Εθνικό Καποδιστριακό Πανεπιστήμιο Αθηνών

Τμήμα Μαθηματικών

Μεταπτυχιακό Πρόγραμμα στη Λογική και Θεωρία Αλγορίθμων και Υπολογισμού

μΠλ∀

Κεντρικά ελεγχόμενα πρωτόκολλα και Ανώνυμα αποκεντρωμένα συστήματα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Νικόλαος Λάμπρου

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 αρχικά θα μελετήσουμε το κεντροποιημένο πρωτόκολλο της κενής υπογραφής **[23]**, ένα καινούργιο είδος υπογραφής που δημοσιεύτηκε από τους Christian Hanser και Daniel Slamanig,το 2013,καθως και την ασφάλεια που μας παρέχει .Η ύπαρξη κεντροποίησης στο παραπάνω πρωτόκολλο όπως θα δούμε παίζει σημαντικό ρόλο καθώς η παράβλεψη της θα μπορούσε να οδηγήσει σε εύρεση καταπακτής για έναν από τους συμμετέχοντες. Στην συνέχεια θα αναφερθούμε στο μη κεντροποιημένο πρωτόκολλο του Bitcoin **[3]** που εισήχθη το 2008 από τον Satoshi Nakamoto καθώς και στην ασφάλεια που μας παρέχει **[1],[2]**,και θα αναφέρουμε μερικά από τα έξυπνα συμβόλαια **[12]** που έχουν χτιστεί πάνω σε αυτό ,εκμεταλλευόμενα τον αποκεντρωτικό του χαρακτήρα .Τέλος θα αναφερθούμε στο νέο νόμισμα ,το Ethereum**[21],[22]** που εισήχθη το 2013-14,από τους Vitalik Buterin, Gavin Wood (και κατά την συγγραφή της διπλωματικής βρισκόταν σε συνεχή εξέλιξη),ποιά πλεονεκτήματα θέλει να πετύχει σε σχέση με το Bitcoin ,καθώς και τον φορμαλιστικό τρόπο λειτουργιάς του.

ABSTRACT

In the present thesis, firstly we study the centralized protocol of the blank signature **[23]**, a new type of signature that was published by Christian Hanser and Daniel Slamanig , in 2013,as well as the security of this scheme. The existence of centralization in the above protocol, as we will see, is a crucial part because if we ignore it, one of the participants can find a trapdoor to the protocol. **Consequently we refer to the decentralized protocol of Bitcoin [3] that was published by** Satoshi Nakamoto **,in 2008 ,as well as the security with which it provide us [1],[2] , and we also refer to some smart contracts [12] that has been built on top of this ,which use its decentralized character. Lastly, we refer to the new decentralized system, the Ethereum[21],[22], that was introduced by** Vitalik Buterin, Gavin Wood **,in 2013-2014 (during this thesis ,its development has been growing),which advantages it wants to achieve in comparison with the Bitcoin as well as the formal way of its function .**

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω τους καθηγητές του μΠλΘ για το επιστημονικό υπόβαθρο που μου παρείχαν ,και ιδιαίτερα την τριμελή επιτροπή Ευστάθιο Ζάχο ,Άγγελο Κιαγιά και Αριστείδη Παγουρτζή που με έφεραν σε επαφή με τον κλάδο της κρυπτογραφίας και με βοήθησαν να αναπτύξω την ερευνητική μου σκέψη. Επίσης ευχαριστώ τον επιβλέποντα Αριστείδη Παγουρτζή για τις συμβουλές και την καλή συνεργασία καθ'όλη την διάρκεια της διπλωματικής.

Τέλος, ευχαριστώ την Κατερίνα και την οικογένεια μου για τη στήριξη που μου παρείχαν όλα τα χρόνια των σπουδών μου.

Περιεχόμενα

1 ΚΕΝΤΡΟΠΟΙΗΜΕΝΑ ΠΡΩΤΟΚΟΛΛΑ

Σε αυτή την ενότητα θα αναφερθούμε στην έννοια κεντροποιημένο πρωτόκολλο. Όπως θα γίνει αντιληπτό και στις άλλες ενότητες(bitcoin,ethereum) στόχος είναι να «πετάξουμε» το trusted authority από το πρωτόκολλο ,ο λόγος είναι ότι στην κρυπτογραφία θέλουμε να κρατήσουμε την εμπιστοσύνη όσο πιο χαμηλά γίνεται ,και οι παίκτες να είναι βέβαιοι ότι είναι ασφαλείς(κάτω από προϋποθέσεις) χωρίς να υπάρχει εμπιστοσύνη.

Παρόλα αυτά, η έμπιστη αρχή μας βοηθά αρχικά να σχεδιάσουμε πολύ απλά ,ευέλικτα και ασφαλή πρωτόκολλα κάτω από ελάχιστες υποθέσεις ,και στο μέλλον μπορεί να γίνει update ενός τέτοιου πρωτοκόλλου εξαφανίζοντας την έμπιστη αρχή , αυξάνοντας τις υποθέσεις .Αυτό το trade off είναι συνήθως αναγκαίο, αρκεί βέβαια να κάνει κανείς μικρές παραχωρήσεις για μεγάλα οφέλη.

Στην συνέχεια θα περιγραφεί ένα κεντροποιημένο πρωτόκολλο ,οι κενές υπογραφές (blank signatures),καθώς και η ασφάλεια που παρέχει.

