθανότητα να το καταφέρει **.[3]**

Έστω ότι η αλυσίδα που έχει ο malicious είναι κατά z blocks μικρότερη από την κύρια αλυσίδα. Τότε για να βρεθεί η πιθανότητα να τα καταφέρει [3] το πρόβλημά ανάγεται στο **Gambler's Ruin Problem [1],[2].** Συγκεκριμένα θέτουμε **[3]** :

1) ότι το R_0=z

2)ότι κάθε γύρος αντιπροσωπεύει την ανακοίνωση του επόμενου block

3) ότι ο παίκτης αντιπροσωπεύει το υπόλοιπο δίκτυο που δεν είναι malicious

4)σε κάθε γύρο αν βγάλει ο παίκτης το επόμενο block αυξάνει το R_i ,την απόστασή του δηλαδή από τον malicious κατά ένα *($R_{i+1}=R_i+1$)*. Αυτό το πετυχαίνει με πιθανότητα p

5)Σε κάθε γύρο αν δεν βγάλει το block ο παίκτης ,αλλά ο malicious μειώνεται η απόστασή τους κατά ένα $(R_{i+1}=R_i-1)$.Αυτό συμβαίνει με πιθανότητα $q=1-p$.

6)Θεωρούμε N=∞ ,δηλαδή συνεχίζουν επ άπειρον τίμιοι miners ,δηλαδή ο παίκτης του παιχνιδιού μας ,μέχρι να τους φτάσει ο malicious ή μέχρι να απέχουν άπειρη απόσταση.

7)Το $P_z=1-{}^{q}/_{p}{}^{z}$ για p>0.5 και P_z =0 για p≤q. Το P_z είναι η πιθανότητα τελικά οι τίμιοι miners να απομακρυνθούν από τον malicious άπειρη απόσταση .

Άρα η πιθανότητα ο malicious να φτάσει τους τίμιους είναι 1- $P_z={}^{q}/_{p}{}^{z}$ για p>0.5 και 1- P_z = 1 για p≤q. Παρατηρούμε ότι αν q<p η πιθανότητα του malicious να πετύχει τον στόχο του μειώνεται εκθετικά ως προς τα block που απέχουν**.[3],[1]**

1.8.3 ΠΙΟ ΠΡΟΣΕΚΤΙΚΟΣ ΥΠΟΛΟΓΙΣΜΟΣ

Στο παραπάνω παράδειγμα στην πραγματικότητα παρόλο που ξέρουμε ότι οι τίμιοι miners έχουν βγάλει z blocks δεν γνωρίζουμε την πρόοδο του malicious.Γι αυτό το λόγο για να υπολογιστεί **[3]** μία πιο ακριβή πιθανότητα επιτυχίας του malicious θα υπολογιστεί υπό τη δέσμευση της αρχικής προόδου του malicious,δηλαδή πόσα blocks είχε βγάλει κατά τη διάρκεια που οι τίμιοι έβγαλαν z blocks. Θεωρούμε ότι p>q ,αφού ο malicious έχει hash power<51%

Έστω A η τυχαία μεταβλητή που δείχνει την η αρχική πρόοδο του malicious.Για παράδειγμα αν είναι A= i απέχει ο malicious από τους τίμιους z-i.

Τότε Pr(επιτυχία malicious)= $\sum_{k=0}^{\infty} Pr(επιτυχία\ malicious/A=k)\cdot Pr(A=k)$

$Pr(επιτυχία\ malicious/A=k)=\ {}^{q}/_{p}{}^{z-k}$,όπως δείξαμε παραπάνω για $k\leq z$ και $Pr(επιτυχία\ malicious/A=k)=1$ *για* $k>z$

Η $Pr(A=k)$ εκφράζει την πιθανότητα ο malicious να βγάλει k blocks κατά τη διάρκεια που οι τίμιοι έχουν βγάλει z. Εάν υποθέσουμε ότι έχουν γίνει συνολικά n ανακοινώσεις blocks κατά τη διάρκεια που έχουν βγάλει οι τίμιοι τα z blocks ο malicious, αν έχει πιθανότητα q να βγάλει το επόμενο block, το πόσα blocks θα βγάλει είναι μία διωνυμική κατανομή με παραμέτρους k, n,q. Αν υποθέσουμε ότι $z=p\cdot n$ και προσεγγίσουμε τη διωνυμική με κατανομή Poisson τότε έχουμε $\lambda=n\cdot q={}^{z}/_{p}\cdot q$ *και*

$$Pr(A = k) = \frac{\lambda^{\kappa}}{k!} \cdot e^{-\lambda}$$

Έτσι έχουμε ότι *Pr(επιτυχία malicious)*= $\sum_{k=0}^{\infty} \frac{\lambda^{\kappa}}{k!} \cdot e^{-\lambda} \cdot \{ {}^{q}/_{p}{}^{z-k}$ *για k≤z*

1 για k>z }

ή αλλιώς ***Pr(επιτυχία malicious)= 1-*** $\sum_{k=0}^{z} \frac{\lambda^{\kappa}}{k!} \cdot e^{-\lambda} \cdot (1 - {}^{q}/_{p}{}^{z-k})$ **.[3]**

1.9 ΜΙΑ 25% ΕΠΙΘΕΣΗ ΣΤΟ BITCOIN NETWORK

Όπως περιγράφηκε παραπάνω σε περίπτωση που το hash power του malicious δεν ξεπερνά το 50% η πιθανότητα η δική του αλυσίδα να γίνει κύρια είναι πολύ μικρή και μειώνεται εκθετικά ως προς τη διαφορά μήκους της κύριας αλυσίδας και της αλυσίδας του.

Υπάρχει **[8]** όμως μία διαφορετικού είδους επίθεση που περιγράφηκε στο **[9]** η οποία απαιτεί 25% hash power. Σε αυτήν την επίθεση ο attacker εργάζεται ακολουθώντας έναν αλγόριθμο , σύμφωνα με τον οποίο δημιουργεί μία δική του αλυσίδα που στην αρχή την διατηρεί κρυφή έως ότου κάποια στιγμή την ανακοινώσει. Με αυτόν τον τρόπο αν τελικά γίνει αποδεκτή η δική του αλυσίδα ,λόγω μήκους ,οι υπόλοιποι miners χάνουν τα χρήματα τους .

Ο αλγόριθμος **[8],[9]** είναι ο παρακάτω:

Έστω X το ποσοστό του hash power του network που έχει ο attacker και Z το ποσοστό του υπόλοιπου network που προσπαθεί να βγάλει το επόμενο block στην αλυσίδα του attacker σε περίπτωση που είναι δημοσιευμένες δύο αλυσίδες.

ΒΗΜΑ Ο: Σε περίπτωση που η private αλυσίδα είναι ίσου μήκους με την public αλυσίδα προσπάθησε να βγάλεις το επόμενο block στην private αλυσίδα .

Αν το καταφέρει (με πιθανότητα X) → πήγαινε στο *ΒΗΜΑ1*

Αλλιώς αν το υπόλοιπο network καταφέρει να βγάλει το επόμενο block στη public αλυσίδα (με πιθανότητα 1-X) → τροποποίησε την private έτσι ώστε να είναι ίδια με την public αλυσίδα.

ΒΗΜΑ 1: Αν η public αλυσίδα είναι ένα block πίσω από την private συνέχισε το mining στην private.

Αν καταφέρεις να βγάλεις το επόμενο block (με πιθανότητα X) → πήγαινε στο ΒΗΜΑ 2(η private είναι δύο blocks μπροστά)

Αλλιώς αν βγάλει το υπόλοιπο network το επόμενο block στην public αλυσίδα (με πιθανότητα 1-X) → πήγαινε στο *ΒΗΜΑ 0΄ (*οι δύο αλυσίδες έχουν ίδιο μήκος)

ΒΗΜΑ 0΄:*Δημοσίευσε την private αλυσίδα.* (Υπάρχουν τώρα δύο δημοσιευμένες αλυσίδες) Κάνε mining στην πρώην private αλυσίδα.

Αν βγάλεις το επόμενο block (με πιθανότητα X) → κάνε reset στο ΒΗΜΑ 0 (ο attacker έχει βγάλει συνολικά δύο έγκυρα blocks και η αλυσίδα του γίνεται κύρια).

Αν το υπόλοιπο δίκτυο βγάλει το επόμενο block στην πρώην private αλυσίδα (με πιθανότητα (1-X)· Z) → κάνε reset στο ΒΗΜΑ 0 (ο attacker έχει βγάλει ένα έγκυρο block και οι τίμιοι miners ομοίως ,ενώ η αλυσίδα του attacker έχει γίνει κύρια)

Αν το υπόλοιπο δίκτυο βρει το επόμενο block στην εξ' αρχής public αλυσίδα → κάνε reset στο ΒΗΜΑ 0 (οι τίμιοι miners έχουν βγάλει δύο blocks ,ενώ ο attacker κανένα αφού η αλυσίδα του δεν έγινε κύρια).

ΒΗΜΑ 2: Συνέχισε να κάνεις mining στην αλυσίδα του attacker.

Αν καταφέρεις να βγάλεις το επόμενο block (με πιθανότητα X) → πήγαινε στο ΒΗΜΑ 3

Αλλιώς εάν το υπόλοιπο network βγάλει block στην public αλυσίδα(με πιθανότητα 1-X) → *δημοσίευσε την private αλυσίδα.*(Είναι ένα block μπροστά από την public και έτσι θα γίνει κύρια και ο miner θα έχει κερδίσει ένα block)

ΒΗΜΑ n (n>2): Συνέχισε να κάνεις mining στην private αλυσίδα.

Αν καταφέρεις να βγάλεις το επόμενο block (με πιθανότητα X) → πήγαινε στο ΒΗΜΑ n+1

Αλλιώς εάν το υπόλοιπο network βγάλει block στην public αλυσίδα(με πιθανότητα 1-X) → πήγαινε στο ΒΗΜΑ n-1.

ΠΑΡΑΤΗΡΗΣΕΙΣ

1)Στον παραπάνω αλγόριθμο παρατηρούμε ότι δημοσιεύεται από τον attacker η private αλυσίδα στο ΒΗΜΑ 0΄ ή στο ΒΗΜΑ 2 πάντα αφού το υπόλοιπο δίκτυο έχει βγάλει block στην public αλυσίδα**.[8],[9]**

2) Στο ΒΗΜΑ Ο΄ σε περίπτωση που ο attacker δεν έχει αρκετή hash power να βγάλει μόνος του το επόμενο block στη δικιά του αλυσίδα η πιθανότητα το υπόλοιπο δίκτυο να βγάλει block στη δικιά του αλυσίδα είναι πολύ μικρό ,καθώς το Z είναι πολύ μικρό. Αυτό ισχύει επειδή ο αλγόριθμος του mining υπαγορεύει να συνεχίζεται το block που έφτασε πρώτο που στην προκειμένη περίπτωση είναι το block του υπόλοιπου network στην public αλυσίδα. **.[8],[9]**

3)Σε αυτό το σημείο η επίθεση που μπορεί να πραγματοποιήσει ο attacker είναι να δημιουργήσει εκατομμύρια κόμβους" Sybil attack", και να τους ενσωματώσει στο network ,οι οποίοι να αναμεταδίδουν μόνο το δικό του block. Σε αυτήν την περίπτωση το Z αντί να πηγαίνει κοντά στο 0 , πλησιάζει το 1 ή καλύτερα το 0.8. (επειδή τα mining pools(συνεργαζόμενοι miners) θα αναμεταδίδουν μεταξύ τους το block που έβγαλαν). **.[8],[9]**

4)Εάν το Z είναι κοντά στο 1 ο attacker δεν βγάζει σχεδόν ποτέ blocks ,τα οποία τελικά δεν ανήκουν στην κύρια αλυσίδα ,αφού στο ΒΗΜΑ Ο΄ γίνεται η δική του αλυσίδα κύρια με πιθανότητα κοντά στο 1 και στο ΒΗΜΑ 2 πάντα η private γίνεται κύρια αλυσίδα ,αφού είναι μεγαλύτερη. **.[8],[9]**

5)Αντίθετα στην παραπάνω περίπτωση στο ΒΗΜΑ Ο΄ και στο ΒΗΜΑ2 το υπόλοιπο δίκτυο βγάζει blocks, τα οποία μετά δεν τους αποφέρουν κέρδη γιατί δεν ανήκουν στη κύρια αλυσίδα .Ο attacker ,δηλαδή είναι πιο αποδοτικός(ως προς το αναμενόμενο κέρδος) με αποτέλεσμα οι τίμιοι nodes ,που λειτουργούν έτσι ώστε να μεγιστοποιήσουν το κέρδος τους να συμφέρει να εργαστούν στην αλυσίδα του attacker . **.[8],[9]**

6) Ένας τρόπος για να αντιμετωπιστεί αυτό είναι ο αλγόριθμος του mining να διαλέγει τυχαία σε ποιο block από αυτά που του έρχονται θα συνεχίσει ,με αποτέλεσμα το Z σε αυτή την περίπτωση να είναι 0.5 και όπως αποδείχτηκε στο paper για να γίνει αυτή η επίθεση να απαιτείται $X \geq 25\%$. Σε περίπτωση όμως που ένας attacker κάνει Sybil attack και διαθέτει $X > 25\%$ δεν τον εμποδίζει αυτό. Επίσης υπάρχει περίπτωση ο attacker να προσελκύσει τους profit-maximizing nodes προκειμένου να αποκτήσει $X > 1/3$ και να πραγματοποιεί την παραπάνω επίθεση χωρίς Sybil attack ,αφού το καταφέρνει ανεξαρτήτως Z ,όπως περιγράφεται στο paper. **.[8],[9]**

7) Βέβαια για να συμβεί αυτό πρέπει να τους το ανακοινώσει ο attacker ότι πραγματοποιεί την επίθεση ,με αποτέλεσμα πολλοί nodes να μην συμμετέχουν για ιδεολογικούς σκοπούς ,δεδομένου ότι είναι ‘’κακό’’ για τη λειτουργία του bitcoin και αυτό δε τους συμφέρει ,αφού το είδος των χρημάτων που κερδίζουν από το mining είναι bitcoin . Εκτός από αυτό μπορεί οι τίμιοι nodes να δεχτούν προσφορές από το mining pool για να μην αποχωρήσουν ,κάτι το οποίο μπορεί να εμποδίσει ακόμη και τους profit- maximizing nodes. **.[8],[9]**

1.10 CONTRACTS

Στις παραπάνω ενότητες είδαμε την λειτουργία του BITCOIN . Η αλυσίδα όμως του BITCOIN και τα transactions πέρα από την προφανή χρησιμότητα που έχουν μπορούν να χρησιμοποιηθούν και για πιο πολύπλοκες συμφωνίες μεταξύ ανθρώπων (contracts).**[12]**

1.10.1 ΣΥΜΒΟΛΑΙΟ ΓΙΑ ΠΟΛΛΕΣ ΜΙΚΡΟΠΛΗΡΩΜΕΣ ΣΤΟΝ ΙΔΙΟ ΠΑΡΑΛΗΠΤΗ

Ένας πελάτης σε μία καφετέρια θέλει να πληρώσει για το WIFI και συγκεκριμένα να πληρώνει ένα πολύ μικρό ποσό BITCOIN π.χ 0.001 BITCOIN ανά κάποια kilobytes χρήσης. Μία λύση ήταν κάθε φορά να πραγματοποιεί ένα transaction που να στέλνει τα χρήματα στο public key . Όμως αυτό απαιτεί χρόνο ,ταυτόχρονα επιβαρύνει το δίκτυο με πολλά transactions που αφορούν πολύ μικρό ποσό χρημάτων και έχουν όλα τον ίδιο παραλήπτη και τέλος αν ο πελάτης βάζει και fees για τον miner του κοστίζουν.

Μια πιο αποδοτική λύση είναι το παρακάτω συμβόλαιο **[12]** το οποίο μπορεί να εφαρμοστεί και σε διάφορες άλλες περιπτώσεις μικροπληρωμών.

Έστω Α ο πελάτης που πληρώνει και Β ο παραλήπτης των πληρωμών.

Ο Α θα ακολουθήσει την παρακάτω διαδικασία **[12]:**

1)Δημιουργεί public key P1 και ζητάει από τον Β το public key του ,έστω P2

2)Δημιουργεί ο Α ένα transaction Τ1 με το οποίο στέλνει χρήματα ,τα οποία μπορούν να εξαργυρωθούν με υπογραφή δικιά του και του Β (output μπορεί να περιλαμβάνει την εντολή : OP_2 < P1>,<P2> ... OP_2 OP_CHECKMULTISIG)

. Το υπογράφει ο Α το transaction ,αλλά δεν το κάνει broadcast ακόμη. Ο λόγος είναι ότι παρόλο που δεν μπορεί ο Β να παραλάβει τα χρήματα ,χωρίς την υπογραφή του Α υπάρχει το ενδεχόμενο ο Β να μην δέχεται να υπογράψει με αποτέλεσμα τα χρήματα του Α να είναι δεσμευμένα στο δίκτυο, χωρίς να μπορεί κανείς να τα εξαργυρώσει.

3) Ο Α δημιουργεί ένα transaction Τ2 το οποίο έχει για input το Τ1 και στέλνει τα χρήματα στο δικό του public key.Το transaction αυτό έχει ένα lock time κάποιο χρονικό διάστημα ,που σημαίνει ό τι θα επιστρέφει όλα τα χρήματα που έστειλε ο Α με το Τ1 πίσω στον εαυτό του μετά από το πέρας του lock time. Αυτό το transaction για να είναι έγκυρο πρέπει να δημοσιοποιηθεί το Τ1 και στο scriptsig να υπάρχουν οι υπογραφές και του Α και του Β. Ο Α χωρίς να βάλει τη δικιά του υπογραφή το στέλνει στο Β να το υπογράψει.

4) Ο Β το υπογράφει με το private key που είναι ζευγάρι με το Ρ2 και το στέλνει στον Α. Ο Β δεν είναι σίγουρος για το τι περιλαμβάνει το Τ1,αφού ξέρει μόνο το hash του που υπάρχει στο input του Τ2, αλλά δεν τον επηρεάζει σε κάτι ,δεδομένου ότι δεν έχει κάτι να χάσει. Με την υπογραφή που έβαλε ο Β ο Α κατοχυρώθηκε ότι δεν υπάρχει περίπτωση τα χρήματά του να μείνουν δεσμευμένα στο δίκτυο.

5)Στη συνέχεια ο Α κάνει verify την υπογραφή του Β με το Ρ2 και αν δεν είναι σωστή σταματάει.

6)Αν είναι σωστή η υπογραφή του Β , υπογράφει ο Α το Τ2 και κάνει broadcast τα transactions.

7)Στη συνέχεια δημιουργεί ένα transaction Τ3 το οποίο έχει input πάλι το Τ1, χωρίς όμως lock time ,το οποίο στέλνει τα χρήματα σε δύο public keys :στο δικό του και στου Β. Συγκεκριμένα στέλνει όλα τα χρήματα στο δικό του και τίποτα στο Ρ2.Ο Α βάζει την υπογραφή του στο scriptsig και το στέλνει στο Β.

8)Ο Β ελέγχει ότι το output έχει το ανάλογο μέγεθος και κάνει verify την υπογραφή του Α .Αν αρχικά ο Β το υπογράψει και το κάνει broadcast ,χωρίς να προσφέρει υπηρεσία δεν κερδίζει τίποτα ,αφού όλα τα χρήματα επιστρέφουν στον Α άμεσα.

9)Στη συνέχεια κάθε φορά που ο Α θέλει να πληρώσει για μία μικρή υπηρεσία στέλνει στον Β το νέο μοίρασμα στο Ρ1 και στο Ρ2, όπου τα χρήματα που πάνε στο Ρ2 αυξάνονται ,ενώ τα χρήματα που πάνε στο Ρ1 μειώνονται, καθώς και την υπογραφή του στο νέο Τ3.

10)Με αυτά τα δεδομένα ο Β μπορεί να ελέγξει το output και την υπογραφή του Α στο Τ3 ,επειδή έχει την αρχική μορφή του Τ3 και τα κομμάτια που τροποποιήθηκαν .Έτσι όσο ο Α πληρώνει ο Β συνεχίζει να του παρέχει υπηρεσίες και το ανάποδο .Μόλις ο Α δεν επιθυμεί άλλες υπηρεσίες και σταματάει να στέλνει ο Β υπογράφει το Τ3 και λαμβάνει τα χρήματά που του αναλογούν ,ενώ τα υπόλοιπα πάνε στον Α . Μόλις ο Β το κάνει αυτό ,το Τ2 επειδή δε θα έχει λήξει το lock time του για να μπορεί να μπει σε block θα προλάβει να μπει πρώτα το Τ3 σε block και μετά το Τ2

που επιστρέφει τα χρήματα στον Α θα είναι άκυρο, έτσι δεν κινδυνεύει ο Β να χάσει τα χρήματά του.

11)Σε περίπτωση που ο Β εξαφανίζεται και δεν παρέχει υπηρεσίες ,ούτε υπογράφει το Τ3 ο Α μετά τη λήξη του lock time θα πάρει τα χρήματά του.

Με αυτό το συμβόλαιο ενώ δημιουργούνται πολλά transaction γίνονται broadcast πολύ λίγα, χωρίς αυτό να επηρεάζει την εμπιστοσύνη μεταξύ του Α και του Β.

1.10.2 ΣΥΜΒΟΛΑΙΟ ΓΙΑ ΤΗΝ ΠΡΑΓΜΑΤΟΠΟΙΗΣΗ ΕΡΓΟΥ ΚΟΙΝΗΣ ΩΦΕΛΕΙΑΣ

Ένα τέτοιο συμβόλαιο **[12]** χρησιμεύει σε περίπτωση που κάποιος είναι διατεθειμένος να πληρώσει ένα μικρό ποσό σε περίπτωση που συνεισφέρουν και υπόλοιποι πολίτες προκειμένου να πραγματοποιηθεί ένα έργο. Για παράδειγμα μπορεί κάποιος να επιθυμεί να πληρώσει ένα μικρό ποσό για να μεταφραστεί μία σελίδα που επιθυμεί και σε περίπτωση που κάνει το ίδιο ένας απαιτούμενος αριθμός ανθρώπων προκειμένου να συγκεντρωθεί το κατάλληλο χρηματικό ποσό για να πληρωθεί η εταιρεία να γίνεται η μετάφραση. Ένα άλλο παράδειγμα είναι να επιθυμεί να γίνει κάποιος ένα δημόσιο έργο που αφορά μία γειτονιά ή να μπει ένα WIFI spot σε ένα κοινόχρηστο χώρο.