1.1 BLANK SIGNATURES

Στις κενές υπογραφές **[23]** το πρόβλημα είναι ότι έχουμε έναν χρήστη **O** που θα τον λέμε originator,που θέλει να δώσει δικαιώματα υπογραφής σε έναν χρήστη **P** που θα τον λέμε proxy.Παράλληλα απαιτείται τα δικαιώματα αυτά να είναι περιορισμένα ,δηλαδή να μπορεί ο **P** να υπογράφει συγκεκριμένα στιγμιότυπα όπου θα τα έχει προκαθορίσει ο **O**. Επίσης αυτή η ανταλλαγή μηνυμάτων δεν θα πρέπει να είναι public verifiable(δεν το επιθυμεί το πρωτόκολλο).Στην συνέχεια ο proxy αφού κάνει την επιλογή του να στέλνει το μήνυμα που επέλεξε(το κάνει public),έτσι ώστε ο καθένας να μπορεί να το ελέγξει χωρείς να αποκαλύπτονται βέβαια οι άλλες επιλογές που απέρριψε ο **P** για τις οποίες είχε δικαίωμα υπογραφής.

Πριν συνεχίσουμε στην ακριβή διατύπωση του πρωτοκόλλου ,καθώς και την ασφάλεια που θέλουμε να έχει (και θα αποδείξουμε),θα αναφερθούμε σε κάποιες προαπαιτούμενες έννοιες,subprotocols όπου θα χρησιμοποιηθούν στη συνέχεια.

1.1.1 PRELIMINARIES

Elliptic curves

Μια ελλειπτική καμπύλη περιορισμένη πάνω σε ένα πεπερασμένο σώμα F_q (π.χ Z_q^*) περιγράφεται από την παρακάτω εξίσωση:

$$E : Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6,$$

(εικόνα από **[23]**)

με συντελεστές $\alpha_1,..,\alpha_6 \in F_q$.Το σύνολο $E(F_q)$ αποτελείται από τα σημεία $(x,y) \in F_q^2$ τα οποία ικανοποιούν την παραπάνω εξίσωση.

Το $E(F_q)$ αποτελεί ομάδα με ουδέτερο στοιχείο το συμβατικό σημείο στο άπειρο(αβελιανή),όπως και στο bitcoin.

Επίσης αν G κυκλική ομάδα με ord(G)=κ, και p πρώτος διαιρέτης του κ, τότε υπάρχει υποομάδα της G με τάξη p,την οποία συμβολίζουμε με G[p].

Ορισμός 1 (Bilinear Map)

Έστω G_1,G_2,G_T κυκλικές ομάδες τάξης p,με τις δυο πρώτες να είναι προσθετικές ενώ η τρίτη πολλαπλασιαστική .Λέμε ότι η απεικόνιση e: $G_1 x\ G_2 \rightarrow G_T$ είναι bilinear map ή pairing αν τα παρακάτω ισχύουν:

Bilinearity: For all $P_1, P_2 \in G_1$ and $P'_1, P'_2 \in G_2$ we have:
- $e(P_1 + P_2, P') = e(P_1, P') \cdot e(P_2, P')$ for all $P' \in G_2$,
- $e(P, P'_1 + P'_2) = e(P, P'_1) \cdot e(P, P'_2)$ for all $P \in G_1$.

Non-degeneracy: If P is a generator of G_1 and P' a generator of G_2, then $e(P, P')$ is a generator of G_T, i.e., $e(P, P') \neq 1_{G_T}$.

Efficiently computable: e can be computed efficiently.

(εικόνα από **[23]**)

Στην περίπτωση που $G_1=G_2$ λέμε ότι το pairing είναι συμμετρικό ,αλλιώς ασύμμετρο.

Ορισμός 2 (Discrete Logarithm Problem (DLP))

Έστω p πρώτος μεγέθους κ (bitlength),και G ομάδα με ord(G)=p και $\alpha \in_R Z_p^*$.Τότε για κάθε ppt(probabilistic polynomial time) αντίπαλο A,ισχύει:

$$\Pr\left(\mathcal{A}(P, \alpha P) = \alpha\right) \leq \epsilon(\kappa).$$

(εικόνα από **[23]**)

όπου ε(κ),αμελητέα συνάρτηση ως προς κ.

Ορισμός 3(t-Strong Diffie Hellman Assumption (t-SDH))

Έστω p πρώτος μεγέθους κ (bitlength),και G ομάδα με ord(G)=p και $\alpha \in_R Z_p^*$ και $(P, \alpha P, .., \alpha^t P) \in G^{t+1}$ για t>0.Τότε για κάθε ppt αντίπαλο A,ισχύει:

$$\Pr\left(\mathcal{A}(P, \alpha P, \alpha^2 P, \ldots, \alpha^t P) = \left(c, \frac{1}{\alpha + c} P\right)\right) \leq \epsilon(\kappa)$$

(εικόνα από **[23]**)

για κάθε $c \in Z_p \backslash \{\alpha\}$.

Digital Signatures

Θα αναφερθούμε συνοπτικά από τι αποτελείται ένα πρωτόκολλο ψηφιακής υπογραφής.

Ορισμός 4(Digital Signature Scheme)

Ένα πρωτόκολλο ψηφιακής υπογραφής είναι ένα διάνυσμα (DKeyGen,Dsign,Dverify) που αποτελείται από πολ/κου χρόνου αλγορίθμους με:

DKeyGen(κ):Είναι ο αλγόριθμος που δέχεται σαν παράμετρο ασφάλειας το κ, και το output είναι το dsk(signing key) και το dpk(verification key).

Dsign(M,dsk):Μας δίνει την υπογραφή σ, του plaintext M με κλειδιά το dsk.

Dverify(σ,M,dpk):Ελέγχει αν η υπογραφή είναι όντως μια υπογραφή πάνω στο M με το dpk,αν ναι επιστρέφει true αλλιώς false.

Polynomial Commitments

Το παρακάτω σχήμα δέσμευσης πολυωνύμων που θα χρησιμοποιηθεί **[24]** έχει αποδειχτεί ότι φέρει τις ιδιότητες του unconditional hiding(δηλαδή ανεξαρτήτως υπολογιστικής ισχύος του αντιπάλου η πιθανότητα να βρει το πολυώνυμο που έχουμε κάνει commit είναι ίση με το να μαντέψει ένα τυχαίο στον χώρο που βρισκόμαστε),και computational binding (δηλαδή ένας πολυωνυμικός αντίπαλος δεν μπορεί να αλλάξει την δέσμευση του).

Το πρωτόκολλο περιγράφεται παρακάτω:

(εικόνα από **[23]**)

Setup(κ, t): Pick two groups G, G_T of the same prime order p (with p being a prime of bitlength κ) having a symmetric pairing $e : G \times G \to G_T$ such that the t-SDH assumption holds. Choose a generators $P \in G$ and $\alpha \in_R \mathbb{Z}_p^*$ and output $\mathsf{ppk} = (G, G_T, p, e, P, \alpha P, \ldots, \alpha^t P)$ as well as $\mathsf{psk} = \alpha$.

Commit$(\mathsf{ppk}, f(X))$: Given $f(X) \in \mathbb{Z}_p[X]$ with $\deg(f) \leq t$ and compute the commitment $\mathcal{C} = f(\alpha)P \in G$ and output $\mathcal{C}$.

Open$(\mathsf{ppk}, \mathcal{C}, f(X))$: Output $f(X)$.

Verify$(\mathsf{ppk}, \mathcal{C}, f(X))$: Verify whether

$$\mathcal{C} = f(\alpha)P$$

holds and output `true` on success and `false` otherwise.

οπου $Z_p[X]$ ο χώρος των πολυωνύμων περιορισμένα(με συντελεστές) στο Z_p.

Παρατηρούμε ότι το f(α)P μπορεί να υπολογιστεί χωρίς την γνώσει του α(παρά μόνο από το στιγμιότυπο t-SDH που μας δύνεται) ως $\sum_{i=0}^{\deg(f)} f^i$ (α^iP).

Έχει αποδειχθεί **[24]** ότι η συνθήκη του binding δεν σπάει αν το $t<\sqrt{2^k}$,γεγονός που στην περίπτωση μας ισχύει καθώς ο αλγόριθμος κατασκευής πολυωνύμων σύμφωνα με την παράμετρο ασφαλείας κ, είναι πολυωνυμικού χρόνου .Επίσης το α πρέπει να παραμείνει άγνωστο λόγω του ότι θα μπορούσε κάποιος με την γνώση του να άλλαζε την δέσμευση του. Έτσι όπως αποδεικνύεται το παραπάνω σχήμα έχει correctness,polynomial binding,unconditional hiding κάτω από την υπόθεση t-SDH στην ομάδα G.

1.1.2 Template and Message Representation

Στην ακόλουθη ενότητα θα οριστεί η έννοια του template,η συγκεκριμενοποίηση του καθώς και ο τρόπος κωδικοποίησης και των δυο.

Ορισμός 5(Message Template)

Ένα template T αποτελείται από μη κενά σύνολα $T_i=\{M_{i1},M_{i2},..,M_{il}\}$ όπου M_{ij} ακολουθία από bits,και έναν αριθμό ταυτότητας που θα συμβολίζεται με id_T.Αν το T αποτελείται από $n=\#T_i$, τότε θα λέμε ότι το μήκος του T είναι n (η αλλιώς μπορεί να φτιάξει λέξεις/προτάσεις μήκους n).Τέλος το μέγεθος ενός template T συμβολίζεται με $|T|=\sum_{i=1}^{n}|T_i|$.

Ορισμός 6(Template Instantation)

Το μήνυμα *M* είναι μια συγκεκριμενοποίηση του template $T=(T)_{i=1}^{n}$,αν για κάθε $1\leq i\leq n$ διαλέξουμε ακριβώς ένα στοιχείο $M_i\in T_i$,και ορίσουμε ως $M=(M)_{i=1}^{n}$.

Ένα μήνυμα *M* είναι έγκυρο αν είναι προερχόμενο από κάποιο template T,το οποίο συμβολίζεται με $M\leq T$ και αντιπροσωπεύει τις διαθέσιμες επιλογές που έχει ορίσει ο originator όπως θα διατυπωθεί και παρακάτω.

Το συμπλήρωμα μιας συγκεκριμενοποίησης του template T,M, συμβολίζεται με $\overline{M}$ και είναι ίσο το T αφαιρώντας τις επιλογές του M, δηλαδή $\overline{M}=(Ti\backslash\{M_i\})_{i=1}^{n}$.

Ορισμός 7(Template Encoding)

Έστω ένα template T=$(T)_{i=1}^{n}$, και H:{0,1} → Z_p μια συνάρτηση αποτυπώματος (hash function).Η συνάρτηση κωδικοποίησης t: $T \rightarrow Z_p[X]$,ορίζεται ως εξής:

$$\mathcal{T} \mapsto \prod_{i=1}^{n} \prod_{M \in T_i} (X - H(M \| id_T \| i)).$$

(εικόνα από **[23]**)

Η τιμή t(T) ονομάζεται template encoding polynomial βαθμού όσο το μέγεθος του T(όπως αυτό έχει οριστεί παραπάνω).