Ο τρόπος που πραγματοποιείται αυτό είναι ο παρακάτω **[12]:**

1) Ένας επιχειρηματίας που είναι διατεθειμένος να πραγματοποιήσει το έργο ανακοινώνει ότι θα το κάνει σε περίπτωση που μαζευτούν συνολικά από πολίτες π.χ 500 BITCOIN και ότι μπορεί να συνεισφέρει ο καθένας. Αρχικά δημιουργεί και ανακοινώνει το public key του το K1.
2) Αν κάποιος πολίτης A_i θέλει να πληρώσει δημιουργεί ένα transaction T_i που έχει σαν input ένα transaction το οποίο μπορεί να εξαργυρώσει και στέλνει 500 BITCOIN στο K1.
3) Ο A_i υπογράφει το transaction T_i με τον τύπο υπογραφής SIGHAH_ALL/SIGHASH_ANYONECANPAY.

4) Σε περίπτωση που ο Α δεν έχει ακριβώς το ποσό που επιθυμεί να διαθέσει ,επειδή θέλουμε τα χρήματα να πηγαίνουν αποκλειστικά στο K1 και όχι σε δικό του public key για ρέστα , για λόγους που θα δούμε στη συνέχεια , κάνει το εξής: δημιουργεί πρώτα ένα transaction , το οποίο θα χρησιμοποιήσει στο input του T_i, που παίρνει για είσοδο ένα transaction για το οποίο τηρεί την εντολή εξαργύρωσης και στέλνει τα χρήματα σε δικά του public keys ,με

αποτέλεσμα να στέλνει στο public key που είναι ζευγάρι με το private key που υπογράφει το T_i ακριβώς το ποσό που επιθυμεί.

5) Στη συνέχεια ο A_i ανεβάζει το transaction T_i στο server του επιχειρηματία ,ο οποίος το αποθηκεύει και αυξάνει την τιμή της μεταβλητής που προσμετρά το συνολικό ποσό που έχει συγκεντρωθεί από τα inputs των transactions που έχουν ανέβει.

6) Μόλις ο μετρητής φτάσει τα 500 bitcoin ο επιχειρηματίας ενώνει τα transactions σε ένα κοινό transaction το οποίο έχει για input όλα τα inputs των transactions ,για output το κοινό output όλων των transactions και στο scriptsig τις υπογραφές όλων των transactions . Αυτό το transaction είναι έγκυρο επειδή το output το οποίο έχουν υπογράψει οι συμμετέχοντες δεν έχει αλλάξει ,ούτε το input του καθενός .Το μόνο που έχει αλλάξει από το transaction του κάθε συμμετέχοντα είναι να προστεθούν και άλλα inputs ,κάτι το οποίο το επιτρέπει η συνθήκη ANYONECANPAY. Στο τέλος ο επιχειρηματίας κάνει broadcast αυτό το transaction και πληρώνεται.

7) Ο κάθε συμμετέχοντας είναι σίγουρος ότι ο επιχειρηματίας δεν μπορεί να χρησιμοποιήσει τα χρήματα του εάν δεν μαζευτεί το απαιτούμενο ποσό για να γίνει το έργο ,επειδή το transaction του έχει μεγαλύτερο output value από input value και είναι άκυρο, Επίσης ακόμα και να ενωθεί με άλλα δεν γίνεται έγκυρο εάν τα χρήματα που μαζεύονται είναι λιγότερα από 500 bitcoin.

   Ένας τρόπος να αποφευχθεί η υπογραφή SIGHASH_ANYONECANPAY είναι να στείλουν μήνυμα με το ποσό που θέλουν να διαθέσουν όσοι θέλουν να συμμετέχουν ,χωρίς να στείλουν transaction και μόλις συγκεντρωθούν 500 BTC να δημιουργείται ένα transaction με αντίστοιχο αριθμό inputs με τον αριθμό των συμμετεχόντων το οποίο θα το υπογράψουν όλοι οι συμμετέχοντες.

1.10.3 ΣΥΜΒΟΛΑΙΟ ΓΙΑ ΔΙΑΧΕΙΡΙΣΗ ΠΡΟΒΛΗΜΑΤΩΝ ΣΕ ΜΗ ΕΠΙΤΥΧΗΜΕΝΗ ΑΓΟΡΟΠΩΛΗΣΙΑ

Σε αυτό το συμβόλαιο **[12]** έχουμε έναν έμπορο Α που στέλνει το προϊόν σε ένα αγοραστή Β και ο Β τον πληρώνει τον Α με BITCOIN . Σε περίπτωση που διεξαχθεί κανονικά η διαδικασία ,χωρίς προβλήματα το συμβόλαιο δεν κάνει τίποτα. Σε περίπτωση όμως που είτε ο έμπορος είτε ο αγοραστής δεν έδρασαν τίμια, θα έχουμε ένα διαμεσολαβητή Γ που έχουν ορίσει ο Α και ο Β ,επειδή τον εμπιστεύονται και οι δύο ,ο οποίος θα μπορεί να αποζημιώσει τον αδικημένο**.**

Συγκεκριμένα ο αγοραστής Α εκτελεί την παρακάτω διαδικασία **[12]**:

1)Συμφωνεί με τον έμπορο Β στο ποιος θα είναι ο Γ. Ο Γ μπορεί να είναι και ένας επαγγελματίας που κάνει αυτή την δουλειά ,εξετάζει δηλαδή ποιος έχει δίκιο και ενεργεί αναλόγως.

2) Ζητάει από τον Β το public key του Π1, ζητάει από το Γ το public key του Π2 και δημιουργεί για τον εαυτό του ένα νέο public key Π3.

3) Στέλνει στο Β το Π2. Ο Β ,για να ελέγξει ότι το Π2 είναι το public key του Γ, στέλνει στον Γ ένα τυχαίο nonce και αφού ο Γ το υπογράψει ο Β κάνει verify.

4)Δημιουργεί και κάνει broadcast ένα transaction με το οποίο θα πληρώσει τον έμπορο Β σε περίπτωση που ο Β είναι τίμιος. Συγκεκριμένα το transaction θα έχει για pubkeyscript : ‘’ *2 <Π1> <Π2> <Π3> 3 CHECKMULTISIGVERIFY* ‘’.

- ✓ Με το παραπάνω transaction σε περίπτωση που ο Α και ο Β είναι τίμιοι ,δηλαδή ο Β στείλει το προϊόν και ο Α θέλει να πληρώσει υπογράφουν και οι δύο σε ένα transaction που στέλνει τα χρήματα στο Β και παίρνει ο Β τα χρήματά που του αναλογούν.

- ✓ Σε περίπτωση που ο Β είναι τίμιος ,αλλά ο Α δεν θέλει να τον πληρώσει, υπογράφει ο Γ και ο Β και παίρνει ο Β τα χρήματά του.

- ✓ Σε περίπτωση που ο Β δεν είναι τίμιος και δεν έστειλε το προϊόν ή το προϊόν ήταν ελαττωματικό τότε ο Α δε δέχεται να υπογράψει ούτε ο Γ ,και ο Β δε πληρώνεται.

2 ZEROCOIN : ANONYMOUS DISTRIBUTED E-CASH FROM BITCOIN

Όπως είχε αναφερθεί παραπάνω αν επιθυμούμε να είναι δύσκολο κάποιος να μάθει το συνολικό αριθμό bitcoin που διαθέτουμε πρέπει να μην συνδέουμε το public key μας με την ταυτότητα μας και επίσης με κάθε συναλλαγή να χρησιμοποιούμε διαφορετικό ζευγάρι (public key, secret key) έτσι ώστε να μην υπάρχει σύνδεση μεταξύ των bitcoin που ανήκουν στον ίδιο κάτοχο. **[5]**

Οι Ian Miers, Christina Garman, Matthew Green, Aviel D. Rubin στο **[22]** προτείνουν μία κατασκευή ,η οποία αν εφαρμοστεί στο bitcoin επιτρέπει ανώνυμα transactions έτσι ώστε να μην υπάρχει σύνδεση μεταξύ διαφορετικών transactions και επομένως σύνδεση μεταξύ bitcoins.

Η ιδέα πίσω από το zerocoin , φαίνεται με το παρακάτω παράδειγμα-πρωτόκολλο **[22]** :

Έστω ότι ο Bob επιθυμεί να λάβει ένα zerocoin αξίας 1$. Τότε θα διαλέξει ένα σειριακό αριθμό S, θα υπολογίσει ένα commitment αυτού του S (χρησιμοποιώντας ένα τυχαίο αριθμό r) που θα αποτελέσει το coin C που γίνεται mint. Ο Bob τοποθετεί το C σε ένα δημόσιο bulletin board ,αφού παραχωρήσει και 1$.

Για να εξαργυρώσει ο Bob το coin του(να το κάνει spend) θα δημιουργήσει ένα σύνολο **C** με όλα τα έγκυρα νομίσματα που υπάρχουν στο bulletin board και θα αποδείξει ότι ξέρει πρώτον ένα νόμισμα C του συνόλου αυτού που αντιστοιχεί σε ένα σειριακό αριθμό S ,και δέυτερον το r με το οποίο μπορεί να ανοίξει τη δέσμευση C του S. Αυτό θα το πραγματοποιήσει δημιουργώντας ένα coin spend transaction που περιλαμβάνει μία non interactive zero knowledge proof π και το S. Η απόδειξη δε θα πρέπει να δίνει σχεδόν καμία πληροφορία για το ποιο είναι το C. Η ύπαρξη του S είναι απαραίτητη για να μη εξαργυρώσει ο Bob το zerocoin 2 φορές , για αυτό θα πρέπει να ελέγχεται αν έχει ξαναεμφανιστεί κι άλλη φορά η ίδια τιμή του S σε άλλο spend transaction. Αν το (π,S) είναι έγκυρο τότε ο Bob μπορεί να πάρει οποιοδήποτε $ επιθυμεί από bulletin board.

Μία ουσιαστική διαφορά της τελικής μορφής του zerocoin είναι **[22]** ότι αντί για bulletin board χρησιμοποιείται η blockchain, όπου μπορούν να αποθηκεύονται τα transactions (mint και spend).

Επίσης σημαντική είναι και η ένταξη accumulator **[23],[22]** στο πρωτόκολλο του zerocoin ,έτσι ώστε ο έλεγχος ότι C∈**C** να γίνεται πιο αποδοτικά από γραμμικά στο πλήθος του **C .** Βέβαια επειδή δεν επιθυμούμε να υπάρχει έμπιστη αρχή πρέπει ο

accumulator να είναι δημόσια υπολογίσιμος και επαληθεύσιμος ,εκτός από το set up ,όπου θα χρειαστεί μία έμπιστη αρχή για τις παραμέτρους. Επίσης θα πρέπει ο accumulator να έχει zero knowledge απόδειξη συμμετοχής ,καθώς και να δεσμεύει το computation party στις τιμές των coin στο σύνολο. **C .**

Πριν περιγραφεί φορμαλιστικά πως λειτουργεί το zerocoin θα περιγραφούν κάποιοι ορισμοί που χρησιμοποιούνται σε αυτό.

2.1 DECENTRALIZED E CASH SCHEME

2.1.1 ΟΡΙΣΜΟΣ

Ένα decentralized e-cash scheme αποτελείται από τους αλγορίθμους :***Setup, Mint, Spend,Verify*.[22]**

- Setup(1^{λ}) →param. Με είσοδο το λ που είναι παράμετρος ασφάλειας έχει σαν έξοδο τις public παραμέτρους param και την περιγραφή ενός συνόλου *C* που απεικονίζει τις επιτρεπτές τιμές νομισμάτων.

- Mint(param) → (c,skc). Με input τις παραμέτρους param έχει σαν έξοδο ένα νόμισμα c ∈ *C* και ένα trapdoor skc.

- Spend(param,c ,skc,R,**C**) →(π,S). Με είσοδο τις param ,ένα νόμισμα c, το trapdoor skc, ένα transaction string R∈{O,1}* και ένα σύνολο από νομίσματα **C,** σε περίπτωση που c ∈ **C ,**όπου **C** είναι υποσύνολο του *C* έχει σαν έξοδο ένα coin spend transaction που περιλαμβάνει μία απόδειξη π και έναν σειριακό αριθμό S. Σε περίπτωση που το c δεν ανήκει στο **C** τότε έχει σαν output το κενό . (Το R αναπαριστά συγκεκριμένη πληροφορία ενός transaction ,όπως για παράδειγμα τον παραλήπτη του νομίσματος που γίνεται spent.

- Verify(param,π,S,R,**C**)→{0,1}. Με είσοδο τις παραμέτρους param ,την απόδειξη π, το σειριακό αριθμό S, το R και το **C** εκτυπώνει 1 αν **C** είναι υποσύνολο του *C* και η τριπλέτα (π,S,R) είναι έγκυρη. Διαφορετικά εκτυπώνει 0.

2.1.2 CORRECTNESS

CORRECTNESS:[22] Ένα decentralized e-cash scheme θεωρείται correct αν ένας honest που θέλει να κάνει mint ένα νόμισμα και στη συνέχεια να το ξοδέψει πραγματοποιεί το πρωτόκολλο ,τότε οι υπόλοιποι χρήστες κάνουν verify με πιθανότητα 1-κάτι αμελητέο.

Φορμαλιστικά:

Έστω Setup(1^{λ}) →param και Mint(param) → (c,skc).

Έστω **C,** υποσύνολο του *C ,* είναι οποιοδήποτε έγκυρο σύνολο από νομίσματα που ικανοποιεί $|\mathbf{C}| \leq poly(\lambda)$.

Έστω Spend(param,c ,skc,R,**C**) →(π,S).

Τότε το σχήμα είναι correct αν για όλα τα **C,**R και τα random coins που χρησιμοποιήθηκαν στους παρακάτω αλγορίθμους ισχύει η παρακάτω ισότητα με πιθανότητα 1-ν(λ) ,όπου ν(λ) είναι αμελητέα συνάρτηση:

Verify(param,π,S,R,**C**∪ {c})=1

2.1.3 SECURITY :ANOMYMITY KAI BALANCE

SECURITY : Το security αποτελείται από δύο ιδιότητες *Anonymity και Balance*.**[22]**

ANONYMITY

Με αυτή την ιδιότητα θέλουμε να διασφαλίσουμε ότι ο αντίπαλος δε μπορεί μέσω του (π,S) να αποκτήσει σχεδόν καμία πληροφορία για το c που έχει γίνει spend.

Συγκεκριμένα ένα decentralized e-cash scheme Π =(Setup,Mint,Spend,Verify) ικανοποιεί την Anonymity αν κάθε πιθανοτικός πολυωνυμικός(p.p.t) $A=(A_1,A_2)$ έχει αμελητέο πλεονέκτημα στο παρακάτω πείραμα **[22]** :

$\textbf{Anonymity}(\Pi, \mathcal{A}, \lambda)$
$params \leftarrow \mathsf{Setup}(1^\lambda)$
For $i \in \{0,1\}$: $(c_i, skc_i) \leftarrow \mathsf{Mint}(params)$
$(\mathbf{C}, R, z) \leftarrow \mathcal{A}_1(params, c_0, c_1)$; $b \leftarrow \{0,1\}$
$(\pi, S) \leftarrow \mathsf{Spend}(params, c_b, skc_b, R, \mathbf{C} \cup \{c_0, c_1\})$
Output: $b' \leftarrow \mathcal{A}_2(z, \pi, S)$

Εικόνα από **[22]**

Το πλεονέκτημα του *A* στο παραπάνω πείραμα είναι : $\left|\Pr[b = b'] - 1/2\right|$

Διαισθητικά στο παραπάνω πείραμα γίνονται mint δύο coins τα οποία δίνονται στον αντίπαλο, αφού έχουν καθοριστεί οι αρχικοί παράμετροι param.

Στη συνέχεια ο αντίπαλος εξάγει ένα σύνολο **C** ,ένα transaction string R και το z το οποίο ίσως τον διευκολύνει στο να επιτύχει.

Έπειτα γίνεται spend ένα από τα δύο νομίσματα με R αυτό που διάλεξε ο αντίπαλος και **C**'=**C**∪{c_1,c_2} ,ώστε η spend να μην μας βγάλει το κενό.

Τέλος ο αντίπαλος από το π,S και το z προσπαθεί να μαντέψει ποιο νόμισμα έγινε spend. Για να έχει το πρωτόκολλο anonymity πρέπει το π,S να μην του προσφέρει πιθανότητα να το βρει μεγαλύτερη από το να το διάλεγε τυχαία συν κάτι αμελητέο.

BALANCE

Με την ιδιότητα του balance επιδιώκεται ένας αντίπαλος να μην μπορεί να κάνει spend περισσότερα νομίσματα από ότι κάνει mint ,ακόμη και αν του δίνεται πρόσβαση σε περισσότερα coins και spend transaction των honest. Για να γίνει πιο ισχυρός ο ορισμός δίνεται στον αντίπαλο η δυνατότητα να τροποποιήσει valid coins τροποποιώντας για παράδειγμα το R**.[22]**

Ένα decentralized e- cash scheme έχει την ιδιότητα του balance αν ο αντίπαλος έχει μη αμελητέο πλεονέκτημα στο παρακάτω πείραμα **[22]** :

Αρχικά παράγονται οι παράμετροι με το Setup ,στη συνέχεια γίνονται mint N νομίσματα και ο αντίπαλος παίρνει σαν είσοδο τις παραμέτρους και τα νομίσματα.

Ο αντίπαλος έχει στη διάθεση του ένα oracle το οποίο σε κάθε ερώτηση j με είσοδο ένα c_j,ένα C_j και ένα R_j εκτυπώνει κενό αν το c_j δεν είναι ένα από τα νομίσματα που γίνονται mint αλλιώς εκτυπώνει Spend(param,c_j ,skc_j,R_j,$\mathbf{C_j}$) →(π_j,S_j) και κρατάει σε record το ζευγάρι (S_j,R_j) που εντάσσεται στο σύνολο T.

Το πλεονέκτημα του αντιπάλου θεωρείται η πιθανότητα να παράγει m νομίσματα και m+1 τετράδες (π',S',R',C') ,όπου περνούν το verify, το C' είναι υποσύνολο των νομισμάτων που έχουν γίνει mint και των νομισμάτων που έχουν παραχθεί από τον αντίπαλο ,τα ζευγάρια (S',R') δεν έχει καταγραφεί στο record του oracle και κανένα S' δεν εμφανίζεται δύο φορές στις τετράδες του αντιπάλου.

Ουσιαστικά ο αντίπαλος για όσα νομίσματα ρωτάει το oracle μπορεί να κάνει spend ίσα σε πλήθος νομίσματα χρησιμοποιώντας διαφορετικό (S,R) από αυτό που υπάρχει στο T ,αν χρησιμοποιήσει το (π,S) αυτών των νομισμάτων για να κάνει mint νέα νομίσματα με άλλο S για τα οποία ξέρει το trapdoor skc. Όμως το πλεονέκτημα του αντιπάλου εμφανίζεται αν μπορεί να κάνει spend m+1 νομίσματα ενώ έχει παράγει m minted νομίσματα.

$\mathsf{Balance}(\Pi, \mathcal{A}, N, \lambda)$
$params \leftarrow \mathsf{Setup}(1^\lambda)$
For $i = 1$ to N: $(c_i, skc_i) \leftarrow \mathsf{Mint}(params)$
Output: $(c'_1, \ldots, c'_m, \mathcal{S}_1, \ldots, \mathcal{S}_m, \mathcal{S}_{m+1})$
$\leftarrow \mathcal{A}^{\mathcal{O}_{spend}(\cdot,\cdot,\cdot)}(params, c_1, \ldots, c_N)$

The oracle $\mathcal{O}_{spend}$ operates as follows: on the j_{th} query $\mathcal{O}_{spend}(c_j, \mathbf{C}_j, R_j)$, the oracle outputs $\perp$ if $c_j \notin \{c_1, \ldots, c_N\}$. Otherwise it returns $(\pi_j, S_j) \leftarrow \mathsf{Spend}(params, c_j, skc_j, R_j, \mathbf{C}_j)$ to $\mathcal{A}$ and records (S_j, R_j) in the set $\mathcal{T}$.

We say that $\mathcal{A}$ wins (i.e., she produces more spends than minted coins) if $\forall \mathfrak{s} \in \{\mathcal{S}_1, \ldots, \mathcal{S}_m, \mathcal{S}_{m+1}\}$ where $\mathfrak{s} = (\pi', S', R', \mathbf{C}')$:

- $\mathsf{Verify}(params, \pi', S', R', \mathbf{C}') = 1$.
- $\mathbf{C}' \subseteq \{c_1, \ldots, c_N, c'_1, \ldots, c'_m\}$.
- $(S', R') \notin \mathcal{T}$.
- S' appears in only one tuple from $\{\mathcal{S}_1, \ldots, \mathcal{S}_m, \mathcal{S}_{m+1}\}$.

Εικόνα από **[22]**

2.2 DYNAMIC ACCUMULATOR

Ένας accumulator (συσσωρευτής) όπως προτάθηκε στο **[24]** και αργότερα βελτιώθηκε στο **[25]** και στο **[23]** είναι ουσιαστικά ένας αλγόριθμος που απεικονίζει (συσσωρεύει) ένα μεγάλο σύνολο από τιμές με μία τιμή μέσω μίας συνάρτησης hash και παρέχει ένα μάρτυρα για κάθε τιμή με τον οποίο αποδεικνύεται ότι η τιμή ανήκει στο σύνολο, ενώ ταυτόχρονα είναι πολύ δύσκολο για μία τιμή που δεν ανήκει στο παραπάνω σύνολο να βρεθεί ανάλογος μάρτυρας.
Στο **[23]** περιγράφεται ένας accumulator ο οποίος ικανοποιεί τη strong collision resistance property κάτω από την strong RSA assumption. Επίσης παρέχει και μία efficient zero knowledge proof ότι μία δεσμευμένη τιμή ανήκει στο σύνολο που συσσωρεύεται. Αυτές είναι κάποιες από τις ιδιότητες που θα μας χρειαστούν στο zerocoin.

STRONG RSA ASSUMPTION : Το flexible RSA problem είναι δύσκολο να λυθεί.

FLEXIBLE RSA PROBLEM : Έστω ένα RSA modulo n και ένα $y \in Z_n^*$,πρέπει να βρεθεί $x \in Z_n^*$ και $e>1$ τω $x^e=y$**.[23]**

2.2.1. ΟΡΙΣΜΟΣ ΚΑΙ ΑΣΦΑΛΕΙΑ

Ορισμός: [23] Ένας ασφαλής accumulator για μία οικογένεια inputs $\{X_k\}$ είναι μία οικογένεια από οικογένειες συναρτήσεων $G=\{F_k\}$ με τις παρακάτω ιδιότητες:

Efficient generation : Υπάρχει ένας αποδοτικός πιθανοτικός αλγόριθμος G που με input 1^k παράγει μία τυχαία συνάρτηση f από το F_k μαζί με μία auxiliary πληροφορία aux_f που συνδέεται με την f.

Efficient evaluation : $f \in F_k$ είναι ένα πολυωνυμικού μήκους circuit τέτοιο ώστε με είσοδο $(u,x) \in U_f \times X_k$ εκτυπώνει ένα $v \in U_f$, όπου U_f είναι ένα 'efficiently samplable' πεδίο ορισμού για την f και X_k είναι το πεδίο ορισμού για την f οι τιμές του οποίου πρόκειται να συσσωρευτούν.

Quasi-commutative : Για όλα τα $k \in F_k$,για όλα τα $u \in U_f$ και για όλα τα $x_1,x_2 \in X_k$ ισχύει $f(f(u,x_1),x_2)=f(f(u,x_2),x_1)$. Αν $X=\{x_1,x_2,\ldots,x_m\}$ υποσύνολο του X_k τότε με $f(u,X)$ ορίζουμε $f(f(\ldots..(u,x_1),\ldots),x_m)$.

(Αυτή η ιδιότητα μας χρειάζεται ώστε να μην παίζει ρόλο η σειρά των τιμών που συσσωρεύονται.)

Witnesses : Για $v \in U_f$ και $x \in X_k$,μία τιμή $w \in U_f$ λέγεται μάρτυρας για το x στο v μέσω της f αν ισχύει v=f(w,x).

Ουσιαστικά το v είναι ίσο με f(u,X) ,όπου X το σύνολο των τιμών που συσσωρεύονται, και w είναι ο μάρτυρας ότι το $x \in X$.

SECURITY

Έστω $U'_f \times X'_k$ είναι το πεδίο που ορίζεται η $f \in F_k$,οπότε ισχύει U_f υποσύνολο του U'_f και X_k υποσύνολο του X'_k. Για όλους τους p.p.t αντιπάλους A_k ισχύει

$$\Pr[f \leftarrow G(1^k); u \leftarrow \mathcal{U}_f; (x, w, X) \leftarrow \mathcal{A}_k(f, U_f, u) : X \subset \mathcal{X}_k; w \in \mathcal{U}'_f; x \in \mathcal{X}'_k; x \notin X; f(w, x) = f(u, X)] = \mathrm{neg}(k)$$

(Εικόνα από **[23])**

Ισχύει ότι μόνο οι τιμές $\{x_1,x_2,\ldots,x_m\}$ που συσσωρεύονται μέσω του accumulator σε μία τιμή πρέπει να ανήκουν στο X_k. Το x μπορεί να ανήκει σε ένα μεγαλύτερο σύνολο X'_k**.[23]**

Στη συνέχεια θα οριστεί πότε ένας ασφαλής accumulator είναι dynamic **[23].**

Ορισμός: Ένας ασφαλής accumulator είναι ***dynamic*** αν έχει την παρακάτω ιδιότητα**:[23]**

Efficient deletion : Υπάρχουν αποδοτικοί αλγόριθμοι D,W τω αν v=f(u,X) , x και x'ανήκουν στο X και f(w,x)=v τότε

- $D(aux_f,v,x')=v'$ τω $v'=f(u,X \setminus \{x'\})$
- $W(f,v,v',x,x')=w'$ τω $f(w',x)=v'$

Αυτή η ιδιότητα ουσιαστικά μας δείχνει ότι υπάρχουν αποδοτικοί αλγόριθμοι που μας δίνουν τη νέα τιμή που συσσωρεύει το σύνολο X χωρίς την τιμή που αφαιρέσαμε από αυτό ,καθώς και το νέο μάρτυρα για κάθε στοιχείο που έμεινε στο X.

Σχετικά με την πρόσθεση τιμής στο σύνολο που συσσωρεύεται αποδεικνύεται ότι μπορεί να υπολογιστεί η νέα τιμή που συσσωρεύει το σύνολο και οι νέοι μάρτυρες για τα στοιχεία αποδοτικά . Συγκεκριμένα ισχύει το παρακάτω λήμμα**:[23]**

ΛΗΜΜΑ :**[23]** Έστω $f \in F_k$ και v=f(u,X) είναι ο συσσωρευτής μέχρι τώρα. Θεωρούμε ότι $v'=f(v,x')=f(u,X \cup \{x'\})$ είναι η τιμή του συσσωρευτή μετά την πρόσθεση στο X του x'. Έστω w είναι ο μάρτυρας για το v ενός στοιχείου που ανήκει στο X . Τότε ο μάρτυρας w' του x για το v' υπολογίζεται σε χρόνο ανεξάρτητο από το μέγεθος του X.

ΑΠΟΔΕΙΞΗ[23] : Θα δειχθεί ότι το w' =f(w,x') είναι ο νέος μάρτυρας μέσω της Quasi-commutative ιδιότητας.

f(w',x)=f(f(w,x'),x)=f(f(w,x),x')=f(v,x')=v'.

Επίσης αποδεικνύεται **[23]** ότι ένας dynamic accumulator είναι ασφαλής απέναντι σε έναν αντίπαλο του παρακάτω σεναρίου : Έχουμε τον manager του accumulator που βγάζει μία συνάρτηση f ,ένα u και το aux_f που το κρατάει κρυφό από τον αντίπαλο. Ο αντίπαλος διαλέγει ένα σύνολο X το οποίο στη συνέχεια μπορεί να το μεταβάλει . Κάθε φορά που ο αντίπαλος προσθέτει ένα στοιχείο στο X, ο manager τροποποιεί κατάλληλα την τιμή του accumulator και κάθε φορά που αφαιρεί ο αντίπαλος ένα στοιχείο από το X ο manager εφαρμόζει την D για να βρει την τιμή του νέου accumulator και την ανακοινώνει. Στο τέλος ο αντίπαλος προσπαθεί να βρει έναν μάρτυρα ενός στοιχείου x 'που δεν ανήκει στο X για την τιμή v του accumulator έτσι όπως είναι διαμορφωμένη εκείνη τη στιγμή.

2.2.2 ΠΡΩΤΟΚΟΛΛΟ DYNAMIC ACCUMULATOR

Στη συνέχεια θα δοθεί μία κατασκευή ενός τέτοιου accumulator **[23].**

Η παρακάτω κατασκευή διαθέτει και μία αποδοτική zero knowledge proof ότι μία τιμή για την οποία δίνεται η δέσμευσή της ανήκει στο σύνολο που έχει συσσωρευτεί. Συγκεκριμένα σαν common inputs στο zero knowledge prorocol έχουμε ένα c=commit(x,r) ,όπου r τυχαίο string ,την accumulating συνάρτηση f και το $v \in U_f$. Το input του prover είναι (r, x $\in X_k$, u$\in U_f$) και θέλει να αποδείξει ότι γνωρίζει x,w, r τω c=commit(x,r) και f(w,x)=v.

Η κατασκευή είναι η ακόλουθη**:[23]**

- $\mathcal{F}_k$ is the family of functions that correspond to exponentiating modulo safe-prime products drawn from the integers of length k. Choosing $f \in \mathcal{F}_k$ amounts to choosing a random modulus $n = pq$ of length k, where $p = 2p'+1$, $q = 2q'+1$, and p,p',q,q' are all prime. We will denote f corresponding to modulus n and domain $\mathcal{X}_{A,B}$ by $f_{n,A,B}$. We denote $f_{n,A,B}$ by f_n and by f when it does not cause confusion.
- $\mathcal{X}_{A,B}$ is the $\{e \in \texttt{primes} : e \neq p', q' \wedge A \leq e \leq B\}$, where A and B can be chosen with arbitrary polynomial dependence on the security parameter k, as long as $2 < A$ and $B < A^2$. $\mathcal{X}'_{A,B}$ is (any subset of) of the set of integer from $[2, A^2 - 1]$ such that $\mathcal{X}_{A,B} \subseteq \mathcal{X}'_{A,B}$.
- For $f = f_n$, the auxiliary information aux_f is the factorization of n.
- For $f = f_n$, $\mathcal{U}_f = \{u \in QR_n : u \neq 1\}$ and $\mathcal{U}'_f = \mathbb{Z}^*_n$.
- For $f = f_n$, $f(u,x) = u^x \bmod n$. Note that $f(f(u,x_1),x_2) = f(u,\{x_1,x_2\}) = u^{x_1 x_2} \bmod n$

- Update of the accumulator value. As mentioned earlier, adding a value $\tilde{x}$ to the accumulator value v can be done as $v' = f(v, \tilde{x}) = v^{\tilde{x}} \bmod n$. Deleting a value $\tilde{x}$ from the accumulator is as follows. $D((p, q), v, \tilde{x}) = v^{\tilde{x}^{-1} \bmod (p-1)(q-1)} \bmod n$.
- Update of witness: As mentioned, updating the witness u after $\tilde{x}$ has been added can be done by $u' = f(u, \tilde{x}) = u^{\tilde{x}}$. In case, $\tilde{x} \neq x \in \mathcal{X}_k$ has be deleted from the accumulator, the witness u can be updated as follows. By the extended GCD algorithm, one can compute the integers a,b such that

$ax + b\tilde{x} = 1$ and then $u' = W(u, x, \tilde{x}, v, v') = u^b v'^a$. Let us verify that $f(u', x) = u'^x \bmod n = v'$:

$$(u^b v'^a)^x = \quad (1)$$

$$((u^b v'^a)^{x\tilde{x}})^{1/\tilde{x}} = \quad (2)$$

$$((u^x)^{b\tilde{x}} (v'^{\tilde{x}})^{ax})^{1/\tilde{x}} = \quad (3)$$

$$(v^{b\tilde{x}} v^{ax})^{1/\tilde{x}} = v^{1/\tilde{x}} = v' \quad (4)$$

(Εικόνα από **[23]**)

ΠΑΡΑΤΗΡΗΣΕΙΣ

1)Παρατηρούμε ότι ο παραπάνω accumulator ικανοποιεί την ιδιότητα **Quasi-commutative ,**αφού ισχύει $f((u,x_1),x_2) = f((u,x_2),x_1) = u^{x1x2} \bmod n$**.[23]**

2) Επίσης έχουμε efficient deletion αφού ορίζονται αποδοτικά η D και η W**[23] :**

$D(aux_f,v,x)=D((p,q),v,x)=v^{\tilde{x}^{-1} mod\ (p-1)q-1)} \bmod n$

$W(u,x,\tilde{x},v,v')=u^b v'^a$

3)Βλέπουμε ότι οι τιμές που ανήκουν στο σύνολο που συσσωρεύεται ($X_{A,B}$) είναι πρώτοι >2 και διάφοροι του p',q' ,κάτι το οποίο μας εξασφαλίζει ότι το $\tilde{x}$ αντιστρέφεται αφού ισχύει $(\tilde{x}, \varphi(n)) = (\tilde{x}, (p-1)(q-1)) = (\tilde{x}, 2p'2q') = 1$. Επίσης ότι είναι πρώτοι μας χρησιμέυει στο ότι $x \neq \tilde{x} \rightarrow (x, \tilde{x}) = 1 \rightarrow$ υπάρχουν a και b τω $ax + b\tilde{x} = 1$**.[23]**

2.2.3 ΑΠΟΔΕΙΞΗ ΑΣΦΑΛΕΙΑΣ ΠΡΩΤΟΚΟΛΛΟΥ

ΘΕΩΡΗΜΑ

Η παραπάνω κατασκευή είναι ένας secure dynamic accumulator κάτω από την strong RSA assumption**.[23]**

ΑΠΟΔΕΙΞΗ

Έστω ότι ο dynamic accumulator δεν ικανοποιεί το security έτσι όπως ορίστηκε παραπάνω ,οπότε ο αντίπαλος με input n και u $\in_R QR_n$ εκτυπώνει με μη αμελητέα πιθανότητα m πρώτους $x_1,x_2,\ldots,x_m \in X_{A,B}$,u'$\in Z_n$*, x'$\in X'_{A,B}$ έτσι ώστε $u'^{x'}= u^{\Pi x_i}$. Θα χρησιμοποιήσουμε τον αντίπαλο για να βρούμε έναν αλγόριθμο B που λύνει το flexible RSA πρόβλημα με μη αμελητέα πιθανότητα**.[23]**

Υποθέτουμε ότι ο B έχει για είσοδο n=pq με p=2p'+1 και q=2q'+1 και u $\in QR_n$ και θέλουμε να εκτυπώνει e>1 και y τω y^e=u. Θα δώσει στον αντίπαλο το (n,u) και θα πάρει με μία μη αμελητέα πιθανότητα (u',x', $x_1,x_2,\ldots,x_m$) ώστε $u'^{x'}= u^{\Pi x_i}$ ή

$u'^{x'}= u^x$,όπου $x=\prod_{i=1}^{m} x_i$. Αν gcd(x,x')=d τότε θα υποθέσουμε ότι d=1 ή $d=x_j$ για κάποιο j τω $1\leq j\leq m$ (λήμμα).

Υπάρχουν δύο περιπτώσεις**:**

1)gcd(d,φ(n))≠1 ,άρα και d≠1. Τότε $d=x_j \in X_{A,B}$ για κάποιο j και κατά συνέπεια d >2 και d πρώτος. Ακόμη φ(n)=4p'q', οπότε d=p' ή d=q' .Όμως ισχύει $\mathcal{X}_{A,B}$ is the $\{e \in \text{primes} : e \neq p', q' \wedge A \leq e \leq B\}$. (Εικόνα από **[23]**)

(Αν ίσχυε ότι d=p' ή d=q' τότε θα μπορούσαμε να παραγοντοποιήσουμε και το n)

2)gcd (d,φ(n))=1.Τότε το d αντιστρέφεται και έχουμε $u'^{x'}= u^x \rightarrow (u^{x'})^{1/d}=(u^x)^{1/d}$ (*)

Τότε θέτουμε $\tilde{x}'$=x'/d και $\tilde{x}$=x/d. Τότε gcd(x,x')=d → gcd($\tilde{x}'$, $\tilde{x}$)=1. Άρα μπορεί ο B να βρει a και b τω $a\tilde{x}+b\tilde{x}'=1$.(**)

Ο B εκτυπώνει $y=u'^a u^b$ και e=$\tilde{x}'$.

Αν ο αντίπαλος πετυχαίνει με μη αμελητέα πιθανότητα τότε ο Β με μη αμελητέα πιθανότητα λύνει το strong RSA problem.

Αυτό συμβαίνει γιατί $y^e=((u^{\tilde{x}'})^{\tilde{x}a} \cdot (u^{\tilde{x}})^{\tilde{x}'b})^{1/\tilde{x}} = ((u^{\tilde{x}})^{\tilde{x}a} \cdot (u^{\tilde{x}})^{\tilde{x}'b})^{1/\tilde{x}} = ((u^{\tilde{x}})^{a\tilde{x}+b\tilde{x}'})^{1/\tilde{x}}$=u ,από (*) και (**).

Επίσης έχουμε ότι το $\tilde{x}$ αντιστρέφεται αφού ισχύει:

1) Αν d=1 τότε ($\tilde{x}$, φ(n))= (x, φ(n))=1 ,αφού ισχύει (x_i,φ(n))=1 για 1≤i≤m($x_i \in X_{A,B}$) και x=$\prod_{i=1}^{m} x_i$,όπου x_i πρώτοι.

2) Αν d≠1 τότε d=x_j για κάποιο j τω 1≤j≤m ,οπότε το $\tilde{x}$ =x/d είναι ίσο με το γινόμενο m-1 πρώτων που είναι πρώτοι με το φ(n) ,άρα ($\tilde{x}$, φ(n))=1.

ΛΗΜΜΑ

Αν gcd(x,x')=d τότε d=1 ή d=x_j για κάποιο j τω 1≤j≤m (λήμμα).[**23]**

ΑΠΟΔΕΙΞΗ

Αν d≠1 τότε επειδή το d διαιρεί το x που είναι γινόμενο των πρώτων $x_1,x_2,\ldots,x_m \in X_{A,B}$ οι πρώτοι παράγοντες του είναι ένα υποσύνολο αυτών των πρώτων.

Έστω ότι ο d έχει παραπάνω από έναν πρώτους παράγοντες, όπως για παράδειγμα x_i,x_j , δηλαδή d= x_ix_j.

Τότε αφού d διαιρεί το x' ισχύει d ≤x' που είναι άτοπο ,γιατί d ≥A^2> x' από 1),2).**[23]**

1)$x_i,x_j \in X_{A,B}$ και άρα είναι ≥A ,οπότε το d είναι ≥A^2

2) A^2>x' ,δεδομένου το x'∈ $X'_{A,B}$

2.3 ΧΡΗΣΗ ACCUMULATOR ΣΤΟ ZEROCOIN

Στη συνέχεια θα παρουσιαστεί το πρωτόκολλο που εμφανίζεται στο **[22]** .Στο πρωτόκολλο ο accumulator περιγράφεται με τους αλγορίθμους AccumSetup,Accumulate,GenWitness,AccVerify**.[22]**

- AccumSetup$(\lambda) \rightarrow params$. On input a security parameter, sample primes p, q (with polynomial dependence on the security parameter), compute $N = pq$, and sample a seed value $u \in QR_N, u \neq 1$. Output (N, u) as $params$.
- Accumulate$(params, \mathbf{C}) \rightarrow A$. On input $params$ (N, u) and a set of prime numbers $\mathbf{C} = \{c_1, \ldots, c_i \mid c \in [\mathcal{A}, \mathcal{B}]\}$,[10] compute the accumulator A as $u^{c_1 c_2 \cdots c_n} \ mod \ N$.

- GenWitness$(params, v, \mathbf{C}) \rightarrow w$. On input $params$ (N, u), a set of prime numbers $\mathbf{C}$ as described above, and a value $v \in \mathbf{C}$, the witness w is the accumulation of all the values in $\mathbf{C}$ besides v, i.e., $w =$ Accumulate$(params, \mathbf{C} \setminus \{v\})$.
- AccVerify$(params, A, v, \omega) \rightarrow \{0, 1\}$. On input $params$ (N, u), an element v, and witness ω, compute $A' \equiv \omega^v \mod N$ and output 1 if and only if $A' = A$, v is prime, and $v \in [\mathcal{A}, \mathcal{B}]$ as defined previously.

(Εικόνα από **[22]**)

Ουσιαστικά η ***AccumSetup*** με είσοδο την παράμετρος ασφάλειας εκτυπώνει το ζεύγος (N,u) σαν παραμέτρους ,όπου το N =pq και u∈QR_N ,όπως είχε περιγραφεί και παραπάνω.

Η ***Accumulate*** παίρνει σαν είσοδο τις παραμέτρους και το σύνολο πρώτων **C** (αντιστοιχεί στο $X_{A,B}$ του accumulator που περιγράφηκε νωρίτερα) που πρόκειται να συσσωρευτούν και εκτυπώνει την τιμή του accumulator A = $u^{c_1 c_2 \cdots c_n} \mod N$.

Η ***GenWitness*** με είσοδο τις παραμέτρους ,ένα στοιχείο v ∈ **C** και το **C ,**εκτυπώνει το μάρτυρα w του v στο A ,ο οποίος είναι ίσος με την Accumulate (param,**C**\{v}).

Η ***AccVerify*** παίρνει σαν εισόδους τις παραμέτρους ,το A ,ένα v και το μάρτυρα του w και ελέγχει αν το w είναι όντως μάρτυρας του v στο A ($w^v \mod N=A$) και αν το v πληροί τις προϋποθέσεις να είναι πρώτος και να ανήκει στο [A,B] ,όπως έχει καθοριστεί.

Όσον αφορά τις **zero knowledge αποδείξεις** στο **[22]** γράφονται με τον εξής συμβολισμό **[29]:** NIZKPoK{(x,y): $h=g^x \cap c=g^y$} που συμβολίζει για παράδειγμα μία non interactive zero-knowledge proof of knowledge των στοιχείων x,y που ικανοποιούν τη σχέση $h=g^x \cap c=g^y$. Συγκεκριμένα με μία non interactive zero knowledge απόδειξη ένας Prover αποδεικνύει σε έναν verifier σε ένα μόνο γύρο ότι ξέρει το x και το y που ικανοποιούν τη σχέση $h=g^x \cap c=g^y$,χωρίς να αποκαλύψει τίποτα(εκτός αμελητέας πιθανότητας) από το x και το y. Επίσης σε περίπτωση που ο Verifier πειστεί από τον Prover με μη αμελητέα πιθανότητα τότε μπορεί ένας knowledge extractor που χρησιμοποιεί τον Prover σαν πρόγραμμα να εξάγει το x και το y με μη αμελητέα πιθανότητα.(soundness)**[26]**

Παρόμοια μία signature of knowledge πάνω στο μήνυμα m συμβολίζεται με ZKSoK[m] {(x,y): $h=g^x \cap c=g^y$} .