Ορισμός 8(Message Encoding)

Με παρόμοιο τρόπο ορίζεται η συνάρτηση κωδικοποίησης μηνυμάτων (με διαφορά ότι το π.ο είναι το σύνολο όλων των συγκεκριμενοποιήσεων ενός template T),

t: $M_T \rightarrow Z_p[X]$,και ορίζεται ως:

$$\mathcal{M} \mapsto \prod_{i=1}^{n} (X - H(M_i \| id_T \| i)),$$

(εικόνα από **[23]**)

Επίσης με παρόμοιο τρόπο ορίζεται και το συμπλήρωμα ενός μηνύματος M:

$$\mathcal{M} \mapsto \prod_{i=1}^{n} \prod_{M \in (T_i \setminus \{M_i\})} (X - H(M \| id_T \| i)).$$

(εικόνα από **[23]**)

Παρατηρούμε ότι το πολυώνυμο κάθε συγκεκριμενοποίησης ενός template T,διαιρεί το πολυώνυμο του T , και το πηλίκο είναι το πολυώνυμο του συμπληρωματικού μηνύματος M, δηλαδή $\overline{m}_T(M)=(\frac{t_T}{m_M}) \in Z_p[X]$.

1.1.3 BLANK SIGNATURE SCHEME

Σε αυτή την ενότητα θα περιγραφεί το πρωτόκολλο της κενής υπογραφής **[23]**. Ένα πρωτόκολλο κενής υπογραφής περιγράφεται από τους παρακάτω αποδοτικούς αλγορίθμους:

KeyGen(κ,t):Το κ∈N είναι η παράμετρος για την δημιουργία σταθερών (παράμετρος ασφαλείας) και το t υποδηλώνει το μέγιστο μέγεθος template (για να είναι το πρωτόκολλο δέσμευσης ασφαλές θα πρέπει $t<\sqrt{2^{\kappa}}$).Δημιουργούνται οι δημόσιες παράμετροι pp(ουσιαστικά είναι ένα στιγμιότυπο του t-SDH,μαζί με μια ομάδα G,μια συνάρτηση ταιριάσματος e,και μια συνάρτηση αποτυπώματος στην G) και τις επιστρέφει.

Sign(T,pp,dsk_o,dpk_p):Ο αλγόριθμος αφού δεχτεί σαν είσοδο το template T,τις δημόσιες παραμέτρους pp,το προσωπικό κλειδί του originator dsk_o,το δημόσιο κλειδί του proxy dpk_p,επιστρέφει την υπογραφή του T(όχι ακριβώς στο T αλλά στην δέσμευση του κωδικοποιημένου πωλ/μου του όπως θα δούμε μετά),σ_T, μαζί με ένα κλειδί το sk_p^T που δίνεται μαζί με την υπογραφή στον proxy και πρέπει να παραμείνει κρυφό(όπως θα δούμε για το hiding condition).

$Verify_T$(T,σ_T,pp,dsk_o,dpk_p, sk_p^T):Αυτός ο αλγόριθμος με την συγκεκριμένη είσοδο εκτυπώνει true η false ανάλογα με αν η υπογραφή που δέχτηκε ταιριάζει με το T.

Inst(T,M,σ_T,pp,dsk_p, sk_p^T):Δέχεται σαν είσοδο μια συγκεκριμενοποίηση του template T,M, και επιστρέφει την υπογραφή σ_M.

$Verify_M$(M,σ_M,pp,dpk_o,dpk_p):Επιστρέφει true η false ανάλογα με το αν το M≤T,και η υπογραφή σ_M είναι έγκυρη.

Παρακάτω δίνεται αναλυτικά το πρωτόκολλο κενής υπογραφής:

KeyGen: On input (κ, t), choose an elliptic curve $E(\mathbb{F}_q)$ with a subgroup of large prime order p generated by $P \in E(\mathbb{F}_q)[p]$, such that the bitlength of p is κ. Choose a pairing $e : E(\mathbb{F}_q)[p] \times E(\mathbb{F}_q)[p] \to \mathbb{F}_{q^k}^*[p]$ and a full-domain cryptographic hash function $H : \{0,1\}^* \to \mathbb{Z}_p$ for use with the encoding functions. Pick $\alpha \in_R \mathbb{Z}_p^*$, compute $(\alpha P, \ldots, \alpha^t P)$ and output $\mathsf{pp} = (H, E(\mathbb{F}_q), e, p, P, \alpha P, \ldots, \alpha^t P)$.

Sign: Given $\mathcal{T}, \mathsf{pp}, \mathsf{dsk}_\mathsf{O}$ and dpk_P, where $\mathcal{T}$ is a template of size $|\mathcal{T}| = \ell$ and length n with $\ell > n$, this algorithm picks a unique $id_\mathcal{T} \in_R \{0,1\}^\kappa$, computes $t_\mathcal{T} = t(\mathcal{T}) \in \mathbb{Z}_p[X]$, picks a secret $\rho \in_R \mathbb{Z}_p^*$ and computes

$$\mathcal{C} = e(\rho \cdot t_\mathcal{T}(\alpha)P, P) \quad \text{and} \quad \tau = \mathsf{DSign}(id_\mathcal{T} \| \mathcal{C} \| n \| \mathsf{dpk}_\mathsf{P}, \mathsf{dsk}_\mathsf{O})$$

and returns the template signature $\sigma_\mathcal{T} = (id_\mathcal{T}, \mathcal{C}, n, \tau)$ as well as $\mathsf{sk}_\mathsf{P}^\mathcal{T} = \rho$.

Verify$_\mathcal{T}$: Given $\mathcal{T}, \sigma_\mathcal{T}, \mathsf{pp}, \mathsf{dpk}_\mathsf{O}, \mathsf{sk}_\mathsf{P}^\mathcal{T}$ and dpk_P, where $\mathcal{T}$ is a template of size $|\mathcal{T}| = \ell$ and length n with $\ell > n$, this algorithm checks whether $|\mathcal{T}| \leq t$. If not, it returns `false`. Otherwise, it computes $t_\mathcal{T} = t(\mathcal{T})$ and checks whether

$$\mathsf{DVerify}(\tau, id_\mathcal{T} \| \mathcal{C} \| n \| \mathsf{dpk}_\mathsf{P}, \mathsf{dpk}_\mathsf{O}) = \mathtt{true} \quad \wedge \quad e(\rho \cdot t_\mathcal{T}(\alpha)P, P) = \mathcal{C}.$$

If so, return `true` and `false` otherwise.

Inst: Given $\mathcal{T}, \mathcal{M}, \sigma_\mathcal{T}, \mathsf{pp}, \mathsf{sk}_\mathsf{P}^\mathcal{T}$ and dsk_P, where $\mathcal{T}$ is a template of size $|\mathcal{T}| = \ell$ and length n with $\ell > n$, this algorithm computes $m_{\overline{\mathcal{M}}} = \overline{m}_\mathcal{T}(\mathcal{M}) \in \mathbb{Z}_p[X]$. Then, it computes

$$\mathcal{C}_{\overline{\mathcal{M}}} = \rho \cdot m_{\overline{\mathcal{M}}}(\alpha)P \quad \text{and} \quad \mu = \mathsf{DSign}(\tau \| \mathcal{C}_{\overline{\mathcal{M}}} \| \mathcal{I}, \mathsf{dsk}_\mathsf{P}).$$

It returns $\sigma_\mathcal{M} = (\mu, \mathcal{C}_{\overline{\mathcal{M}}}, \mathcal{I}, \sigma_\mathcal{T})$.

Verify$_\mathcal{M}$: Given $\mathcal{M}, \sigma_\mathcal{M} = (\mu, \mathcal{C}_{\overline{\mathcal{M}}}, \mathcal{I} = (|M_i|)_{i=1}^n, \sigma_\mathcal{T}), \mathsf{pp}$, dpk_P and dpk_O this algorithm verifies whether

$$\mathsf{DVerify}(\tau, id_\mathcal{T} \| \mathcal{C} \| n \| \mathsf{dpk}_\mathsf{P}, \mathsf{dpk}_\mathsf{O}) = \mathtt{true} \ \wedge \ \mathsf{DVerify}(\mu, \tau \| \mathcal{C}_{\overline{\mathcal{M}}} \| \mathcal{I}, \mathsf{dpk}_\mathsf{P}) = \mathtt{true} \ \wedge$$

$$|\mathcal{I}| = n \quad \wedge \quad \sum_{i=1}^{n} |M_i| = |\mathcal{M}|$$

On failure return `false`, otherwise evaluate $m_\mathcal{M} = m_\mathcal{T}(\mathcal{M})$ and check whether

$$e(m_\mathcal{M}(\alpha)P, \mathcal{C}_{\overline{\mathcal{M}}}) = \mathcal{C}$$

On success return `true` and `false` otherwise.

(εικόνα από **[23]**)

Μετά από την εκτέλεση της γεννήτριας παραμετροποίησης ο originator δεσμεύεται σε ένα template T,αφού έχει επιλέξει τυχαία ένα στοιχείο του Z_p^*.Ο λόγος είναι για να κρύψει το πολυώνυμο που διάλεξε. Στην συνέχεια υπογράφει την δέσμευση του μαζί και με τις άλλες παραμέτρους και στέλνει την υπογραφή του μαζί με το sk_p^T στον proxy.

Ο proxy στην συνέχεια ελέγχει αν η δέσμευση που του δόθηκε ταιριάζει με το template T,και αν η υπογραφή είναι έγκυρη. Έπειτα ο proxy διαλέγει μια αρχικοποίηση του T,M,και δεσμεύεται στο συμπληρωματικό πολυώνυμο της με τον ίδιο ακριβώς τρόπο που δεσμεύτηκε ο originator στο T. Υπογράφει την δέσμευση του μαζί και με τις άλλες παραμέτρους ,και κάνει δημόσιο το M μαζί με την υπογραφή του (όπως και παραπάνω όταν λέμε υπογραφή εννοούμε μια σειρά δεδομένων που μέσα σε αυτά είναι και η υπογραφή καθώς και η ίδια η δέσμευση του πολυωνύμου ,το μήκος του, καθώς και το $\mathrm{id_T}$).

Τέλος ο καθένας μπορεί να ελέγξει ότι οι υπογραφές του O,P είναι έγκυρες καθώς και το γεγονός ότι όντως η συγκεκριμενοποίηση M προήρθε όντως από το T (αυτό όπως θα δούμε παρακάτω είναι το correctness που ισχύει ,αλλά θα αποδειχθεί αναλυτικά ένα ολοκληρωμένο μοντέλο ασφάλειας).

Μια παρατήρηση είναι ότι το α στο στιγμιότυπο t-SDH πρέπει πάντα να δίνεται από μια έμπιστη αρχή καθώς η γνώση του θα έσπαγε το binding condition(trapdoor**,[24]**)για τον originator (θα μπορούσε να βρει κάποιο τυχαίο πολυώνυμο με απλά το ίδιο evaluation στο α).

1.1.4 SECURITY OF THE SCHEME

Σε αυτή την ενότητα θα περιγραφεί τι ορίζεται ως ασφάλεια και κάτω από ποιές προϋποθέσεις αυτή επιτυγχάνεται **[23]**.

Οι υποθέσεις είναι ένα ασφαλές σύστημα υπογραφής (όπως αυτό περιγράφεται σε προηγούμενες ενότητες),μια ασφαλή συνάρτηση hash(collision resistant,second preimage resistant),τέλος το t-SDH είναι δύσκολο στην ομάδα $E(F_q)$.