2.4 ΠΡΩΤΟΚΟΛΛΟ ZEROCOIN

Στη συνέχεια περιγράφεται το decentralized e-cash scheme του **[22]** ,το οποίο είναι ασφαλές κάτω από την strong RSA assumption, discrete logarithm assumptions και την υπόθεση ύπαρξης ενός zero-knowledge proof συστήματος.

Αρχικά ο **Setup** αλγόριθμος με είσοδο την παράμετρο ασφαλείας λ καλεί την AccumSetup με την ίδια παράμετρο και παίρνει το (N,u). Στη συνέχεια παράγει και τις υπόλοιπες παραμέτρους ,όπως φαίνεται στη συνέχεια και εκτυπώνει το διάνυσμα params=(N,u,p,q,g,h).

Στη συνέχεια ο αλγόριθμος **Mint** με είσοδο το params εκτυπώνει ένα νόμισμα και το trapdoor του με τον εξής τρόπο : διαλέγει S,r από το Z_q^* και υπολογίζει το $c=g^S h^r \bmod p$ ώστε το c να είναι πρώτος και να ανήκει στο σύνολο [A,B] .Αυτό το χρειαζόμαστε για να εντάξουμε το c στις τιμές που θα συσσωρευτούν με τον accumulator που περιγράφηκε παραπάνω. Το trapdoor του c είναι το (S,r). Ουσιαστικά η mint παράγει ένα νόμισμα και το trapdoor το οποίο θα χρειαστεί για να μπορείς να το ξοδέψεις.

Ο αλγόριθμος **Spend** δέχεται τις παραμέτρους ,ένα νόμισμα c ,το trapdoor sk_c ,ένα transaction string R και το σύνολο **C** που αποτελεί το σύνολο των νομισμάτων που έχουν γίνει mint και θα συσσωρευτούν στην τιμή A μέσω της συνάρτησης Accumulate. Αν το c δεν ανήκει στο **C** βγάζει το κενό. Αλλιώς ο αλγόριθμος υπολογίζει το A=Accumulate ((N,u),**C**) και το μάρτυρα w για το c στο A w=GenWitness((N,u),c,**C**).Στο τέλος εκτυπώνει το ζευγάρι (π,S) με το οποίο ο κάτοχος του c μπορεί να ξοδέψει το νόμισμά του. Το π είναι η signature of knowledge πάνω στο R του κάτοχου του c με την οποία ο κάτοχος αποδεικνύει ότι υπάρχει ένα νόμισμα c μέσα στο σύνολο **C**

1) για το οποίο ξέρει το μάρτυρά του w στο A (AccVerify ((N,u),A,c,w)=1)

2) το οποίο είναι η δέσμευση του S ($c=g^S h^r$) και για το οποίο γνωρίζει το r.

Τις τιμές (c,w,r) πρέπει να τις γνωρίζει μόνο ο κάτοχος του c έτσι ώστε να μην μαθευτεί πιο νόμισμα ξοδεύει ,γι αυτό χρησιμοποιείται signature of knowledge.

Ο αλγόριθμος **Verify** δέχεται το params, την π,το S, το R,το **C** και αφού υπολογίσει το A=Accumulate((N,u),C) ελέγχει την υπογραφή π πάνω στο R χρησιμοποιώντας τις γνωστές δημόσιες τιμές και αναλόγως εκτυπώνει 0 ή 1.

- $\mathsf{Setup}(1^\lambda) \to params$. On input a security parameter, run $\mathsf{AccumSetup}(1^\lambda)$ to obtain the values (N, u). Next generate primes p, q such that $p = 2^w q + 1$ for $w \geq 1$. Select random generators g, h such that $\mathbb{G} = \langle g \rangle = \langle h \rangle$ and $\mathbb{G}$ is a subgroup of $\mathbb{Z}_q^*$. Output $params = (N, u, p, q, g, h)$.
- $\mathsf{Mint}(params) \to (c, skc)$. Select $S, r \leftarrow \mathbb{Z}_q^*$ and compute $c \leftarrow g^S h^r \bmod p$ such that $\{c \text{ prime} \mid c \in [\mathcal{A}, \mathcal{B}]\}$.[11] Set $skc = (S, r)$ and output (c, skc).
- $\mathsf{Spend}(params, c, skc, R, \mathbf{C}) \to (\pi, S)$. If $c \notin \mathbf{C}$ output $\perp$. Compute $A \leftarrow \mathsf{Accumulate}((N, u), \mathbf{C})$ and $\omega \leftarrow \mathsf{GenWitness}((N, u), c, \mathbf{C})$. Output (π, S) where π comprises the following signature of knowledge:[12]

$$\pi = \mathsf{ZKSoK}[R]\{(c, w, r) : \mathsf{AccVerify}((N, u), A, c, w) = 1 \ \wedge \ c = g^S h^r\}$$

- $\mathsf{Verify}(params, \pi, S, R, \mathbf{C}) \to \{0, 1\}$. Given a proof π, a serial number S, and a set of coins $\mathbf{C}$, first compute

$A \leftarrow \mathsf{Accumulate}((N, u), \mathbf{C})$. Next verify that π is the aforementioned signature of knowledge on R using the known public values. If the proof verifies successfully, output 1, otherwise output 0.

Εικόνα από **[22]**

Στο πρωτόκολλο **[22]** υπάρχει η υπόθεση ότι υπάρχει μία έμπιστη αρχή που παράγει τις αρχικές παραμέτρους με την Set up. Αυτή η αρχή είναι απαραίτητη μόνο στην αρχή του πρωτοκόλλου.

2.5 ΕΦΑΡΜΟΓΗ ΤΟΥ ΠΡΩΤΟΚΟΛΛΟΥ ΣΤΟ BITCOIN

Ο τρόπος που εφαρμόζεται το παραπάνω πρωτόκολλο ώστε να έχουμε ανωνυμία στο bitcoin είναι ο εξής**[22]:**
Ας υποθέσουμε ότι η Alice θέλει να μετατρέψει μία ποσότητα (d bitcoins) σε ένα zerocoin το οποίο όταν ξοδευτεί να μην εμφανίζεται το public key της Alice . Η Alice θα ακολουθήσει τη παρακάτω διαδικασία:

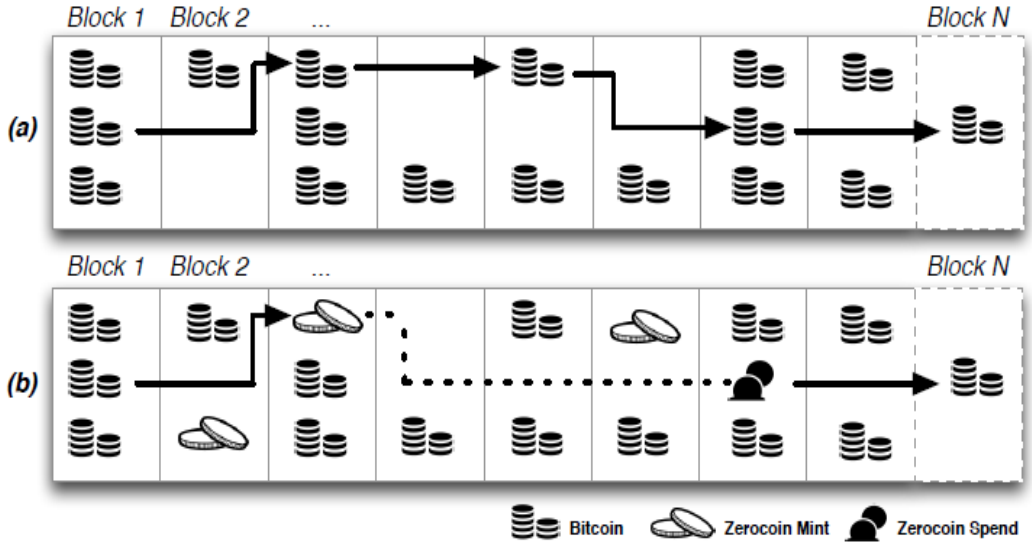
1)Αρχικά η Alice θα 'τρέξει' τον Mint αλγόριθμο με είσοδο το param που έχει προκαθοριστεί και θα πάρει σαν έξοδο ένα νόμισμα c(zerocoin) και το skc .Το skc πρέπει να το διατηρήσει κρυφό .

2)Στη συνέχεια θα δημιουργήσει ένα mint transaction με το οποίο εξαργυρώνει d+fees κλασσικά bitcoin .Στο output του transaction η Alice τοποθετεί το c.

3)Μόλις αυτό το transaction μπει στο blockchain το c συμπεριλαμβάνεται στο σύνολο που συσσωρεύεται στον γενικό accumulator A.

4)Για να ξοδέψει η Alice το zerocoin c και να το στείλει στον Bob:α) θα δημιουργήσει ένα μέρος του spend transaction (ptx) το οποίο θα περιλαμβάνει σαν input το reference από ένα από τα mint transactions που έχουν δημιουργηθεί και δεν έχουν χρησιμοποιηθεί σαν input άλλου transaction, και σαν output το public key του Bob .β) θα δημιουργήσει το σύνολο **C** με όλα τα έγκυρα mint transactions της αλυσίδας.γ) θα τρέξει τον αλγόριθμο Spend με είσοδο (params,c,skc,hash(ptx)=R,**C**) και θα πάρει το (π,S). δ) θα τοποθετήσει στο scriptsig του ptx το (π,S).

5) Μόλις το ptx εμφανιστεί οι nodes για να ελέγξουν ότι είναι έγκυρο α)θα τρέξουν τον αλγόριθμο Verify με είσοδο (params,π,S,hash(ptx),**C**) και θα ελέγξουν αν έχει output 1.β) θα ελέγξουν ότι το S δεν έχει ξαναεμφανιστεί σε άλλο spend transaction.γ)θα ελέγξουν ότι το referenced mint transaction του input δεν έχει ξαναχρησιμοποιηθεί σε άλλο spend transaction. Αν όλα είναι εντάξει το transaction θα μπει στην αλυσίδα και η Alice θα εξαργυρώσει 1 zerocoin αξίας d BITCOIN.



Εικόνα από [22]

ΠΑΡΑΤΗΡΗΣΕΙΣ

1)Σαν output του spend transaction μπορούσε αντί για το public key του παραλήπτη να τοποθετηθεί πάλι ένα zerocoin**.[22]**

2) Η ανωνυμία προέρχεται από το γεγονός ότι στο scriptsig αντί για την υπογραφή του αποστολέα υπάρχει μία signature of knowledge π πάνω σε μία απλή μορφή του spend transaction ,έτσι ώστε να μην φαίνεται ποιο zerocoin ξοδεύεται. Ταυτόχρονα δεσμεύεται ο αποστολέας ότι όντως γνωρίζει το trapdoor από ένα coin στο C το οποίο είναι δέσμευση του S. **[22]**

3)Το S χρησιμεύει για να μην έχουμε double spending **.[22]**

ΥΠΟΛΟΓΙΣΜΟΣ ΤΟΥ ACCUMULATOR

Από τον ορισμό του αλγορίθμου Verify φαίνεται ότι με κάθε επίκληση του Verify γίνεται υπολογισμός του accumulator A.

Αυτό το κόστος μπορεί να μειωθεί αν όπως προτείνεται στο **[22]** ο miner προσθέτει τα νέα νομίσματα που γίνονται mint στο παρόν block στον accumulator του

προηγούμενου block, και το νέο accumulator τον αποθηκεύει στο coinbase transaction.

ΝΕΟΙ ΤΥΠΟΙ TRANSACTION

Στο **[22]** προτείνεται να προστεθεί στο BITCOIN μία νέα εντολή : η ZERO-COIN_MINT. Με αυτή την προσθήκη στο scriptPubKey του mint transaction θα αναγράφεται αυτή η εντολή και το νόμισμα c.(Οι nodes θα πρέπει να ελέγξουν ότι το c έχει τη μορφή που πρέπει) .Στο spend transaction ,στο scriptsig θα αναγράφεται το (π,S) μαζί με το reference του block που περιέχει τον accumulator που χρησιμοποιήθηκε στην π. Έτσι οι nodes μπορούν να κάνουν verify χρησιμοποιώντας το συγκεκριμένο accumulator.

2.6 ΑΠΟΔΕΙΞΗ ΑΣΦΑΛΕΙΑΣ ΠΡΩΤΟΚΟΛΛΟΥ

2.6.1 BALANCE

1*) Αν η signature proof έχει soundness, στο random oracle μοντέλο, το strong RSA πρόβλημα είναι δύσκολο και το πρόβλημα του διακριτού λογαρίθμου είναι δύσκολο στο G τότε το παραπάνω πρωτόκολλο Π=(Setup,Mint,Spend,Verify) ικανοποιεί την balance ιδιότητα.**[22]**

ΑΠΟΔΕΙΞΗ

Ας υποθέσουμε ότι ένας αντίπαλος Α κερδίζει το balance game με μη αμελητέα πιθανότητα ε. Θα κατασκευαστεί ένας αλγόριθμος Β που παίρνει σαν input (p,q,g,h) ,όπου $G=\langle g\rangle = \langle h\rangle$ είναι μία υποομάδα της Z_p^* τάξης q, και εκτυπώνει $x \in Z_q$ τω $g^x \equiv h \pmod p$.

Ο Β λειτουργεί ως εξής**:[22]**

Με input (p,q,g,h) αρχικά χρησιμοποιεί την AccumSetup, για να παράγει τις παραμέτρους (N,u) και θέτει param=(N,u,p,q,g,h).

Στη συνέχεια θα προσπαθήσει να χρησιμοποιήσει τον αντίπαλο ,γι αυτόν θα προσπαθήσει να του προσφέρει σαν input ό,τι περιμένει από το balance game. Συνεπώς ο Β θα χρησιμοποιήσει την Mint με είσοδο param Κ φορές , θα πάρει Κ ζευγάρια (c_i,skc_i) ,όπου $skc_i =(S_i ,R_i)$και θα τρέξει τον αλγόριθμο Α του αντιπάλου με είσοδο $(param,c_1,..,c_k)$.

Κάθε φορά που ρωτάει ο αντίπαλος το oracle για κάποιο c_i ,ο Β απαντάει χρησιμοποιώντας το skc_i. .Αν $c_i \in \{ c_1,..,c_k\}$ τότε ο Β κρατάει το ζευγάρι (S_i,R_i). Έστω $(S_1,R_1),\ldots(S_l,R_l)$ τα ζευγάρια που έχει κρατήσει ο Β στο record.

Στο τέλος του παιχνιδιού ο Α εκτυπώνει ένα σύνολο από Μ coins $(c_1',\ldots,c'_M)$ και Μ+1 έγκυρες τετράδες $(\pi_i',S_i',R_i',\mathbf{C_i'})$.

Για j=1 μέχρι Μ+1 ο Β εφαρμόζει τον ZKSoK extractor στην π_j' προκειμένου να κάνει extract τις τιμές (c_j^*,r_j^*) και αναλόγως το αποτέλεσμα διακρίνονται οι εξής περιπτώσεις :

1) Ο extractor αποτυχαίνει → ο Β σταματάει (abort) και εκτυπώνει $EVENT_{EXT}$

2)c_j^* δεν ανήκει στο $\mathbf{C_j'}$ → ο Β σταματάει και εκτυπώνει $EVENT_{ACC}$

3) $c_j^* \in \{ c_1,..,c_k\}$ → a) Αν για κάποιο i ισχύει $(S_j',r_j^*) =(S_i ,r_i)$ και $R_j' \neq R_i$ τότε ο Β σταματάει και εκτυπώνει $EVENT_{FORGE}$

b) αλλιώς αν για κάποιο i ισχύει $(S_j',r_j^*) =(S_i ,r_i)$ τότε ο Β σταματάει και εκτυπώνει $EVENT_{COL}$

c) αλλιώς ο Β θέτει $(a,b)=(S_i,r_i)$.

4)Αν για κάποιο i $c_j^*= c_i^*$ ο Β θέτει $(a,b)=(S'_i,r_i^*)$.

Αν ο Β δεν έχει σταματήσει τότε έχουμε (c_j^*,r_j^*, S_j',a,b) και από το soundness του π έχουμε $c_j^* \equiv g^{S'_j}h^{r^*_j} \equiv g^a h^b$ (mod p). Ο Β εκτυπώνει $x=(S_j'-a)\cdot(b- r_j^*)^{-1}$ mod q και λύνεται το πρόβλημα του διακριτού λογαρίθμου.

ΑΝΑΛΥΣΗ

Σε περίπτωση που ο adversary Α κερδίζει το Balance game και ο αλγόριθμος Β δεν σταματάει (abort) μπορούμε να εξάγουμε $(c_1^*,\ldots,c_{M+1}^*)$ τω $\forall$ j$\in$ [1,M+1] , $c_j^* \in C_j' \in\{c_1,\ldots c_K,c_1',\ldots.,c'_M\}$ με κάθε S_j' μοναδικό και διαφορετικό από τα S του record του oracle. Αφού ο adversary παράγει Μ νομίσματα και ξοδεύει Μ+1 νομίσματα έχουμε δύο περιπτώσεις **[22] :**

1) Ο Α ξοδεύει ένα από τα νομίσματα που έχει για input ,αλλά παρέχει ένα νέο serial number S για αυτό (περίπτωση 3,a)

2)Ο Α έχει ξοδέψει το ίδιο νόμισμα δύο φορές ,αλλά χρησιμοποιώντας διαφορετικό serial number S (περίπτωση 4).

Σε κάθε τέτοια περίπτωση λύνεται ο διακριτός λογάριθμος.

Συνεπώς η πιθανότητα να υπολογίσει το διακριτό λογάριθμο ο Β είναι ίση με την πιθανότητα και να επιτύχει ο Α και να μην κάνει abort ο Β **[22].**

Pr(Bsucc)$=Pr(Asucc \cap Bnot\ abort)= Pr(A_{SUCC})- Pr\ (Asucc \cap Babort) \geq$

$Pr\ (A_{SUCC})- P\ (EVENT_{EXT} \cup EVENT_{ACC} \cup EVENT_{FORGE} \cup EVENT_{COL}) \geq$ ***ε-((M+1)$v_1(\lambda)$ + $v_2(\lambda)$+ $v_3(\lambda)$+1/l) [22]***

Pr($EVENT_{FORGE}$)≤ $v_3(\lambda)$ κάτω από την υπόθεση διακριτού λογαρίθμου.[22]

Pr($EVENT_{EXT}$)≤ (M+1)$v_1(\lambda)$ *,όπου $v_1(\lambda)$ είναι η αμελητέα πιθανότητα με την οποία ο extractor αποτυχαίνει σε input π.[22]*

Όταν συμβαίνει το $EVENT_{COL}$ ο Α έχει παράγει ένα ζευγάρι τω (S_j', r_j^*) $=(S_i, r_i)$,χωρίς το S'_j να έχει παραχθεί από το oracle (επειδή $R_j'=R_i$). Αν υποθέσουμε ότι υπάρχουν l ζευγάρια (S,r) για τα οποία ισχύει $c_j^*=g^S h^r$ η πιθανότητα ο Α να διαλέξει ένα από αυτά είναι μικρότερη από 1/l .Άρα ***Pr($EVENT_{COL}$)≤ 1/l .[22]***

Pr($EVENT_{ACC}$)≤$v_2(\lambda)$,κάτω από τη strong RSA υπόθεση.[22],[23]

Αποδεικνύεται **[22]** ότι ένας αντίπαλος Α΄ που εκτυπώνει $EVENT_{ACC}$ με μη αμελητέα πιθανότητα μετά από ένα balance game τότε μπορεί να χρησιμοποιηθεί για να βρούμε μάρτυρα w για ένα στοιχείο που δεν είναι μέλος του accumulator και στη συνέχεια **[23]** να λυθεί το strong RSA problem.

Έστω ότι έχουμε έναν αλγόριθμο Β' ο οποίος έχει για input ένα strong RSA instance (N,u) .

Στη συνέχεια ο Β διαλέγει (p,q,g,h) ,όπως στη setup και θέτει params=(N,u,p,q,g,h). Μετά με τη mint (param) παράγει τα νομίσματα $(c_1,\ldots c_K)$,καθώς και τα trapdoor αυτών .
Έπειτα μπορεί να αναπαραστήσει το oracle και να παράγει το περιβάλλον του balance game για να παράγει ο Α΄ ένα ζευγάρι (π'C') σαν output και (αν το κάνουμε extrac)t ένα c* που δεν ανήκει στο C'.
Ύστερα ο Β' κάνει extract w* από π'και το χρησιμοποιεί , όπως στο **[23]** για να λύσει το strong RSA problem.(στην απόδειξη όπου u' μπαίνει w* και όπου x' το c*)**[22]**

2.6.2 ANONYMITY

Θα χρησιμοποιηθεί ένας simulator ο οποίος παράγει με τη Setup (1^{λ}) τις παραμέτρους param .Στη συνέχεια διαλέγει δύο νομίσματα C_0,C_1 τα οποία είναι ομοιόμορφα κατανεμημένα στο σύνολο των πρώτων στο διάστημα [A,B] και τα δίνει στον A_1.Ο A_1 διαλέγει με όποια στρατηγική θέλει το C και το R. Έπειτα ο simulator παράγει μία simulated zero knowledge signature of knowledge π και έναν τυχαίο serial number S από το Z_q* και τα δίνει στον A_2. Τότε αν π είναι τουλάχιστον υπολογιστικά zero knowledge τότε όλες οι τιμές που παρέχονται στον αντίπαλο από τον simulator είναι με πιθανότητα 1 – negl κατανεμημένες όπως στο πραγματικό παιχνίδι του ANONYMITY. Όποτε αφού ο αντίπαλος εκτός αμελητέας πιθανότητας δεν μπορεί να διαχωρίσει το (π',S') που λαμβάνει από το πραγματικό παιχνίδι με το (π,S) που λαμβάνει από τον simulator και είναι τελείως ανεξάρτητο από το b ,η πιθανότητα να βρει το νόμισμα που έχει γίνει spend στο πραγματικό παιχνίδι είναι ½ +κάτι αμελητέο.**[22]**

3 ETHEREUM : A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER

3.1 ΕΝΑΛΛΑΚΤΙΚΕΣ ΕΦΑΡΜΟΓΕΣ ΑΠΟ ΤΟ BITCOIN

Η ύπαρξη της blockchain σαν μία κοινή συμφωνία για την ιστορία του bitcoin και η ιδέα να πραγματοποιείται proof of work για να βγει το επόμενο block είναι δύο πολύ σημαντικά χαρακτηριστικά της λειτουργίας του bitcoin ,τα οποία ενέπνευσαν τη δημιουργία πολλών εναλλακτικών εφαρμογών . (alternative blockchain applications) .Κάποιες από αυτές τις εφαρμογές σαν κοινή συμφωνία του δικτύου , χρησιμοποιούν την ίδια αλυσίδα με το bitcoin και κάποιες άλλες διαφορετική αλυσίδα .

Αυτοί είναι δύο διαφορετικοί τρόποι προσέγγισης **[20]** και καθένας από αυτούς, παρόλο που έχει αρκετά πλεονεκτήματα έχει και κάποιο μειονέκτημα. Οι εφαρμογές που χρησιμοποιούν την αλυσίδα του bitcoin (metacoins) χρησιμοποιούν τα blocks της αλυσίδας για να αποθηκεύσουν τα metacoin transactions. Το metacoin protocol έχει όμως μία σημαντική διαφορά από τον τρόπο που λειτουργεί το bitcoin :δε

μπορεί να αποφύγει την ύπαρξη μη έγκυρων metacoin transactions σε έγκυρο block ,απλά κάθε φορά που οι nodes βρίσκουν ένα μη έγκυρο tranaction το αγνοούν και δε μεταβάλλουν την κατάσταση . Έτσι κάθε node του δικτύου για να ελέγξει ότι ένα transaction είναι έγκυρο πρέπει να ερευνά από την αρχή την αλυσίδα ,κάτι το οποίο τον εξαναγκάζει να διαφυλάσσει όλη την blockchain ή να εμπιστεύεται ένα server για αυτό. Αντίθετα ένας light node στο bitcoin protocol μπορούσε να ελέγξει ότι ένα transaction είναι έγκυρο διαπιστώνοντας με SPV αν ανήκει σε κάποιο block της κύριας αλυσίδας ,η οποία μάλιστα αν απέχει και αρκετά blocks από οποιαδήποτε άλλη αλυσίδα υπάρχει πολύ μικρή πιθανότητα να θεωρηθεί αργότερα μη κύρια. **[20]**

Από την πλευρά εφαρμογές που έχουν ξεχωριστή αλυσίδα από το bitcoin χρησιμοποιούν ουσιαστικά άλλο δίκτυο από αυτό του bitcoin απαιτώντας άλλο proof of work. Αυτό έχει σαν αποτέλεσμα αν δημιουργηθούν πολλές τέτοιες εφαρμογές να διαμοιράζονται οι miners και να είναι πιο χρονοβόρο να βγαίνουν blocks σε περίπτωση που δε μειώνεται συνεχώς η δυσκολία του block**.[20]**

Όπως είδαμε στην προηγούμενη ενότητα ακόμα και χωρίς την εφαρμογή κάποιου metacoin protocol πάνω στο bitcoin protocol μπορούν να πραγματοποιηθούν πολλα smart contracts χρησιμοποιώντας την script-based language του bitcoin **[12]** . Στη συνέχεια θα περιγραφεί ένα αποκεντρωμένο σύστημα που στοχεύει στη πραγματοποίηση συμβολαίων.

3.2 ΣΚΟΠΟΣ ΔΗΜΙΟΥΡΓΙΑΣ ETHEREUM

Το ethereum **[20], [21]** είναι και αυτό ένα αποκεντρωμένο σύστημα που χρησιμοποιεί την ιδέα ύπαρξης μίας blockchain .

Οι δημιουργοί του **[20],[21]** στοχεύουν στο να αποτελέσει το ethereum ένα σύστημα (crypto- law system) το οποίο συμβάλλει στη δημιουργία συμφωνίας και συμβολαίων μεταξύ ανθρώπων που δεν εμπιστεύονται ο ένας τον άλλο ή ακόμα κάτι πιο ισχυρό : οι χρήστες του συστήματος να είναι σίγουροι ανεξαρτήτως με ποιους οργανισμούς ή ανθρώπους που αλληλεπιδρούν για τα πιθανά αποτελέσματα και τον τρόπο με τον οποίο προκύπτουν. Αν το σκεφτούμε αυτό είναι ιδιαίτερα δύσκολο ακόμα και στην πραγματική ζωή ,αφού η απόφαση ενός δικαστή επηρεάζεται από πάρα πολλούς παράγοντες που είναι δύσκολο να τους ορίσουμε όλους και ανά περίπτωση να είμαστε σίγουροι για την απόφασή του.

3.3 ΔΙΑΦΟΡΕΣ ETHEREUM ΑΠΟ ΤΟ BITCOIN

Κάποιες από τις διαφορές **[20]** με το bitcoin είναι οι παρακάτω:

1) Η γλώσσα που χρησιμοποιείται στο Ethereum **είναι *Turing complete* .** Αυτό σημαίνει ότι μπορούμε να χρησιμοποιήσουμε loop ,όπως είχαμε αναφέρει και σε προηγούμενη ενότητα ,με αποτέλεσμα να μη χρειάζεται να χρησιμοποιούμε πολλές φορές τη εντολή if .

2)Επίσης στη blockchain του bitcoin σε αντίθεση με το ethereum **δεν αποθηκεύεται αποτύπωμα της κατάστασης** ,η οποία δείχνει την ιδιοκτησία όλων των bitcoins, έτσι όπως διαμορφώνεται μετά από την ανακοίνωση κάθε block. Αυτό έχει σαν αποτέλεσμα εάν επιθυμεί κάποιος να καταγράψει την κατάσταση μιας δεδομένης στιγμής θα πρέπει να ξεκινήσει από το genesis block και ανά transaction κάθε block να εφαρμόζει τη συνάρτηση APPLY ως εξής: ***APPLY(S,TX)=S' OR ERROR* .**

Bitcoin As A State Transition System

Η κατάσταση κάθε στιγμής είναι το σύνολο από UTXO (unspent transaction outputs) .Τα transactions αυτά είναι έγκυρα ,έχουν μπει σε κάποιο block της κύριας αλυσίδας , αλλά δεν έχουν ξοδευτεί ακόμα , οπότε μπορούν να χρησιμοποιηθούν σαν inputs σε άλλα transactions.

Η συνάρτηση APPLY (S,TX) βγάζει error :

- ✓ Αν το output του referenced transaction του input του TX έχει ξοδευτεί και συνεπώς δεν ανήκει στο S
- ✓ Αν το input values είναι μικρότερα από τα output values
- ✓ Αν στο TX δεν υπάρχει το κατάλληλο scriptsig έτσι ώστε να μπορεί το input να ξοδευτεί.

Αλλιώς η APPLY υπολογίζει την S' =S –{INPUTS TX}+ {OUTPUTS TX}

3)Value-blindness : Στο bitcoin είναι δύσκολο να δημιουργηθούν συμβόλαια ασφάλειας ,όπως το παρακάτω : Ο Α θέλει μία ασφάλεια ότι εάν έχει στην κατοχή του y BTC αξίας 100$ θα συνεχίσει να έχει μετά από ένα μήνα BTC αξίας πάλι 100$

ανεξάρτητα από την ισοτιμία BTC και δολλαρίου. Έτσι μπορεί να πραγματοποιήσει ένα συμβόλαιο με έναν παίκτη Β , ο οποίος σε περίπτωση που πέσει η αξία του BTC σε σχέση με το δολλάριο θα δώσει στον Α τη διαφορά έτσι ώστε να έχει ο Α πάλι BTC συνολικής αξίας 100$,ενώ αν αυξηθεί η αξία του BTC θα κερδίσει y-χ BTC ,όπου χ είναι BTC αξίας 100$. Ο λόγος που είναι δύσκολο να δημιουργηθούν τέτοια συμβόλαια είναι ότι πρώτον πρέπει να υπάρχει ένα oracle που να δίνει κάθε φορά την ισοτιμία και δεύτερον ότι πρέπει να έχει υπογραφεί από τους παίκτες ένα transaction που να προβλέπει ανάλογα με την ισοτιμία όλους τους δυνατόν συνδυασμούς χρημάτων που στέλνονται στον Α και τον Β.

4) Blockchain-blindness : Επίσης είναι πιο δύσκολο στο bitcoin να δημιουργηθούν συμβόλαια στοιχήματος ,επειδή τα transactions δεν έχουν πρόσβαση σε πηγή τυχαιότητας που προκύπτει από το nonce ή το previous hash των blocks της blockchain ,αφού το περιεχόμενό τους δεν εξαρτάται από το block στο οποίο θα μπουν.

3.4 MODIFIED MERKLE PATRICIA TREE(TRIE)

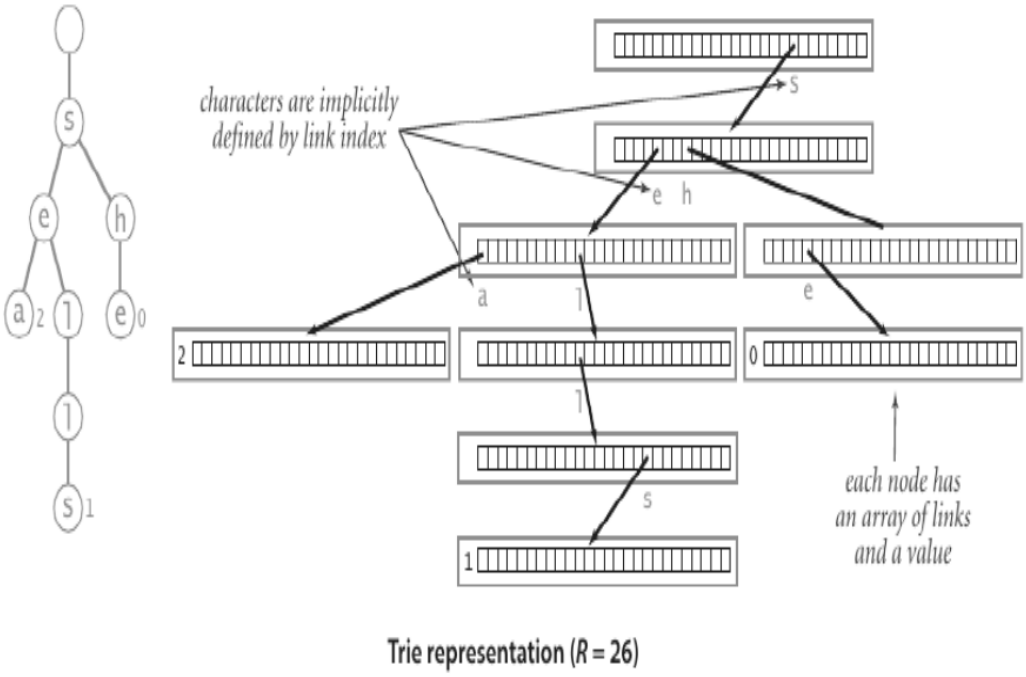
Σε αυτή την υποενότητα θα περιγραφεί ένα είδος βάσης δεδομένων (δέντρο) **[21]** που έχει στόχο να αποθηκεύει με πιο αποδοτικό τρόπο από το radix tree ζευγάρια της μορφής (key, value) ,όπου το key και το value είναι ακολουθία από bytes.

3.4.1 BASIC RADIX TREE

Αρχικά θα περιγραφεί η δομή ενός ***basic radix tree*.[16],[14]** Ένα radix tree αποτελείται από κόμβους της μορφής [value,$_{i0,}$……,i_{k-1}] ,όπου k είναι το μήκος του αλφάβητου στο οποίο θέλουμε να είναι γραμμένο το key(για παράδειγμα αν θέλουμε να είναι σε δεκαεξαδικό το k θα είναι 16 και ο κάθε κόμβος θα έχει 17 θέσεις. Το ii είναι είτε null είτε το hash ενός άλλου κόμβου .Επίσης υπάρχει ένα hash table ,ώστε να μπορεί να ανακτηθεί μία κορυφή από το hash της .

Αναζήτηση μίας value για ένα key σε ένα radix tree

Για **να αναζητήσουμε** το value που αντιστοιχεί σε ένα κλειδί ακολουθείται η παρακάτω διαδικασία: έστω ότι το key μας είναι 6a3f.Θα πάμε στη ρίζα στη θέση i_6(η πρώτη είναι η θέση για το value) και θα δούμε τι γράφει .Εκεί θα είναι αποθηκευμένο το hash της επόμενης κορυφής στη οποία πρέπει να πάμε (σαν link). Στη επόμενη κορυφή θα πάμε στην θέση i_{10} και θα κάνουμε το ίδιο έως ότου εξαντληθεί το κλειδί και φτάσουμε στην τελευταία κορυφή ,όπου στην πρώτη θέση θα υπάρχει το value που αναζητούμε**.[16],[14]**



Trie representation ($R = 26$)

Εικόνα από [14]

Update σε ένα radix tree

Ένας αλγόριθμος **[16]** για να γίνει update σε ένα radix tree , δηλαδή να πάρουμε τη ρίζα και ένα ζευγάρι (key, value) που θέλουμε να προσθέσουμε και να διαμορφώνουμε το νέο radix tree , είναι ο παρακάτω **[16]:**

```
def update(node,key,value):
    if key == '':
        curnode = db.get(node) if node else [ NULL ] * 17
        newnode = curnode.copy()
        newnode['value'] = value
    else:
        curnode = db.get(node) if node else [ NULL ] * 17
        newnode = curnode.copy()
        newindex = update(curnode[key[0]],key[1:],value)
        newnode[key[0]] = newindex
    db.put(hash(newnode),newnode)
    return hash(newnode)
```

(Εικόνα από **[16])**

Η ιδέα του παραπάνω αλγορίθμου είναι ότι **[16]** :

1)δέχεται *το hash της ρίζας του radix tree στο οποίο θέλουμε να κάνουμε update , και το ζευγάρι (key, value)που θέλουμε να προσθέσουμε* (έχει πρόσβαση στο hash table και το τροποποιεί μέσω των συναρτήσεων db.get και db.put αντίστοιχα)
2)Παίρνει ένα ένα τα γράμματα του key key[i] και κάθε φορά καλεί αναδρομικά τον εαυτό του με είσοδο(*το hash της κορυφής στην οποία είναι να μεταβούμε ,το υπόλοιπο κλειδί ,το αρχικό value*) μέχρι να εξαντληθεί το key .Για να βρίσκει κάθε φορά την επόμενη κορυφή που αντιστοιχεί στο hash που βρίσκεται στη θέση key[i] της προηγούμενης κορυφής χρησιμοποιεί το hash table. Αν φτάσει το key στο τελευταίο γράμμα του μέγιστου κοινού μέρους που έχει με τα keys του δέντρου με ρίζα ‘ node’ επειδή δεν υπάρχουν οι απαραίτητες κορυφές δημιουργούνται νέες.
3)Μόλις εξαντληθεί το κλειδί π.χ στη θέση j τοποθετεί στην τελευταία κορυφή το value, βρίσκει το hash της κορυφής και τροποποιεί τη θέση key[j] της προηγούμενης κορυφής και το hash table . Αυτό συνεχίζει να το κάνει μέχρι να φτάσει στη ρίζα αναδρομικά.
4) Στο τέλος επιστρέφει το hash της ρίζας του νέου δέντρου. **[16]**

Delete σε radix tree

```
def delete(node,key):
    if key == '' or node is NULL:
        return NULL
    else:
        curnode = db.get(node)
        newnode = curnode.copy()
        newindex = delete(curnode[key[0]],key[1:])
        newnode[key[0]] = newindex
        if len(filter(x -&gt; x is not NULL, newnode)) == 0:
            return NULL
```

Εικόνα από **[16]**

Το delete ζευγαριού (key,value) στον παραπάνω αλγόριθμο **[16]** γίνεται με τον ίδιο τρόπο αναδρομής μόνο που τώρα μόλις βρει την κορυφή που τελειώνει το key επιστρέφει null ,διαγράφοντας τα παιδιά της και επίσης στη συνέχεια σβήνει και τις περιττές ενδεχομένως κορυφές που υπάρχουν αν το key είχε πολύ μικρό κοινό μέρος με άλλα κλειδιά .Αυτό το κάνει επιστρέφοντας null μόλις μία κορυφή έχει όλες τις θέσεις null.

3.4.2 ΠΛΕΟΝΕΚΤΗΜΑΤΑ MODIFIED MERKLE PATRICIA TREE ΣΕ ΣΧΕΣΗ RADIX TREE

Παρατηρείται ότι **[14],[16]** για να αναζητήσουμε κλειδιά με κάποιο αρχικό κοινό μέρος ακολουθούμε ακριβώς την ίδια διαδρομή μέχρι το τελευταίο κοινό γράμμα. Αυτό έχει σαν αποτέλεσμα το value κλειδιών με κοινό μέρος να είναι κοντά αν σχηματίζαμε το radix tree. Το αρνητικό σε αυτό το σημείο είναι ότι αν μας είχε ζητηθεί να αποθηκεύσουμε ένα(key, value) ,όπου το key είναι πολύ μεγάλο και το κοινό του μέρος με άλλα κλειδιά είναι πολύ μικρό εμείς θα δημιουργούσαμε εξαιτίας μόνο αυτού του ζευγαριού πολλές κορυφές και επίσης στην αναζήτηση πάλι θα

έπρεπε να επισκεφτούμε εξίσου πολλές κορυφές .Επίσης έχουμε θέση value σε κάθε κορυφή ασχέτως αν υπάρχει κλειδί που έχει εξαντληθεί σε εκείνο το σημείο.
Αυτά τα σημεία βελτιώθηκαν στο modified merkle patricia tree **[21]** κάτι το οποίο το κάνει πιο αποδοτικό σε χώρο και χρόνο αναζήτησης ,μεγάλο πλεονέκτημα για μεγάλα keys .

3.4.3 ΕΙΔΗ ΚΟΡΥΦΩΝ MODIFIED MERKLE PATRICIA TREE

Συγκεκριμένα το **modified merkle patricia tree** για να το καταφέρει αυτό έχει και άλλα είδη κορυφών. Τα είδη κορυφών είναι τα παρακάτω **[16]:**

1) Null (αναπαριστάται σαν κενό string)
2) Ένας πίνακας με δύο αντικείμενα [key,value] **(extension node ή leaf node)**
3) Ένας πίνακας με δεκαεφτά αντικείμενα [a_0,..a_{15},at] **(branch node)**

Το **δεύτερο είδος** κορυφής **[16]** στη θέση key κάθε φορά έχει ένα κομμάτι κλειδιού και στη θέση value έχει κάποια τιμή μόνο στην περίπτωση που το key έχει **τερματικό** (δηλαδή η κορυφή μας είναι φύλλο,**leaf node**), αλλιώς στη θέση value υπάρχει μία άλλη κορυφή ή το hash της ανάλογα με το μέγεθος της. Σε αυτήν την περίπτωση η κορυφή μας είναι **extension node** και μας οδηγεί στην επόμενη κορυφή είτε επειδή δεν υπάρχει ζευγάρι (key,value) που το key να τελειώνει σε αυτό το σημείο ,οπότε δε χρειάζεται να υπάρχει τιμή είτε τελειώνει κάποιο κλειδί k1 εκεί ,αλλά υπάρχει key k_2 που να περιέχει k_1 ,οπότε χρειάζεται να υπάρχει η επόμενη κορυφή της διαδρομής.

Σε αυτή την περίπτωση η τιμή της k_1 βρίσκεται στην τελευταία θέση a_t της επόμενής κορυφής η οποία θα είναι μορφής 3 και αναλόγως το επόμενο ψηφίο του δεκαεξαδικού του k_2 π.χ j υπάρχει στην αντίστοιχη θέση της κορυφής το υπόλοιπο κομμάτι του k_2 και η τιμή του k_2 .

Αν υπήρχαν και άλλα κλειδιά π.χ k_3,k_4 με κοινό μέρος με το k_2 ,όπου k_3 μέρος του k_4 στη θέση j υπάρχει αναφορά της επόμενης κορυφής (για παράδειγμα k_1=do,k_2=dog,k_3=doge).

Παρατηρούμε **[16]** ότι η **κορυφή μορφής 3** έχει τόσες θέσεις όσα τα ψηφία του δεκαεξαδικού όπου μπορεί να έχει null ή το υπόλοιπο κλειδί και τιμή ή επόμενη κορυφή διαδρομής .Υπάρχει στο τέλος και άλλη μία θέση που έχει null ή τιμή όπως είδαμε προηγουμένως.

Όπως είδαμε παραπάνω **[16]** μία κορυφή μορφής 2 μπορεί στη θέση value να έχει τιμή (leaf node) ή άλλη κορυφή και να είναι extension node. Για να διαχωρίζουμε τα δύο είδη κορυφών χρειαζόμαστε μία κωδικοποίηση των keys που να το προβλέπει αυτό και να δείχνει αν το κάθε κλειδί είναι τερματικό ή όχι. Γι αυτόν και για άλλους

λόγους που θα περιγραφούν παρακάτω χρησιμοποιείται ο παρακάτω αλγόριθμος **[16]** κωδικοποίησης των keys που δέχεται κλειδιά με τη μορφή nibbles(αριθμος που αναπαριστάνεται από 4 bit ή ένα ψηφίο δεκαεξαδικού-κάθε byte μπορεί να αναπαρασταθεί από δύο nibbles) και να εξάγει μία ακολουθία bytes.