Αρχικά θα περιγραφούν κάποιες ιδιότητες που είναι επιθυμητές να έχει το πρωτόκολλο της κενής υπογραφής ,καθώς και οι αποδείξεις τους:

1.1.4.1 Correctness

Το πρωτόκολλο πρέπει να έχει signature correctness(δηλαδή αν ο originator ακολουθήσει την διαδικασία όπως προβλέπεται ο proxy στο $Verify_T$ να επιστρέψει true),instantiation correctness(όπως και ο originator έτσι και ο proxy αν συγκεκριμενοποιήσει το μήνυμα του μέσω της Inst τότε οποιοσδήποτε τρέξει τον αλγόριθμο $Verify_M$ να επιστρέψει true),και τέλος signature soundness(να μην μπορεί ο originator να αλλάξει το template για το οποίο δεσμεύτηκε και υπέγραψε).

Πιο αναλυτικά:

signature correctness:Για οποιοδήποτε ζευγάρι κλειδιών $(dsk_o,dpk_o)\in DKeyGen(\kappa)$ και $(dsk_p,dpk_p)\in DKeyGen(\kappa)$,οποιαδήποτε παράμετρος $pp\in DKeyGen(\kappa,t)$,οποιαδήποτε template T με υπογραφή $\sigma_T= Sign(T,pp,dsk_o,dpk_p)$ που έχει υπολογιστεί όπως ορίζει το σύστημα υπογραφής, απαιτείται ο έλεγχος

(εικόνα από **[23]**)

$$\mathsf{Verify}_{\mathcal{T}}(\mathcal{T}, \sigma_{\mathcal{T}}, \mathsf{pp}, \mathsf{dpk}_{\mathsf{O}}, \mathsf{sk}_{\mathsf{P}}^{\mathcal{T}}, \mathsf{dpk}_{\mathsf{P}}) = \mathtt{true}$$

να ισχύει με πιθανότητα 1.

Η απόδειξη είναι τετριμμένη.

instantiation correctness: Για οποιαδήποτε ζευγάρι κλειδιών $(dsk_o,dpk_o)\in DKeyGen(\kappa)$ και $(dsk_p,dpk_p)\in DKeyGen(\kappa)$,οποιαδήποτε παράμετρος $pp\in DKeyGen(\kappa,t)$,οποιοδήποτε template T με υπογραφή σ_T= Sign(T,pp,dsk_o,dpk_p) που έχει υπολογιστεί όπως ορίζει το σύστημα υπογραφής ,και sk_p^T τυχαιο τετοιο ώστε:

$$\mathsf{Verify}_{\mathcal{T}}(\mathcal{T}, \sigma_{\mathcal{T}}, \mathsf{pp}, \mathsf{dpk_O}, \mathsf{sk_P^{\mathcal{T}}}, \mathsf{dpk_P}) = \mathtt{true},$$

(εικόνα από **[23]**)

και για οποιαδήποτε συγκεκριμενοποίηση του T,M,η υπογραφή

$$\sigma_{\mathcal{M}} = \mathsf{Inst}(\mathcal{T}, \mathcal{M}, \sigma_{\mathcal{T}}, \mathsf{pp}, \mathsf{sk_P^{\mathcal{T}}}, \mathsf{dsk_P}),$$

(εικόνα από **[23]**)

να έχει υπολογιστεί όπως ορίζει το πρωτόκολλο υπογραφής ,απαιτείται ο έλεγχος

$$\mathsf{Verify}_{\mathcal{M}}(\mathcal{M}, \sigma_{\mathcal{M}}, \mathsf{pp}, \mathsf{dpk_P}, \mathsf{dpk_O}) = \mathtt{true}$$

(εικόνα από **[23]**)

να ισχύει με πιθανότητα 1.

Η απόδειξη είναι τετριμμένη.

signature soundness: : Για οποιοδήποτε ζευγάρι κλειδιών $(dsk_o,dpk_o)\in DKeyGen(\kappa)$ και $(dsk_p,dpk_p)\in DKeyGen(\kappa)$,οποιαδήποτε παράμετρος $pp\in DKeyGen(\kappa,t)$,οποιοδήποτε template T με υπογραφή σ_T= Sign(T,pp,dsk_o,dpk_p) που έχει υπολογιστεί όπως ορίζει το σύστημα υπογραφής ,απαιτείται ότι για κάθε $(sk_p^{T^*},T^*)\neq(sk_p^T,T)$ η πιθανότητα ο έλεγχος

$$\mathsf{Verify}_{\mathcal{T}}(\mathcal{T}^*, \sigma_{\mathcal{T}}, \mathsf{pp}, \mathsf{dpk_O}, \mathsf{sk_P^{\mathcal{T}^*}}, \mathsf{dpk_P}) = \mathtt{true}$$

(εικόνα από **[23]**)

να ισχύει είναι αμελητέα ως προς την παράμετρο κ.

Πρόταση: Το σχήμα BDS έχει την ιδιότητα του correctness**[23]**.

Απόδειξη

Λόγω ότι η υπογραφή σ_T είναι fix, το πρώτο μέρος του $Verify_T$ είναι πάντα αληθής καθώς δεν έχει σχέση με το T^*. Άρα για να πραγματοποιηθεί η επίθεση ο επιτιθέμενος Α, θα πρέπει να βρει ένα $(sk_p^{T^*},T^*)\neq(sk_p^T,T)$ τ.ω $C^*=C$.

Αυτό μπορεί να γίνει αν καταφέρει κατά την διάρκεια κωδικοποίησης να παράγει το ίδιο πολυώνυμο με αυτό του Τ. Κάτι τέτοιο θα σήμαινε ότι θα είχε βρει δεύτερη εικόνα για n hash evaluations(αν υποθέσουμε ότι n είναι το μέγεθος του Τ) ,κάτι που γίνεται μόνο με αμελητέα πιθανότητα λόγω ότι η hash από υπόθεση είναι resistant second preimage.