3.4.4 HEX –PREFIX ENCODING

Καταρχήν πρέπει να τονιστεί ότι επειδή χρειάζονται 2 ψηφία δεκαεξαδικού για να αναπαραστήσουμε ένα byte πρέπει να τα keys να έχουν άρτιο πλήθος ψηφίων .Μία λύση ήταν αν ένα key είχε περιττό αριθμό δεκαεξαδικών ψηφίων να προσθέταμε στην αρχή ένα 0 .Όμως σε αυτήν την περίπτωση δε θα ξεχωρίζαμε μεταξύ τους τα κλειδιά 0f574 και f574 **[16].** Γι αυτό τον λόγο πρέπει να ξέρουμε αν το αρχικό πλήθος είναι άρτιο ή περιττό. Συνεπώς είναι χρήσιμο να υπάρχει μία σημαία η οποία να δείχνει το parity του κλειδιού και αν είναι τερματικό ή όχι. Η σημαία που χρησιμοποιείται στον παρακάτω αλγόριθμο **[16]** είναι ένα nibble που μπαίνει στην αρχή του key και διαμορφώνεται ως εξής : το λιγότερο σημαντικό bit είναι 1 αν το key είναι περιττο και 0 αν είναι άρτιο και το δεύτερο λιγότερο σημαντικό bit είναι 1 αν το key έχει τερματικό και 0 αλλιώς. Αν το key είναι άρτιο ,επειδή με το αρχικό nibble θα γίνει περιττό ,μετά το πρώτο nibble προσθέτουμε το 0.

$$\mathrm{HP}(\mathrm{x},t):\mathrm{x}\in\mathbb{Y} \equiv \begin{cases} (16f(t),16\mathrm{x}[0]+\mathrm{x}[1],16\mathrm{x}[2]+\mathrm{x}[3],\ldots) & \text{if } \|\mathrm{x}\| \text{ is even} \\ (16(f(t)+1)+\mathrm{x}[0],16\mathrm{x}[1]+\mathrm{x}[2],16\mathrm{x}[3]+\mathrm{x}[4],\ldots) & \text{otherwise} \end{cases}$$

$$f(t) \equiv \begin{cases} 2 & \text{if } t \\ 0 & \text{otherwise} \end{cases}$$

(Εικόνα από **[21])**

Παραπάνω είναι η συνάρτηση **[21]** η οποία δέχεται ένα πίνακα από nibbles x με μία boolean μεταβλητή στο τέλος t=1,αν το x έχει τερματικό και 0 αλλιώς και εκτυπώνει το key στη μορφη που θέλουμε. Παρακάτω είναι ένας αλγόριθμος **[16]** που υλοποιεί το παραπάνω.

```
def compact_encode(hexarray):
    term = 1 if hexarray[-1] == 16 else 0
    if term: hexarray = hexarray[:-1]
    oddlen = len(hexarray) % 2
    flags = 2 * term + oddlen
    if oddlen:
        hexarray = [flags] + hexarray
    else:
        hexarray = [flags] + [0] + hexarray
    // hexarray now has an even length whose first nibble is the flags.
    o = ''
    for i in range(0,len(hexarray),2):
        o += chr(16 * hexarray[i] + hexarray[i+1])
    return o
```

Εικόνα από **[16]**

(16*hexarray[i] +hexarray[i+1] μετατρέπει δύο nibbles στο αντίστοιχο byte).

Παράδειγμα: [0, 15, 1, 12, 11, 8, 16] →'\x02\x0f\x1c\xb8'

3.4.5 RLP ENCODING METHOD

Η RLP **[21] ,[17]** είναι μία μέθοδος κωδικοποίησης πινάκων με bytes έτσι ώστε να μπορούν να αποθηκευτούν στη σειρά διάφορα αντικείμενα από bytes(items) χωρίς να υπάρχει σύγχυση. Αυτή η μέθοδος χρησιμοποιείται τόσο για να αποθηκευτούν τα values και οι κορυφές του modified patricia tree, όσο και σε άλλα σημεία του ethereum ,όπως θα δούμε αργότερα. Τα items μπορεί να είναι ένα string ή ακόμη και μία λίστα από items.

Πιο συγκεκριμένα ο αλγόριθμος RLP **[17]** κωδικοποίησης πραγματοποιεί τα εξής:

1) Αν είναι ένα μόνο byte στο διάστημα [0X00, 0x7f] τότε δεν μεταβάλλεται καθόλου με την RLP κωδικοποίηση.
2) Αν έχουμε ένα string μήκους 0-55 τότε το αποτέλεσμα της κωδικοποίησης

έχει στην αρχή ένα byte που είναι 0x80 +το μήκος του string, στο διάστημα [0x80, 0xb7] και μετά το string.

3) Αν έχουμε ένα string μήκους μεγαλύτερου από 55 τότε η κωδικοποίηση δίνει στην αρχή ένα byte που είναι ίσο με το 0xb7+ το μήκος του μήκους του string σε δυαδική μορφή , στο διάστημα [0xb8, 0xbf], στη συνέχεια το μήκος του string και τέλος το string **(π.χ** για string μήκους 1037 είναι /xb9/x04/x0c και το string).
4) Αν έχουμε μία λίστα από items με αθροιστικό μήκος 0-55 ,τότε το πρώτο byte είναι 0xc0+το μήκος του string , στο διάστημα [0xc0, 0xf7], ακολουθούμενο από την rlp κωδικοποίηση των items.
5) Αν έχουμε μία λίστα με αθροιστικό μήκος μεγαλύτερο από 55 bytes τότε το πρώτο byte είναι το 0xf7+ το μήκος του μήκους της λίστας σε δυαδική μορφή ,στο διάστημα [0xf8, 0xff],ακολουθούμενο από το μήκος της λίστας ακολουθούμενο από την rlp κωδικοποίηση των inputs.

```
def rlp_encode(input):
    if isinstance(input,str):
        if len(input) == 1 and chr(input) < 128: return input
        else: return encode_length(len(input),128) + input
    elif isinstance(input,list):
        output = ''
        for item in input: output += rlp_encode(item)
        return encode_length(len(output),192) + output

def encode_length(L,offset):
    if L < 56:
        return chr(L + offset)
    elif L < 256**8:
        BL = to_binary(L)
        return chr(len(BL) + offset + 55) + BL
    else:
        raise Exception("input too long")

def to_binary(x):
    return '' if x == 0 else to_binary(int(x / 256)) + chr(x % 256)
```

Εικόνα από [17]

<u>ΠΑΡΑΤΗΡΗΣΕΙΣ ΣΤΟΝ ΑΛΓΟΡΙΘΜΟ</u>

- Στην encode_length το L μπορεί να φτάσει μέχρι τον μέγιστο ακέραιο που μπορεί να αναπαρασταθεί με 8 bytes
- Το offset κάθε φορά μας δείχνει πρώτο κομμάτι του αθροίσματος για να βρούμε το πρώτο byte αναλόγως με το εάν είναι string ή λίστα (0x80 ή 0xc0)
- Στην encode_length μέσα στην ch αθροίζουμε το 55 + offset που είναι κάθε φορά το 0xb7 και το 0xf7
- Η συνάρτηση to_binary μετατρέπει έναν ακέραιο σε ακολουθία bytes. Για παράδειγμα το 1037 σε /x04/x0c **. [17]**

3.4.6 ΠΕΡΙΓΡΑΦΗ ΤΟΥ MODIFIED MERKLE PATRICIA TREE

Έστω ότι έχουμε ένα σύνολο από ζευγάρια (key ,value) που είναι ακολουθίες από bytes $J = \{(k_0 \in B;\ v_0 \in B),\ (k_1 \in B;\ v_1 \in B),\ldots.\}$τα οποία θέλουμε να αποθηκεύσουμε σε μία βάση δεδομένων και να έχουμε ένα πιστοποιητικό που τα αντιπροσωπεύει. Η βάση δεδομένων θα είναι το δέντρο που θα περιγραφεί παρακάτω και το πιστοποιητικό είναι το hash της ρίζας του, που όπως θα δούμε οποιαδήποτε αλλαγή σε κορυφή του δέντρου το μεταβάλλει ,υπό την προϋπόθεση ότι η hash είναι collision resistance. Αυτό κιόλας είναι και το χαρακτηριστικό του merkle tree που θέλαμε να έχει το modified merkle patricia tree**.[21]**

Πιο συγκεκριμένα ορίζεται η συνάρτηση **[21]** TRIE (*J*) ως το πιστοποιητικό.

$$\texttt{TRIE}(\mathfrak{I}) \equiv \begin{cases} () & \text{if} \quad \mathfrak{I} = \varnothing \\ \texttt{SHA3}(c(\mathfrak{I}, 0)) & \text{otherwise} \end{cases}$$

(Εικόνα από **[21]**)

Η αναφορά σε ένα ζευγάρι του *J* θα γίνεται ως εξής ∀ I ∈ *J* I≡(I_0,I_1)

Μπορούν να μετατραπούν τα keys του συνόλου *J* σε σειρά από nibbles, αντί για σειρά από bytes. Υποθέτουμε ότι έχουμε big endian επισήμανση, δηλαδή ο υπολογιστής διαβάζει από αριστερά στα δεξιά. Για να γίνει αυτό παίρνουμε ένα byte (τον integer που το αντιπροσωπεύει)και αφού κάνουμε την ευκλείδεια διαίρεση με το 16 το πηλίκο δίνει το πρώτο nibble και το υπόλοιπο το δεύτερο.

Έτσι διαμορφώνεται το σύνολο y(*J)* ως εξής **[21]:**

$$y(\mathfrak{J}) = \{(\mathbf{k}'_0 \in \mathbb{Y}, \mathbf{v}_0 \in \mathbb{B}), (\mathbf{k}'_1 \in \mathbb{Y}, \mathbf{v}_1 \in \mathbb{B}), ...\}$$

$$\forall_n \quad \forall_{i:i<2\|\mathbf{k}_n\|} \quad \mathbf{k}'_n[i] \equiv \begin{cases} \lfloor \mathbf{k}_n[\lfloor i \div 2 \rfloor] \div 16 \rfloor & \text{if } i \text{ is even} \\ \mathbf{k}_n[\lfloor i \div 2 \rfloor] \bmod 16 & \text{otherwise} \end{cases}$$

(Εικόνα από **[21]**)

Όπως είδαμε όταν περιγράφαμε τα είδη των κορυφών πολλές φορές μέσα σε μία κορυφή υπάρχει αναφορά σε άλλη κορυφή .Κάθε κορυφή κωδικοποιείται με RLP και αν το αποτέλεσμα της κωδικοποίησης είναι μικρότερο από 32 bytes τότε η αναφορά στην κορυφή γίνεται αυτούσια ,αλλιώς μέσω της SHA-3 hash,ενώ υπάρχει ένα table με αποθηκευμένες τις αντίστοιχες κορυφές με τα hash τους. Αυτό φαίνεται παρακάτω **[21]** ,όπου έχουμε τη συνάρτηση n όπου δείχνει πως αποθηκεύεται μία κορυφή c(*J* ,i).

$$n(\mathfrak{J}, i) \equiv \begin{cases} () & \text{if} \quad \mathfrak{J} = \varnothing \\ c(\mathfrak{J}, i) & \text{if} \quad \|c(\mathfrak{J}, i)\| < 32 \\ \texttt{SHA3}(c(\mathfrak{J}, i)) & \text{otherwise} \end{cases}$$

(Εικόνα από **[21]**)

Σε αυτό το σημείο θα περιγραφούν **[21]** πώς ακριβώς δημιουργούνται οι κορυφές c(*J* ,i) του modified merkle patricia tree αν μας δοθεί ένα σύνολο *J*.

$$c(\mathfrak{I},i) \equiv \begin{cases} \mathrm{RLP}\left(\left(\mathrm{HP}(I_0[i..(\|I_0\|-1)], true), I_1\right)\right) & \text{if } \|\mathfrak{I}\|=1 \text{ where } \exists I : I \in \mathfrak{I} \\ \mathrm{RLP}\left(\left(\mathrm{HP}(I_0[i..(j-1)], false), n(\mathfrak{I},j)\right)\right) & \text{if } i \neq j \text{ where } j = \arg\max_x : \exists \mathfrak{l} : \|\mathfrak{l}\| = x : \forall_{I \in \mathfrak{I}} : I_0[0..(x-1)] = 1 \\ \mathrm{RLP}\left(\left(u(0), u(1), ..., u(15), v\right)\right) & \text{otherwise where } u(j) \equiv n(\{I : I \in \mathfrak{I} \wedge I_0[i] = j\}, i+1) \end{cases}$$

$$v = \begin{cases} I_1 & \text{if } \exists I : I \in \mathfrak{I} \wedge \|I_0\| = i \\ () & \text{otherwise} \end{cases}$$

(Εικόνα από **[21]**)

- Το πρώτο κομμάτι της αναδρομικής σχέσης αναφέρεται σε κορυφές (key,value) τύπου 2 που είναι leaf nodes. Έχουμε τέτοιο είδος κορυφής σε περίπτωση που το *J* έχει πλέον μόνο ένα ζευγάρι.
- Το δεύτερο κομμάτι της αναδρομικής αναφέρεται σε κορυφές τύπου 2 (key,value) που είναι extension nodes. Έχουμε αυτό το είδος κορυφής σε περίπτωση που υπάρχει κοινό μέρος όλων των keys του *J* . Σε αυτήν την περίπτωση αναζητούμε το μέγιστο κοινό κομμάτι όλων των keys του συνόλου *J* και βάζουμε το hp του σαν key ,και σαν value αναδρομικά την επόμενη κορυφή.
- Το τρίτο κομμάτι της αναδρομικής αναφέρεται σε branch node. Έχουμε τέτοιο είδος σε περίπτωση που έχουμε παραπάνω από ένα ζευγάρια στο *J* και το I0[i] δεν είναι το ίδιο για όλα τα I∈ *J*. Σε αυτή την περίπτωση χωρίζουμε το *J* σε υποσύνολα που έχουν κοινό I0[i] =j και σε κάθε θέση u(j) αντίστοιχα βάζουμε την επόμενη κορυφή αναδρομικά με το κατάλληλο σύνολο ζευγαριών. Σε περίπτωση που υπήρχε ζευγάρι που το πλήθος των στοιχείων του key είναι ίσο με i βάζουμε στην τελευταία θέση το value του ζευγαριού. Μία τέτοια περίπτωση είναι όταν η προηγούμενη κορυφή είναι extension ,αλλά υπήρχε κλειδί που τελείωνε σε εκείνο το σημείο. **[21]**

<u>ΠΑΡΑΔΕΙΓΜΑ</u>

Έστω ότι έχουμε **[19]** τα ζευγάρια {('dog', 'puppy'), ('horse', 'stallion'), ('do', 'verb'), ('doge', 'coin')} ή αλλιώς αν τα μετατρέψουμε

```
[ 6, 4, 6, 15, 16 ] : 'verb'
[ 6, 4, 6, 15, 6, 7, 16 ] : 'puppy'
[ 6, 4, 6, 15, 6, 7, 6, 5, 16 ] : 'coin'
[ 6, 8, 6, 15, 7, 2, 7, 3, 6, 5, 16 ] : 'stallion'
```

Εικόνα από **[19]**

Τότε η αναδρομική σχέση **[21]** λειτουργεί ως εξής **[19]:**

Πλήθος J≠1 και όλα τα ζευγάρια έχουν κοινό το 6.
Άρα **c(J,0)**=RLP((HP(6,false), n(J,1)))

Το επόμενο στοιχείο των keys δεν είναι κοινό σε όλα τα κλειδιά, άρα θα φτιάξουμε branch node

c(J,1)=RLP(NULL,NULL,NULL,NULL,n(J1,2),NULL,NULL,NULL,n(J2,2) ,NULL,NULL, NULL,NULL,NULL,NULL,NULL,' '), όπου **J1** περιέχει τα ζευγάρια με κλειδιά do,dog,doge και **J2** περιέχει ζευγάρι με κλειδί horse.

c(J1,2) =RLP(HP(6 15 ,false),n(J1,3))

c(J1,3)=RLP (NULL, NULL,NULL,NULL,NULL,NULL,n(J1',4),NULL,NULL,NULL,NULL,NU LL,NULL,NULL,NULL,NULL,'**verb**'),όπου **J1'** περιέχει ζευγάρια με κλειδιά dog,doge

c(J1',4)= RLP(HP(7,false),n(J1',5))

c(J1',5)=RLP(NULL,NULL,NULL,NULL,NULL,NULL,n(J1'',6),NULL,NU LL,NULL,NULL,NULL,NUL,LNULL,NULL,'**puppy**'),όπου **J1''**έχει το ζευγάρι με κλειδί doge.
c(J1'',6)=RLP(HP(5,true),'**coin**')

c(J2,2)=RLP(HP(6,15,7,2,2,3,6,5,true),'**stallion'**)

3.5 ΑΝΑΛΥΤΙΚΗ ΠΕΡΙΓΡΑΦΗ ΔΟΜΗΣ ETHEREUM

3.5.1 BLOCKCHAIN PARADIGM

Το ethereum **[20], [21]** μπορεί να θεωρηθεί ως μία transaction –based state machine(κάτι στο οποίο έχει ομοιότητα με το bitcoin) με την έννοια ότι ξεκινάμε από ένα genesis state και εκτελούμε ανά έγκυρο block που βγαίνει τα transactions που συμπεριλαμβάνονται σε αυτό σχηματίζοντας την final state του ethereum ,η οποία αποτυπώνεται στο blockheader , όπως θα δούμε παρακάτω σε αντίθεση με το bitcoin

. Το transaction αναπαριστά μία έγκυρη μετάβαση μεταξύ καταστάσεων. Υπάρχουν και invalid αλλαγές κατάστασης οι οποίες είναι ενδιάμεσες, όπως για παράδειγμα η αφαίρεση χρημάτων από ένα λογαριασμό χωρίς να προστεθούν σε ένα άλλο.

- ✓ Η συνάρτηση που περιγράφει την valid αλλαγή της κατάστασης $\boldsymbol{\sigma_t}$ σε $\boldsymbol{\sigma_{t+1}}$ με την εφαρμογή ενός transaction T είναι η **Y** (**ethereum state transition function) [21] :**

  $$\boldsymbol{\sigma_{t+1} \equiv Y(\sigma_t, T)}$$

- ✓ Η συνάρτηση **[21]** που περιγράφει την αλλαγή της κατάστασης $\boldsymbol{\sigma_t}$ σε $\boldsymbol{\sigma_{t+1}}$ με την εφαρμογή ενός block B $\equiv(\ldots..,(T_0,T_1,\ldots..))$ είναι η **Π(block-level state transition function) :**

  $$\boldsymbol{\sigma_{t+1} \equiv \Pi(\sigma_t, B)}$$

- ✓ Η συνάρτηση **[21**] που επιβραβεύει το miner (με coinbase transaction) είναι η **Ω(block finalization state transition function)**

  $$\boldsymbol{\Pi(\sigma,B) \equiv \Omega(B, Y(Y(\sigma,T_0),T_1)\ldots.)}$$

  Το ETHEREUM έχει εσωτερικό συνάλλαγμα τα Ether (ETH) με μικρότερη υποδιαίρεση τα Wei **[21].**
  1 Ether$\rightarrow 10^{18}$ Wei

Multiplier	Name
10^0	Wei
10^{12}	Szabo
10^{15}	Finney
10^{18}	Ether

Εικόνα από [21]

3.5.2 WORLD STATE

Με τον όρο world state(state)**[21]** εννοούμε την αντιστοίχηση μεταξύ addresses(160 bits) και accounts states ,οι οποίες θα περιγραφούν παρακάτω. Αν θεωρήσουμε για keys τα addresses και για values τις account states αφου τις έχουμε σειρικοποιήσει μέσω της RLP τότε μπορεί αυτή η αντιστοίχηση να αναπαρασταθεί με ένα modified merkle patricia tree το οποίο αποθηκεύεται ξεχωριστά από το blockchain ,αλλά το hash της ρίζας του που συνδέεται και επηρεάζεται από όλα τα στοιχεία του δέντρου αποθηκεύεται στο blockheader σαν πιστοποιητικό για την παρούσα κατάσταση.**[21]**

3.5.3 ACCOUNT STATE

Με τον όρο account state εννοούμε **[21]** τα περιεχόμενα ενός account σε μία συγκεκριμένη κατάσταση (state).

Υπάρχουν δύο ειδών accounts **[21]** : α) **contract accounts** β) **non contract accounts**.

Τα non contract accounts ή αλλιώς simple accounts αντιστοιχούν στους nodes του δικτύου ενώ τα contract accounts είναι οντότητες που δημιουργούνται με ειδικά transactions και βοηθούν στην πραγματοποίηση αξιόπιστων συμβολαίων .

Τα account states αποτελούνται από τα παρακάτω πεδία **[21]** :

- ✓ **Nonce:** Είναι ένας αριθμός που δείχνει σε περίπτωση που έχουμε simple account τον αριθμό transactions που έχει στείλει αυτό το address και σε περίπτωση που έχουμε contract account τον αριθμό των contract creations που έχουν γίνει από αυτό το account. Εάν έχουμε το account ενός address α σε μία κατάσταση σ αυτό το πεδίο συμβολίζεται $\sigma[\alpha]_n$.

- ✓ **Balance :** Ο αριθμός των Wei που κατέχει η συγκεκριμένη διεύθυνση. Συμβολίζεται $\sigma[\alpha]_b$.

- ✓ **Stateroot :** μία 256-bit hash της ρίζας ενός trie structure που κωδικοποιεί τα περιεχόμενα του storage του account .Συμβολίζεται $\sigma[\alpha]_s$.

- ✓ **codeHash:** Αυτό το πεδίο είναι $\neq \emptyset$ μόνο σε contract accounts και είναι το hash του EVM code που έχει το account και εκτελείται μόλις λάβει ένα message call. Σε αντίθεση με άλλα πεδία δε μπορεί να μεταβληθεί μετά τη δημιουργία του. Συμβολίζεται με $\sigma[\alpha]_c = SHA3(b)$,όπου **b** είναι ο κώδικας. Σε περίπτωση που έχουμε simple account $\sigma[\alpha]_c = SHA3(())$.