Ένας δεύτερος τρόπος θα ήταν ο Α να σπάσει την δέσμευση C βρίσκοντας ένα τέτοιο ζευγάρι .Θα δειχθεί ότι αν καταφέρει και το επιτύχει με μη-αμελητέα πιθανότητα τότε μπορούμε να κατασκευάσουμε έναν αλγόριθμο Β τ.ω αν χρησιμοποιήσει τον Α σαν oracle, θα μπορέσει να σπάσει την υπόθεση t-SDH στην ομάδα $E(F_q)$.

Ο Β θα δουλέψει ως εξής:

Θα δέχεται ως είσοδο ένα στιγμιότυπο t-SDH,το $(P,\alpha P,..,\alpha^t P)$,στην συνέχεια με παράμετρο t ο Β θα τρέξει την KeyGen (κ,t) (για κ, $t<\sqrt{2^\kappa}$),θα δώσει τις $pp(H,E(F_q),e,p,\ P,\alpha P,..,\alpha^t P)$ στον Α μαζί με τα ζευγάρια κλειδιών (ένα του proxy,και ένα του originator) που πήρε από την KeyGen (κ,t).Αν ο Α καταφέρει να νικήσει επιστρέφοντας $(sk_p^{T^*},T^*)\neq(sk_p^T,T)$ με:

$$\mathsf{Verify}_{\mathcal{T}}(\mathcal{T}^*,\sigma_{\mathcal{T}},\mathsf{pp},\mathsf{dpk}_{\mathsf{O}},\mathsf{sk}_{\mathsf{P}}^{\mathcal{T}^*},\mathsf{dpk}_{\mathsf{P}})=\mathtt{true},$$

(εικόνα από **[23]**)

αυτό θα σημαίνει ότι $C^*=C$

$\Leftrightarrow e(p\cdot t(\alpha)P,P)= e(p^*\cdot t^*(\alpha)P,P) \Leftrightarrow e(P,P)^{pt(\alpha)}= e(P,P)^{p^*t^*(\alpha)} \Leftrightarrow e(P,P)^{pt(\alpha)-p^*t^*(\alpha)}=1.$

Τότε το α θα ήταν ρίζα του πολυωνύμου t'(X)=pt(X)-p*t*(X), και επειδή υπάρχουν αποδοτικοί αλγόριθμοι που μπορούν και παραγοντοποιούν πολυώνυμα πάνω σε πεπερασμένα σώματα **[25]** ο Β θα υπολόγιζε το α, και στην συνέχεια επιλέγοντας ένα $c \in_R Z_p \setminus \{-\alpha\}$, θα μας επέστρεφε το $(c, \frac{1}{\alpha+c})$ που αποτελεί λύση του t-SDH στο $E(F_q)$.

Συνοψίζοντας δείχτηκε ότι αν **Α** το ενδεχόμενο ο αντίπαλος να νικήσει, **Β** το ενδεχόμενο να βρει second preimage,και **Γ** να σπάσει το binding:

Pr(A=win)=Pr(A∩B)+Pr(A∩Γ)=negl(k)+negl(k)=negl'(k).

1.1.4.2 Unforgeability

Οποιοσδήποτε αντίπαλος μαθαίνοντας μόνο τις δημόσιες πληροφορίες (δηλαδή τις pp και τα δημόσια κλειδιά του proxy και originator),να μην μπορεί να επιτεθεί στο πρωτόκολλο. Συγκεκριμένα να μην μπορεί να παράγει έγκυρη υπογραφή template η μηνύματος. Η ασφάλεια αυτή διατυπώνεται στο παρακάτω παιχνίδι:

Setup: The challenger C runs $\mathsf{KeyGen}(\kappa, t)$ to obtain pp. Furthermore, C runs $\mathsf{DKeyGen}(\kappa)$ of a secure digital signature scheme twice to generate $(\mathsf{dsk_O}, \mathsf{dpk_O})$ and $(\mathsf{dsk_P}, \mathsf{dpk_P})$. It gives the adversary A the resulting public parameters and keys $\mathsf{pp}, \mathsf{dpk_O}$ and $\mathsf{dpk_P}$ and keeps the private keys $\mathsf{dsk_O}$ and $\mathsf{dsk_P}$ to itself.

Query: The adversary A has access to a template signing oracle $\mathcal{O}_{\mathcal{T}}$ and access to a message signing oracle $\mathcal{O}_{\mathcal{M}}$. Both oracles are simulated by the challenger C.

- On receiving a template signing query $\mathcal{T}_i$, C checks whether such a query has already been issued. If so, C retrieves $(\mathcal{T}_i, \mathsf{sk_P}^{\mathcal{T}_i}, \sigma_{\mathcal{T}_i})$ and returns $\sigma_{\mathcal{T}_i}$. Otherwise, C runs $\mathsf{Sign}(\mathcal{T}_i, \mathsf{pp}, \mathsf{dsk_O}, \mathsf{dpk_P})$, returns $\sigma_{\mathcal{T}_i}$ and stores the so obtained $(\mathcal{T}_i, \mathsf{sk_P}^{\mathcal{T}_i}, \sigma_{\mathcal{T}_i})$.
- On receiving a message signing query $(\mathcal{T}_i, \mathcal{M}_{i_j})$, C checks, whether a template signing query for $\mathcal{T}_i$ has already been made and whether $\mathcal{M}_{i,j} \preceq \mathcal{T}_i$. If not, C returns $\perp$. Otherwise, C checks whether the query $(\mathcal{T}_i, \mathcal{M}_{i_j})$ has already been made. If so, C retrieves $(\mathcal{T}_i, \mathcal{M}_{i_j}, \sigma_{\mathcal{M}_{i_j}})$ and returns $\sigma_{\mathcal{M}_{i_j}}$. If not, C retrieves $(\mathcal{T}_i, \mathsf{sk_P}^{\mathcal{T}_i}, \sigma_{\mathcal{T}_i})$, runs $\mathsf{Inst}(\mathcal{T}_i, \mathcal{M}_{i_j}, \sigma_{\mathcal{T}_i}, \mathsf{pp}, \mathsf{sk_P}^{\mathcal{T}_i}, \mathsf{dsk_P})$, returns $\sigma_{\mathcal{M}_{i_j}}$ and stores the tuple $(\mathcal{T}_i, \mathcal{M}_{i_j}, \sigma_{\mathcal{M}_{i_j}})$.