Όσον αφορά το **trie structure που κωδικοποιεί το storage** στην πραγματικότητα έχει αποθηκεύσει ζευγάρια (key,value) ,όπου keys είναι ακολουθίες από 32 bytes και τα values είναι RLP κωδικοποιημένοι integers. **[21]**

Συγκεκριμένα έχουμε $\sigma[\alpha]_s = TRIE(L_I^*(\sigma[\alpha]_s))$,όπου α) $L_I(k,v)=(k,RLP(v))$, με $k \in B_{32}$ και $v \in P$, β) L_I^* δέχεται πολλά ζευγάρια και κάνει το ίδιο με την L_I σε όλα (το ίδιο θα ισχύει συμβολιστικά και για κάθε συνάρτηση f* → $f^*((x_0, x_1, ...)) \equiv (f(x_0), f(x_1), ...)$ εικόνα από **[21]**

και γ) **TRIE((χ_1, υ_1)(,χ_2,υ_2),…..)** επιστρέφει το hash της ρίζας του trie structure που έχει αποθηκευμένα τα ζευγάρια που έχει ως είσοδο. (**B_X** είναι ακολουθία bytes μήκους x και **P** είναι το σύνολο των θετικών ακεραίων). **[21]**

Όσον αφορά ***το modified merkle patricia tree (trie***) **[21]** που αποτυπώνει την world state έχουμε:

$$L_S(\boldsymbol{\sigma}) \equiv \{p(a) : \boldsymbol{\sigma}[a] \neq \varnothing\}$$

$$p(a) \equiv \left(a, \texttt{RLP}\left((\boldsymbol{\sigma}[a]_n, \boldsymbol{\sigma}[a]_b, \boldsymbol{\sigma}[a]_s, \boldsymbol{\sigma}[a]_c)\right)\right)$$

(Εικόνα από **[21]**)

Όπου $L_s(\sigma)$ είναι το σύνολο ζευγαριών (key, value) ,όπου το key είναι ίσο με α ,το address, και το value είναι ίσο με το RLP του account state ,ώστε παρόλο που γράφεται το account state σαν ακολουθία από bytes να έχει μία δομή και να μπορεί να ανακτηθεί .

Για κάθε address και κατάσταση για να έχουμε ένα έγκυρο account state πρέπει να ισχύουν τα παρακάτω **[21]** :

$$\forall a : \boldsymbol{\sigma}[a] = \varnothing \ \vee \ (a \in \mathbb{B}_{20} \ \wedge \ v(\boldsymbol{\sigma}[a]))$$

(Εικόνα από **[21]**)

Όπου υ είναι η συνάρτηση ελέγχου εγκυρότητας του account **[21]** (account validity function)

$$v(x) \quad \equiv \quad x_n \in \mathbb{P} \wedge x_b \in \mathbb{P} \wedge x_s \in \mathbb{B}_{32} \wedge x_c \in \mathbb{B}_{32}$$

(Εικόνα από **[21]**)

Το addresss πρέπει να είναι 160 bits ,το nonce και το balance πρέπει να είναι θετικοί ακέραιοι και το stateroot και codehash πρέπει να είναι 256 bits ,λόγω hash.

3.5.4 ΔΟΜΗ TRANSACTION

Έχουμε δύο ειδών transaction**[21]** : τα transactions που καταλήγουν σε message call, όπως θα δούμε παρακάτω και τα transactions με τα οποία έχουμε contract creation ,δηλαδή δημιουργία contract account.

Τα πεδία που περιέχουν είναι τα παρακάτω **[21]** :

- ✓ **nonce:** Είναι ο αριθμός των transactions που έχει στείλει ο sender .Συμβολίζεται $\mathbf{T_n}$.

- ✓ **gasprice** :ο αριθμός των Wei που δίνεται ανά μονάδα gas για το κόστος υπολογισμού που προκύπτει από την εκτέλεση του transaction από τον miner .Συμβολίζεται $\mathbf{T_p}$.
- ✓ **Gaslimit**: ο μέγιστος αριθμός gas που επιτρέπεται να χρησιμοποιηθεί προκειμένου να εκτελεστεί το transaction. Συμβολίζεται $\mathbf{T_g}$.

- ✓ **To :** σε περίπτωση που έχουμε το πρώτο είδος transaction είναι το address του account του παραλήπτη του message call ,ενώ σε περίπτωση που έχουμε transaction που καταλήγει σε contract account είναι zero address. Συμβολίζεται με $\mathbf{T_t}$.

- ✓ **Value :** Είναι ο αριθμός των Wei που δίνονται στον παραλήπτη του message call ή σε περίπτωση που έχουμε contract creation είναι η παρακαταθήκη που θα έχει το contract account. Συμβολίζεται με $\mathbf{T_v}$.

- ✓ **V,r,s:** τιμές που σχετίζονται με την υπογραφή του transaction και χρησιμοποιούνται για τον καθορισμό του αποστολέα . Συμβολίζονται με T_w , T_r, T_s.

- ✓ **Init :** αυτό το πεδίο **υπάρχει μόνο σε transactions που οδηγούν σε contract creation.** Είναι ένας πίνακας απεριόριστου μήκους με bytes που προσδιορίζει ένα κομμάτι κώδικα(EVM –code) που καθορίζει την αρχική διαδικασία στη δημιουργία του contract account . Ουσιαστικά με την εκτέλεση αυτού του κώδικα δημιουργείται το σώμα (body) του contract account, δηλαδή ο κώδικάς του που μετά τη δημιουργία μένει αμετάβλητος και εκτελείται κάθε φορά που το contract account δέχεται ένα message call , είτε μέσω transaction είτε

εξαιτίας του ίδιου του κώδικα (body). Ο κώδικας στο init εκτελείται μία φορά και μετά διαγράφεται

Συμβολίζεται με $\mathbf{T_i}$.

- ✓ **Data :** υπάρχει **μόνο σε message –call transactions**. Είναι ένας απροσδιόριστου μήκους byte array που περιέχει τα input data του message call. Συμβολίζεται με $\mathbf{T_d}$.

Με το σύμβολο S(T) θα αναφερόμαστε **[21]** στον αποστολέα του transaction.

Συνοψίζοντας έχουμε **[21]**

$$L_T(T) \equiv \begin{cases} (T_n, T_p, T_g, T_t, T_v, T_{\mathbf{i}}, T_w, T_r, T_s) & \text{if } T_t = 0 \\ (T_n, T_p, T_g, T_t, T_v, T_{\mathbf{d}}, T_w, T_r, T_s) & \text{otherwise} \end{cases}$$

(Εικόνα από **[21]**)

, όπου στην πρώτη περίπτωση έχουμε transaction που οδηγεί σε contract account και γι αυτό T_t=0 και έχουμε T_i ,ενώ στη δεύτερη έχουμε message call transaction και έχουμε T_d.

$$\begin{array}{llllll} T_n \in \mathbb{P} & \wedge & T_v \in \mathbb{P} & \wedge & T_p \in \mathbb{P} & \wedge \\ T_g \in \mathbb{P} & \wedge & T_w \in \mathbb{P} & \wedge & T_r \in \mathbb{P} & \wedge \\ T_s \in \mathbb{P} & \wedge & T_{\mathbf{d}} \in \mathbb{B} & \wedge & T_{\mathbf{i}} \in \mathbb{B} & \wedge \\ T_t \in \mathbb{B}_{20} & & & & & \end{array}$$

(Εικόνα από **[21]**)

<u>ΠΑΡΑΤΗΡΗΣΕΙΣ</u>

Παρατηρείται ότι τα transactions στο ethereum μπορεί να έχουν πιο πολύπλοκη μορφή με αποτέλεσμα να χρειάζονται περισσότερο χρόνο για να ελεγχθούν και να εκτελεστούν από τους miners .Για να υπάρχει το κίνητρο για τους miners να συμπεριλαμβάνουν στο block τους περίπλοκα transactions προβλέπεται ανταμοιβή ανάλογα με τα υπολογιστικά βήματα που εκτελούν . Έτσι στα transactions υπάρχουν κάποια πεδία και συγκεκριμένα το gasprice και gaslimit που συνδέονται με το gas που είναι τα computational steps του miner όταν εκτελεί τα transactions**.[21]**

3.5.5 BLOCK

Το block **[21]** αποτελείται τρία μέρη:

α) το blockheader **H** του παρόντος block

β) πληροφορίες **R** που σχετίζονται με τα transactions(*transaction receipts*)

γ) λίστα **U** από uncle blockheaders που είναι blockheaders των blocks (uncles) που έχουν κοινό πατέρα με τον πατέρα του παρόντος block . Το block μπορεί να αναπαρασταθεί $B \equiv (B_H, B_R, B_U)$.

BLOCKHEADER

- ✓ **parentHash :** Το SHA3 256 –bit hash ολόκληρου του προηγούμενου block (του πατέρα). Συμβολίζεται με $\mathbf{H_p}$.
- ✓ **unclesHash :** Το SHA3 256 –bit hash της λίστας των uncles του παρόντος block. Συμβολίζεται $\mathbf{T_u}$.
- ✓ **Coinbase :** Η 160-bit address που καταλήγουν όλα τα fees από τα επιτυχημένο mining αυτού του block. Συμβολίζεται $\mathbf{H_b}$.
- ✓ **Stateroot :** Το SHA3 256 –bit hash της ρίζας του state tree που απεικονίζει την κατάσταση(world state) μετά από την εφαρμογή της συνάρτησης Π με είσοδο το stateroot του προηγούμενου block και το παρόν block. Συμβολίζεται $\mathbf{H_r}$.
- ✓ **transactionsTrie**: Το SHA3 256-bit hash του trie structure στο οποίο αποθηκεύονται όλα τα transactions του παρόντος block. Συμβολίζεται $\mathbf{H_t}$.
- ✓ **difficulty :** ένας αριθμός που δείχνει τη δυσκολία του παρόντος block και θα δούμε στη συνέχεια πώς υπολογίζεται. Συμβολίζεται $\mathbf{H_d}$.

- ✓ **number :** ο αριθμός των προηγούμενων blocks στην αλυσίδα που βρίσκεται το παρόν block. Το genesis block θεωρούμε ότι έχει number 0. Συμβολίζεται $\mathbf{H_i}$.

- ✓ **minGasPrice :** ο αριθμός που δείχνει το μικρότερο gasprice που επιτρέπεται να έχουν τα transactions του παρόντος block ,ώστε να είναι αποδοτικό το mining. Συμβολίζεται $\mathbf{H_m}$.

- ✓ **gaslimit :** το τρέχον όριο του gas που επιτρέπεται να χρησιμοποιηθεί για την εκτέλεση όλων των transactions . Συμβολίζεται $\mathbf{H_l}$.

- ✓ **gasUsed :** Το συνολικό gas που χρησιμοποιήθηκε στο παρόν block για την εκτέλεση όλων των transactions. . Συμβολίζεται $\mathbf{H_u}$.

- ✓ **timestamp :** η τιμή ίση με το output της Unix's time() τη στιγμή που βγήκε το block. . Συμβολίζεται $\mathbf{H_s}$.

- ✓ **extradata :** ένας πίνακας με bytes που περιέχει στοιχεία του block . Με εξαίρεση το genesis block θα είναι 32 bytes ή λιγότερα. . Συμβολίζεται $\mathbf{H_x}$.

- ✓ **nonce :** ένα 256-bit hash που αποδεικνύει ότι έχει γίνει αρκετός υπολογισμός σε αυτό το block. Συμβολίζεται $\mathbf{H_n}$ **[21]**

<u>ΠΑΡΑΤΗΡΗΣΕΙΣ</u>

Κάποιες βασικές διαφορές με το bitcoin είναι **[21]:**

1) ότι στο blockheader υπάρχει το stateroot που είναι το αποτύπωμα της κατάστασης μετά την εφαρμογή των transactions του block.

2) Ότι υπάρχουν στο blockheader πεδία που σχετίζονται με το gas που ξοδεύτηκε για τα transactions του block και με την τιμή του gas.

3) Ότι στο block συμπεριλαμβάνονται και blockheaders uncle blocks ,κάτι που όπως θα δούμε παρακάτω σε συνδυασμό και με νέα συμφωνία για το ποια είναι η κύρια αλυσίδα θα διευκολύνει στην αντιμετώπιση του selfish mining.

3.5.6 TRANSACTION RECEIPT

Το transaction receipt **[21]** είναι μία τριπλέτα που αποτελείται από τρία μέρη και αφορά κάθε transaction.:

A) Το παρόν transaction, συμβολίζεται με $\mathbf{R_T}$.
Β) το world state μετά τη εφαρμογή του παρόντος transaction, συμβολίζεται με $\mathbf{R_\sigma}$.
Γ) το συνολικό gas που έχει χρησιμοποιηθεί για το block μέχρι και την εφαρμογή του παρόντος transaction, συμβολίζεται με $\mathbf{R_u}$ **,** όπου $R_u \in P$

Έτσι έχουμε **[21]** :

$$R \equiv (R_T, R_\sigma, R_u)$$

(Εικόνα από **[21]**)

Επειδή θα χρειαστεί RLP κωδικοποίηση το transaction receipt η συνάρτηση που το μετατρέπει σε κατάλληλη μορφή είναι **η $L_R(R)$=** $L_R((R_T , R_\sigma , R_u)$ **)**

Για την οποία ισχύει :

$$L_R(R) \equiv (L_T(R_T), \texttt{TRIE}(L_S(R_\sigma)), R_u)$$

(Εικόνα από **[21]**)

Η δεύτερη συντεταγμένη είναι το hash της ρίζας του trie structure που αναπαριστά την κατάσταση R_σ.

Με B_T συμβολίζουμε τη λίστα των transaction receipts που αντιστοιχούν σε όλα τα transactions του block ,όπως φαίνεται παρακάτω :

$$B_{\mathbf{T}} \equiv (B_{\mathbf{R}}[0]_T, B_{\mathbf{R}}[1]_T, ...)$$

(Εικόνα από **[21]**)

Επίσης θα χρειαστεί RLP κωδικοποίηση πέρα το transaction receipt και το blockheader ,καθώς και ολόκληρο το block .Έτσι οι αντίστοιχες συναρτήσεις **[21]** που θα το φέρουν σε κατάλληλη μορφή είναι οι $\mathbf{L_H}$,$\mathbf{L_B}$,όπως περιγράφονται παρακάτω :

$$
\begin{aligned}
L_H(H) &\equiv (\ H_p, H_u, H_b, H_r, H_t, H_d, \\
&\qquad H_i, H_m, H_l, H_u, H_s, H_x, H_n\ ) \\
L_B(B) &\equiv \big(L_H(B_H), L_R^*(B_{\mathbf{R}}), L_H^*(B_{\mathbf{U}})\big)
\end{aligned}
$$

(Εικόνα από **[21]**)

$$
\begin{array}{llllll}
H_p \in \mathbb{B}_{32} & \wedge & H_u \in \mathbb{B}_{32} & \wedge & H_b \in \mathbb{B}_{20} & \wedge \\
H_r \in \mathbb{B}_{32} & \wedge & H_t \in \mathbb{B}_{32} & \wedge & H_d \in \mathbb{P} & \wedge \\
H_i \in \mathbb{P} & \wedge & H_m \in \mathbb{P} & \wedge & H_l \in \mathbb{P} & \wedge \\
H_u \in \mathbb{P} & \wedge & H_s \in \mathbb{P} & \wedge & H_x \in \mathbb{B} & \wedge \\
H_n \in \mathbb{B}_{32} & & & & &
\end{array}
$$

(Εικόνα από **[21]**)

Η $\mathbf{L_H}$ ουσιαστικά παραθέτει σειριακά τα διάφορα πεδία του blockheader που τα περιγράψαμε παραπάνω ,όπως κάνει και **η** $\mathbf{L_T}$ με τα πεδία του transaction.

Η $\mathbf{L_B}$ παραθέτει τα τρία κύρια μέρη του block ,όπου χρησιμοποιεί την παραπάνω συνάρτηση $\mathbf{L_H}$ για το header του παρόντος block ,όπως και για τα headers των uncles. Για τα transaction receipts που αντιστοιχούν στα transactions όλου του παρόντος block χρησιμοποιείται η $\mathbf{L_R^*}$,που εφαρμόζει L_R σε κάθε transaction receipt. **[21]**

3.5.7 VALIDITY ΕΝΟΣ BLOCK

Για να θεωρηθεί ένα block έγκυρο **[21]** πρέπει να ακολουθηθεί η παρακάτω διαδικασία :

$$
\begin{array}{rcll}
H_u' & \equiv & \texttt{SHA3}(\texttt{RLP}(L_H^*(B_{\mathbf{U}}))) & \wedge \\
H_t & \equiv & \texttt{TRIE}(\{\forall i < \|B_{\mathbf{T}}\|, i \in \mathbb{P} : p(i, L_R(B_{\mathbf{R}}[i])\})) & \wedge \\
H_r & \equiv & \texttt{TRIE}(L_S(\Pi(\boldsymbol{\sigma}, B))) &
\end{array}
$$

(Εικόνα από **[21]**)

1)Να ελεγχθεί ότι το hash της λίστας των uncles B_U του παρόντος block,αφού έχει εφαρμοστεί σε αυτήν L^*_H και RLP είναι ίση με το H_u που εμφανίζεται στο blockheader του παρόντος block .

2)Να ελεγχθεί αν το H_t του blockheader είναι όντως το hash της ρίζας του structure που αποθηκεύονται τα transactions του παρόντος block ή πιο συγκεκριμένα τα ζευγάρια P(index transaction στο block ,transaction receipt) ≡(RLP(index transaction),RLP(transaction receipt)).

3)Να ελεγχθεί αν το H_r είναι όντως το hash της ρίζας του Merkle Patricia Tree που απεικονίζει το world state έτσι όπως έχει διαμορφωθεί μετά την εφαρμογή της συνάρτησης Π πάνω στην κατάσταση **σ** στην οποία είχαμε μείνει μετά την εφαρμογή του πατέρα του παρόντος block $P(B_H)$.

Ισχύει :

$$\texttt{TRIE}(L_S(\boldsymbol{\sigma})) = P(B_H)_{H_r}$$

(Εικόνα από **[21]**)

4)Να ελεγχθεί αν $\mathbf{H_p} \equiv$ SHA3(RLP(P(B_H)).

5)Να ελεγχθούν τα πεδία H_m,H_i, H_u.

6)Η blockheader validity function V να είναι true.

$$
\begin{aligned}
V(H) \quad \equiv \quad & \texttt{PoW}(H, H_n) \leqslant \frac{2^{256}}{H_d} \quad \wedge \\
& H_d = D(H) \quad \wedge \\
& H_l = L(H) \quad \wedge \\
& H_s > P(H)_{H_s} \quad \wedge \\
& \|H_x\| < 1024
\end{aligned}
$$

(Εικόνα από **[21]**)

<u>ΟΡΙΣΜΟΙ ΚΑΙ ΠΑΡΑΤΗΡΗΣΕΙΣ</u>

- ✓ Το **difficulty** ενός blockheader Η συμβολίζεται **D(H)** και ορίζεται ως εξής **[21]:**

$$
D(H) \equiv \begin{cases} 2^{22} & \text{if} \quad H_i = 0 \\ P(H)_{H_d} + \lfloor \frac{P(H)_{H_d}}{1024} \rfloor & \text{if} \quad H_s < P(H)_{H_s} + 42 \\ P(H)_{H_d} - \lfloor \frac{P(H)_{H_d}}{1024} \rfloor & \text{otherwise} \end{cases}
$$

(Εικόνα από **[21]**)

Παρατηρείται ότι το difficulty των blocks αυξομειώνεται προκειμένου να διατηρείται όσο το δυνατόν πιο σταθερός ο ρυθμός που βγαίνουν τα blocks. Συγκεκριμένα η δυσκολία του block μειώνεται μόλις το timestamp του παρόντος block έχει γίνει μεγαλύτερο από αυτό του πατέρα του +42,έτσι ώστε να μην καθυστερήσει πολύ να βγει το block**.[21]**

- ✓ Το timestamp του block **[21]** πρέπει να είναι μεγαλύτερο από το timstamp του πατέρα του : $\mathbf{H_s > P(H_s)}$

- ✓ Το **gaslimit [21]** κάθε φορά μεταβάλλεται σε συνάρτηση με το $\mathbf{P(H)_{Hl}}$,δηλαδή το gaslimit του πατέρα του block και με το $\mathbf{P(H)_{Hu}}$

,δηλαδή το άθροισμα του gas που χρησιμοποιήθηκε για να εκτελεστούν όλα τα transactions του πατέρα του block.

$$L(H) \equiv \begin{cases} 10^6 & \text{if} \quad H_i = 0 \\ 125000 & \text{if} \quad L'(H) < 125000 \\ L'(H) & \text{otherwise} \end{cases}$$

$$L'(H) \equiv \lfloor \frac{1023 P(H)_{H_l} + \lfloor \frac{6}{5} P(H)_{H_u} \rfloor}{1024} \rfloor$$

(Εικόνα από **[21]**)

- ✓ Το $\mathbf{H_n}$ **[21]** θα πρέπει να ικανοποιεί την σχέση $\mathbf{POW(H,H_n) \leq \frac{2^{256}}{H_d}}$.

Προφανώς όσο μεγαλώνει το H_d ,τόσο μεγαλώνουν οι αναμενόμενες δοκιμές nonce που απαιτούνται για να βρεθεί το κατάλληλο.

3.5.8 ΕΚΤΕΛΕΣΗ ΤΩΝ TRANSACTION

Στο ethereum επειδή έχουμε Turing complete γλώσσα είναι σημαντικό να προβλέψουμε τι θα πραγματοποιηθεί σε περίπτωση που έχουμε έναν ατέρμονα βρόγχο σε ένα κομμάτι κώδικα, όπως για παράδειγμα στο T_i ενός transaction, λόγω λάθους ή λόγω spamming. **[20],[21]**

Επίσης δεδομένου ότι τα transaction στο ethereum δεν εκφράζουν απλά τη μεταφορά χρημάτων από ένα public key σε ένα άλλο, αλλά μέσω αυτών πραγματοποιούνται πιο πολύπλοκες διαδικασίες είναι απαραίτητο οι miners να έχουν ένα κίνητρο να συμπεριλάβουν ένα πιο πολύπλοκο transaction στο block τους. **[20],[21]**

Τα δύο παραπάνω προβλήματα λύνονται **[20]** , **[21]** ενσωματώνοντας στα transactions τα πεδία gasprice T_p και gaslimit T_g όπως είδαμε παραπάνω . Το gaslimit ουσιαστικά δείχνει το μέγιστο αριθμό computational steps, που εκφράζονται με gas, τα οποία ο miner θα πραγματοποιήσει για να εκτελέσει το transaction .Για κάθε gas ο miner πληρώνεται όσο είναι το gasprice. Ακόμα και σε περίπτωση που το gas δεν επαρκεί για την εκτέλεση του transaction ο miner πληρώνεται για όσα computational steps έχει πραγματοποιήσει. Επίσης σε περίπτωση που το limitgas είναι περισσότερο από το gas που απαιτήθηκε για την εκτέλεση του transaction ο miner θα πληρωθεί μόνο για το gas που πραγματοποίησε**.[21]**

Στη συνέχεια θα περιγραφεί **[21]** πότε ένα ***transaction είναι έγκυρο.***

Σαν g_o θεωρούμε το αρχικό gas που απαιτείται για να ξεκινήσει η εκτέλεση του transaction . Σε περίπτωση που έχουμε message call transaction το αρχικό gas είναι $\|T_d\|$,δηλαδή πλήθος bytes του πεδίου data , πολλαπλασιαζόμενο με το gas που έχει οριστεί ανά byte data + το gas που απαιτείται εξαρχής με τη δημιουργία ενός transaction ($G_{transaction}$). Σε περίπτωση που έχουμε contract creation transaction το αρχικό gas είναι $\|T_i\|$, δηλαδή πλήθος bytes του initialisation EVM code, πολλαπλασιαζόμενο με το gas που έχει οριστεί ανά byte του T_i + $G_{transaction}$.

$$g_0 \equiv \begin{cases} \|T_\mathbf{i}\| G_{txdata} + G_{transaction} & \text{if} \quad T_t = 0 \\ \|T_\mathbf{d}\| G_{txdata} + G_{transaction} & \text{otherwise} \end{cases}$$

(Εικόνα από **[21]**)

1)Για να θεωρηθεί ένα transaction έγκυρο είναι απαραίτητο το g_o να είναι μικρότερο ή ίσο από το gaslimit .

2)Επίσης πρέπει η υπογραφή του sender να είναι έγκυρη.

3)Ακόμη πρέπει το nonce του transaction να είναι το ίδιο με το nonce του account του sender S(T).

4)Το balance του sender θα πρέπει να έχει τουλάχιστον τόσα ether όσα το v_0 που είναι ίσο με τα ether που θα σταλούν στο miner αν εξαντλήσει το gaslimit + τα ether που πρέπει να σταλούν στον παραλήπτη του transaction.

$$v_0 \equiv T_g T_p + T_v$$

(Εικόνα από **[21]**)

Ένα ακόμα πράγμα που πρέπει να ελέγξει ο miner **[21]** για να θεωρηθεί το transaction ότι είναι έγκυρο και να ξεκινήσει να το εκτελεί είναι ότι το συνολικό gas που έχει ξοδέψει για να εκτελέσει όλα τα προηγούμενα transactions $l(B_R)_u$ + το gaslimit αυτού του transaction δεν ξεπερνούν το συνολικό gas B_{Hl} που επιτρέπεται για το block.

Η συνάρτηση που συνοψίζει τα παραπάνω είναι η εξής **[21]** :

$$
\begin{aligned}
S(T) &\neq \varnothing \quad \wedge \\
\boldsymbol{\sigma}[S(T)] &\neq \varnothing \quad \wedge \\
T_n &= \boldsymbol{\sigma}[S(T)]_n \quad \wedge \\
g_0 &\leq T_g \quad \wedge \\
v_0 &\leq \boldsymbol{\sigma}[S(T)]_b \quad \wedge \\
T_l &\leq B_{Hl} - \ell(B_{\mathbf{R}})_u
\end{aligned}
$$

(Εικόνα από **[21]**)

Έτσι και θεωρηθεί έγκυρο το transaction ξεκινά η εκτέλεση του με πρώτα βήματα **[21]** :

1) να αυξηθεί το nonce του account του sender κατά ένα.
2) να αφαιρεθεί από το gaslimit T_g το g_0 για να υπολογιστεί το υπόλοιπο διαθέσιμο gas g.
3) Να αφαιρεθεί από το balance του account του sender v_0 ether.

Αν θεωρήσουμε **σ** την αρχική world state τότε η προσωρινή κατάσταση που προκύπτει μετά τα παραπάνω βήματα είναι η **σ_0** **[21]**.

$$
\begin{aligned}
\boldsymbol{\sigma}_0 &\equiv \boldsymbol{\sigma} \quad \text{except:} \\
\boldsymbol{\sigma}_0[S(T)]_b &\equiv \boldsymbol{\sigma}[S(T)]_b - v_0 \\
\boldsymbol{\sigma}_0[S(T)]_n &\equiv \boldsymbol{\sigma}[S(T)]_n + 1
\end{aligned}
$$

(Εικόνα από **[21]**)

Σαν **σ_p** θα θεωρήσουμε **[21]** την προσωρινή κατάσταση μετά την εκτέλεση του transaction , χωρίς να έχει αποζημιωθεί ακόμα ο miner. Η **σ_p** μαζί με το gas g' ,που δεν έχει χρησιμοποιηθεί και έχει περισσέψει από το gaslimit, και τη suicide list s, που θα δούμε στη συνέχεια, προκύπτουν μέσω των συναρτήσεων Θ_3 και Λ ανάλογα με το αν έχουμε transaction message call ή contract creation transaction. **[21]**

$$
(\boldsymbol{\sigma}_P, g', \mathrm{s}) \equiv \begin{cases} \Lambda(\boldsymbol{\sigma}_0, S(T), T_o, g, T_p, T_v, T_{\mathbf{i}}) & \text{if} \quad T_t = 0 \\ \Theta_3(\boldsymbol{\sigma}_0, S(T), T_o, T_t, g, T_p, T_v, T_{\mathbf{d}}) & \text{otherwise} \end{cases}
$$

(Εικόνα από **[21]**)

$$
g \equiv T_g - g_0
$$

(Εικόνα από **[21]**)

Στη συνέχεια θα έχουμε την κατάσταση **[21]** **σ*** ,όπου το g'·gasprice ether επιστρέφονται στο balance του account του sender και (Tg-g')·gasprice ether πηγαίνουν στο coinbase H_b που είναι το balance του account του miner.

$$\begin{aligned} \boldsymbol{\sigma}^* &\equiv \boldsymbol{\sigma}_P \quad \text{except} \\ \boldsymbol{\sigma}^*[s]_b &\equiv \boldsymbol{\sigma}_P[s]_b + g' T_p \\ \boldsymbol{\sigma}^*[m]_b &\equiv \boldsymbol{\sigma}_P[m]_b + (T_g - g') T_p \\ m &\equiv B_{H\,b} \end{aligned}$$

(Εικόνα από **[21]**)

Τέλος η τελική κατάσταση **σ'** προκύπτει αν διαγράψουμε όλα τα accounts που βρίσκονται στην suicide list. **[21]**

$$\begin{aligned} \boldsymbol{\sigma}' &\equiv \boldsymbol{\sigma}^* \quad \text{except} \\ \forall i \in \mathbf{s} : \boldsymbol{\sigma}'[i] &\equiv \varnothing \end{aligned}$$

(Εικόνα από **[21]**)

Συνοψίζοντας οι προσωρινές καταστάσεις οι οποίες αναφέραμε είναι **σ→σ0 →σ_p →σ*→σ'.**

3.5.9 CONTRACT CREATION

Σαν creation function **[21]** θεωρούμε τη Λ που αναφέραμε παραπάνω ,η οποία δέχεται σαν είσοδο μία κατάσταση σ,το sender S(T) , τον original transactor **o**, το διαθέσιμο gas g, την gasprice p , το value του transaction v και τον initialization EVM code I και μας δίνει τη νέα κατάσταση σ' μετά τη δημιουργία του contract το υπολειπόμενο gas g' και τη suicide list s.

$$(\boldsymbol{\sigma}', g', \mathbf{s}) \equiv \Lambda(\boldsymbol{\sigma}, s, o, g, p, v, \mathbf{i})$$

(Εικόνα από **[21]**)

Πριν όμως αναλυθεί η Λ θα δούμε κάποιες άλλες συναρτήσεις που χρησιμοποιεί ,όπως **η Α** και **η Ξ** . **[21]**

Η Α ουσιαστικά δημιουργεί το address του contract account με τον εξής τρόπο : παίρνει το nonce και τον sender ,εφαρμόζει RLP και μετά hash και κρατάει από τα 256 bits τα τελευταία 160 bits. Αυτά θα αποτελέσουν το address που επιθυμούμε. Αν αυτό το address που προκύπτει ανήκει σε άλλο account ,τότε αυξάνουμε τη διεύθυνση κατά ένα αφού την μετατρέψουμε σε έναν big endian ακέραιο. **[21]**

$$
\begin{aligned}
A(s,n) &\equiv a \quad \text{where:} \\
a &= \arg\min_{x} : x \geqslant a' \wedge \boldsymbol{\sigma}[x] = \varnothing \\
a' &= \mathcal{B}_{96..255}\Big(\texttt{SHA3}\Big(\texttt{RLP}\big(\ (s,n)\ \big)\Big)\Big)
\end{aligned}
$$

(Εικόνα από **[21]**)

Για να προκύψει η επόμενη κατάσταση **σ*** από **την σ** εφαρμόζεται η Α και το account με το address που δημιουργείται έχει μηδέν nonce, αφού δεν έχει στείλει ακόμη κανένα transaction, balance ίσο με το value του transaction ,κενό storage και code hash ίσο με το hash του κενού.

Εδώ πρέπει να επισημανθεί ότι το nonce μειώνεται κατά ένα πριν μπει για είσοδο της Α έτσι ώστε το n να είναι το nonce του sender τη στιγμή που είχε δημιουργηθεί το transaction.(Πριν εφαρμόσουμε την Α είχε αυξηθεί το nonce του sender κατά ένα από προηγούμενο στάδιο) **[21]**

$$
\begin{aligned}
\boldsymbol{\sigma}^* &\equiv \boldsymbol{\sigma} \quad \text{except:} \\
\boldsymbol{\sigma}^*[A(s,\boldsymbol{\sigma}[s]_n - 1)] &\equiv \big(0, v, \texttt{TRIE}(\varnothing), \texttt{SHA3}(())\big)
\end{aligned}
$$

(Εικόνα από **[21]**)

Στη συνέχεια πριν φτάσουμε στη **σ'** πρέπει να εφαρμοστεί ο αρχικός κώδικας EVM code **i,** με σκοπό να δημιουργηθεί το body του contract account. Για να γίνει αυτό απαιτείται gas. Σε περίπτωση που δεν επαρκέσει το gas(out of gas exception) η κατάσταση επανέρχεται στην κατάσταση πριν εφαρμοστεί το contract creation ,και η κατάσταση του νέου account που δημιουργήθηκε θεωρείται το κενό. **[21]**

Η Ξ ουσιαστικά σε περίπτωση out of gas δημιουργεί την κατάσταση σ** με σ**[α] να είναι κενό, ενώ σε άλλη περίπτωση αφήνει την κατάσταση ως έχει και δίνει το body του νέου account σε ακολουθία bytes **o** και το υπολειπόμενο gas. **[21]**

Έτσι μέσω της Ξ(code execution function) διαμορφώνουμε την κατάσταση σ' **[21]** .

$$(\sigma^{**}, g', \mathbf{s}, \mathbf{o}) \equiv \Xi(\sigma^*, g, I)$$

$$\sigma' \equiv \begin{cases} \sigma^{**} & \text{if} \quad \sigma^{**}[a] = \varnothing \\ \sigma^{**} \quad \text{except:} & \\ \sigma'[a]_{\mathbf{b}} = \mathbf{o} & \text{otherwise} \end{cases}$$

$$I_a \equiv a$$
$$I_o \equiv o$$
$$I_p \equiv p$$
$$I_{\mathbf{d}} \equiv ()$$
$$I_s \equiv s$$
$$I_v \equiv v$$
$$I_{\mathbf{b}} \equiv \mathbf{i}$$

(Εικόνα από **[21]**)

ΠΑΡΑΤΗΡΗΣΕΙΣ

1) Αφού το body έχει διαμορφωθεί μπορεί να στέλνει message calls, να δημιουργεί άλλα contract accounts και να τροποποιεί το storage του εφόσον ο κώδικας το προβλέπει ή ως αποτέλεσμα ενός message call με παραλήπτη αυτό**.[21]**
2) Σε περίπτωση που η εκτέλεση του κώδικα i οδηγήσει σε κενό body επειδή i είναι κενό ή επειδή έχει κάπου stop πριν δημιουργηθεί body τότε δεν διαγράφεται το account, απλά μένουν δεσμευμένα τα v ether για πάντα αφού δε μπορεί το account να αλληλεπιδράσει. Με αυτό τον τρόπο έχουμε τη δημιουργία ενός zombie account, **.[21]**

3.5.10 MESSAGE CALL

Στην περίπτωση που το transaction μας είναι τύπου message call αντί για τη συνάρτηση Λ έχουμε τη συνάρτηση **Θ**. **[21]**

$$(\sigma', g', \mathbf{s}, \mathbf{o}) \equiv \Theta(\sigma, s, o, r, g, p, v, \mathbf{d})$$

(Εικόνα από **[21]**)

Αν θεωρήσουμε **[21]** ότι η κατάσταση πριν εφαρμοστεί το transaction μας είναι η **σ** τότε σαν **σ_1** θεωρούμε την κατάσταση αφού έχει μεταφερθεί το value στην διεύθυνση του παραλήπτη.

$$\begin{aligned} \sigma_1 &\equiv \sigma \quad \text{except:} \\ \sigma_1[r]_b &\equiv \sigma[r]_b + v \end{aligned}$$

(Εικόνα από **[21]**)

Στη συνέχεια σε περίπτωση που ο παραλήπτης δεν είναι contract account και δεν έχει κώδικα η κατάσταση δεν μεταβάλλεται.

Αντίθετα αν ο παραλήπτης είναι ένα contract account τότε εκτελείται ο κώδικας του με input το πεδίο data **d** του transaction μέσω της βοήθειας της Ξ .Η νέα πλέον κατάσταση θεωρείται η **σ**** που επιστρέφει η Ξ .Σε περίπτωση που το message call προκαλείται από ένα VM –code κατά την εκτέλεση χρησιμοποιείται μία άλλη συνιστώσα του transaction που είναι το output **o**.

Εάν δεν επαρκεί το gas του message call για να εκτελεστεί ο κώδικας η κατάσταση επανέρχεται στην προηγούμενη μορφή σ_1 πριν ξεκινήσει η εκτέλεση. Το σ** που επιστρέφει εδώ η Ξ είναι το∅.

$$\begin{aligned} \sigma' &\equiv \begin{cases} \sigma_1 & \text{if} \quad \sigma_1[r]_c = \varnothing \\ \sigma_1 & \text{if} \quad \sigma^{**} = \varnothing \\ \sigma^{**} & \text{otherwise} \end{cases} \\ (\sigma^{**}, g', \mathbf{s}, \mathbf{o}) &\equiv \Xi(\sigma_1, g, I) \\ I_a &\equiv a \\ I_o &\equiv o \\ I_p &\equiv p \\ I_{\mathbf{d}} &\equiv \mathbf{d} \\ I_s &\equiv s \\ I_v &\equiv v \end{aligned}$$

(Εικόνα από **[21]**)

Η εκτέλεση του κώδικα γίνεται με τη βοήθεια της ethereum virtual machine (EVM) η οποία είναι μία quasi –Turing complete μηχανή καταστάσεων. Με τον χαρακτηρισμό quasi εννοούμε ότι στη διαμόρφωση των καταστάσεων πέρα από τις εντολές και το input παίζει ρόλο και το gas που περιορίζει τα υπολογιστικά βήματα. **[20],[21]**

Η EVM διαθέτει τριών ειδών μνήμες: το stack (LIFO ουρά), τη memory (byte array)και το storage (key,value). Το storage σε αντίθεση με τη memory δεν είναι προσωρινό, γι αυτό για να υπάρχει κίνητρο να μην αποθηκεύονται πολλά δεδομένα σε αυτό και επιβαρύνονται οι nodes ,κοστίζει περισσότερο gas η χρησιμοποίησή του μέσω των transactions. **[20],[21]**

3.5.11 ETHEREUM MAIN CHAIN

Το Ethereum διαφοροποιείται από το bitcoin **[21]** ως προς το ποια θεωρείται η κύρια αλυσίδα του δικτύου και ακολουθεί μία μορφή του GHOST protocol. Στο bitcoin αποδεκτή γίνεται η πιο μακριά αλυσίδα η οποία απεικονίζει ουσιαστικά το περισσότερο συνολικό difficulty.

Στο ethereum στο συνολικό difficulty της αλυσίδας θα συνεισφέρουν και κάποια stale blocks που αποφασίζει να συμπεριλάβει στο block του ο miner. Συγκεκριμένα ο miner συμπεριλαμβάνει στο block του κάποια uncle blocks (ο πατέρας των οποίων είναι παππούς του) και το συνολικό difficulty της αλυσίδας ορίζεται ως το άθροισμα του συνολικού difficulty του πατέρα με το difficulty του παρόντος block και το άθροισμα των difficulties των uncle blocks .

$$B_t \equiv B'_t + B_d + \sum_{U \in B_{\mathbf{U}}} U_d$$

$$B' \equiv P(B_H)$$

(Εικόνα από **[21]**)

3.5.12 REWARDS

Για να έχει κίνητρο ο miner να συμπεριλαμβάνει uncle blocks αμείβεται **[21]** για κάθε uncle block που συμπεριλαμβάνει ,ενώ ταυτόχρονα αμείβονται και οι κάτοχοι των uncle blocks .

Πιο συγκεκριμένα ο miner που βγάζει το κάθε block πέρα από την αμοιβή που προβλέπεται για το παρόν block $\mathbf{R_b}$,κερδίζει και το 1/8 της αμοιβής από κάθε uncle που συμπεριλαμβάνει .

Ο κάθε κάτοχος των uncle blocks αμείβεται με τα ¾ της αμοιβής του block.

Τέλος η συνάρτηση που καθορίζει την τελική κατάσταση, world state, που αποτυπώνεται στο block, η οποία καθορίζει και τις αμοιβές, είναι **η Ω [21]** .

$$
\begin{aligned}
\Omega(B,\sigma) &\equiv \sigma' : \sigma' = \sigma \quad \text{except:} \\
\sigma'[B_{Hb}]_b &= \sigma[B_{Hb}]_b + (1 + \frac{|B_{\mathbf{U}}|}{8})R_b \\
\forall_{U \in B_{\mathbf{U}}} \quad \sigma'[U_b]_b &= \sigma[U_b]_b + \frac{3}{4}R_b
\end{aligned}
$$

(Εικόνα από **[21]**)

ΠΗΓΕΣ

[1] Karl Sigman **,** "http://www.columbia.edu/~ks20/stochastic-I/stochasticIGRP.pdf", 2009

[2] W. Feller, "An introduction to probability theory and its applications," 1957.

[3]Satoshi Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System",2008

[4] Block, " https://en.bitcoin.it/wiki/Blocks "

[5] Developer Guide – Bitcoin, " https://bitcoin.org/en/developer-guide "

[6] Mastering Bitcoin," http://chimera.labs.oreilly.com/books/1234000001802/ch06.html#_peer_ "

[7] Script Bitcoin , " https://en.bitcoin.it/wiki/Script "

[8] Vitalik Buterin , " Selfish Mining: A 25% Attack Against the Bitcoin Network, http://bitcoinmagazine.com/7953/selfish-mining-a-25-attack-against-the-bitcoin-network/ ",2013

[9] Ittay Eyal, Emin Gun Sirer , "Majority is not Enough: Bitcoin Mining is Vulnerable",2013

[10] Transaction – Bitcoin, **"**https://en.bitcoin.it/wiki/Transaction**"**

[11] Block hashing algorithm – Bitcoin, **"** https://en.bitcoin.it/wiki/Block_hashing_algorithm **"**

[12] Contracts – Bitcoin, **"** https://en.bitcoin.it/wiki/Contracts **"**

[13] Merkle tree, **"** http://en.wikipedia.org/wiki/Merkle_tree **"**

[14] C code for Radix Tree (or Trie) | Programming Geeks - Coding Logic, " http://programminggeeks.com/c-code-for-radix-tree-or-trie/ "

[15] Ethereum Development Tutorial · ethereum/wiki Wiki · GitHub , " https://github.com/ethereum/wiki/wiki/Ethereum-Development-Tutorial "

[16] Patricia Tree · ethereum/wiki Wiki · GitHub, " https://github.com/ethereum/wiki/wiki/Patricia-Tree "

[17] RLP · ethereum/wiki Wiki · GitHub , " https://github.com/ethereum/wiki/wiki/RLP "

[18] Understanding the ethereum trie | Easy There Entropy, " https://easythereentropy.wordpress.com/2014/06/04/understanding-the-ethereum-trie/ "

[19] Merkle Patricia Tree Specification, " http://vitalik.ca/ethereum/patricia.html "

[20] Vitalik Buterin, "A Next-Generation Smart Contract and Decentralized Application Platform",2013

[21] GAVIN WOOD, "ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER",2014

[22] Ian Miers, Christina Garman, Matthew Green, Aviel D. Rubin, , " Zerocoin: Anonymous Distributed E-Cash from Bitcoin",2013

[23] J. Camenisch and A. Lysyanskaya, "Dynamic accumulators and application to efficient revocation of anonymous credentials,"
in CRYPTO '02, 2002, pp. 61–76.

[24] J. Benaloh and M. de Mare, "One-way accumulators: a decentralized alternative to digital signatures," in EUROCRYPT
'93, vol. 765 of LNCS, 1994, pp. 274–285.

[25] N. Bari´c and B. Pfitzmann, "Collision-free accumulators and fail-stop signature schemes without trees," in EUROCRYPT
'97, vol. 1233 of LNCS, 1997, pp. 480–494.

[26] proof of knowledge , " http://en.wikipedia.org/wiki/Proof_of_knowledge "

[27] Σημειώσεις Ε. Ζάχου - Α. Παγουρτζή, ΕΜΠ, 2014.

[28] Α.Κιαγιάς: Τεχνικές Σύγχρονης Κρυπτογραφίας

[29] J. Camenisch and M. Stadler, "Efficient group signature schemes for large groups," in CRYPTO '97, vol. 1296 of
LNCS, 1997, pp. 410–424.