All of these queries can be made adaptively.

Output: The adversary A outputs either a triple $(\mathcal{T}^*, \sigma_{\mathcal{T}^*}, \mathsf{sk_P}^{\mathcal{T}^*})$ or a pair $(\mathcal{M}^*, \sigma_{\mathcal{M}^*})$. A wins if either

T1 $\mathsf{Verify}_{\mathcal{T}}(\mathcal{T}^*, \sigma_{\mathcal{T}_i}, \mathsf{pp}, \mathsf{dpk_O}, \mathsf{sk_P}^{\mathcal{T}_i}, \mathsf{dpk_P}) = \mathtt{true}$, where $\mathcal{T}^*$ is an unqueried template and $\mathsf{sk_P}^{\mathcal{T}_i}$, and $\sigma_{\mathcal{T}_i}$ correspond to one queried template $\mathcal{T}_i$,

T2 $\mathsf{Verify}_{\mathcal{T}}(\mathcal{T}^*, \sigma_{\mathcal{T}^*}, \mathsf{pp}, \mathsf{dpk_O}, \mathsf{sk_P}^{\mathcal{T}^*}, \mathsf{dpk_P}) = \mathtt{true}$ for some unqueried template $\mathcal{T}^*$ with corresponding unqueried $\mathsf{sk_P}^{\mathcal{T}^*}$, and $\sigma_{\mathcal{T}^*}$,

M1 $\mathsf{Verify}_{\mathcal{M}}(\mathcal{M}^*, \sigma_{\mathcal{M}_{i_j}}, \mathsf{pp}, \mathsf{dpk_P}, \mathsf{dpk_O}) = \mathtt{true}$, where $\mathcal{M}^* \neq \mathcal{M}_{i_j}$ is an unqueried message and $\sigma_{\mathcal{M}_{i_j}}$ corresponds to an already queried $\mathcal{M}_{i_j} \preceq \mathcal{T}_i$, or

M2 $\mathsf{Verify}_{\mathcal{M}}(\mathcal{M}^*, \sigma_{\mathcal{M}^*}, \mathsf{pp}, \mathsf{dpk_P}, \mathsf{dpk_O}) = \mathtt{true}$ for some unqueried message $\mathcal{M}^*$ from template signature $\sigma_{\mathcal{T}_i}$ for some previously queried template $\mathcal{T}_i$, such that either

(a) $\mathcal{M} \preceq \mathcal{T}_i$, or

(b) $\mathcal{M} \not\preceq \mathcal{T}_i$.

(εικόνα από **[23]**)

Όπως φαίνεται παραπάνω ο αντίπαλος έχει στην διάθεση του δυο signing oracles τα οποία τα ελέγχει ο challenger.Μπορεί να ρωτά υπογραφές για διάφορα template T_i, και να του δίνεται η υπογραφή τους ή και υπογραφές για συγκεκριμενοποιήσεις για κάποιο από τα templates που είχε ρωτήσει.

Στο τέλος για να νικήσει θα πρέπει είτε να βρει κάποιο template που δεν το είχε ρωτήσει στο oracle το οποίο περνάει το verify με υπογραφή κάποιου template που είχε ρωτήσει στο query phase (T1 win condition),η να βρει μια τελείως νέα υπογραφή που να περνάει το verify (T2 win condition).Τώρα για τα μηνύματα για να κερδίσει αρκεί είτε να βρει ένα μήνυμα που δεν το έχει ρωτήσει στο query phase και περνάει το verify με υπογραφή ενός μηνύματος που το έχει ρωτήσει (M1),η να βρει μια νέα υπογραφή ενός μηνύματος M προερχόμενο από ένα template T που το είχε ρωτήσει στο query phase (M2a),η τέλος να βρει μια υπογραφή ενός μηνύματος M που δεν προέρχεται από κανένα template T που είχε ρωτήσει(M2b).

Ορισμός: Το BDS έχει την ιδιότητα unforgeability αν οποιοσδήποτε ppt αντίπαλος A, κερδίζει στο παραπάνω παιχνίδι με αμελητέα πιθανότητα ως προς την παράμετρο ασφαλείας.

Πρόταση: Το BDS έχει την ιδιότητα unforgeability **[23].**

<u>Απόδειξη</u>:

Για την περίπτωση (T1) ο αντίπαλος θα πρέπει να βρει T^* τ.ω $T^* \neq T_i$ για κάθε i στο query step. Λόγω του αλγορίθμου Verify_T θα πρέπει να ισχύει:

$$\mathcal{C}_i = e(\rho^* \cdot t_{\mathcal{T}\bullet}(\alpha)P, P),$$

(εικόνα από **[23]**)

για κάποιο i που ρώτησε. Πράγμα που σημαίνει ότι αν ο A κατάφερε να βρει ένα T^* τ.ω να ισχύουν ένα από τα δυο:

α)$t_{T_i}(X)=t_{T^*}(X)$

Γεγονός που θα σήμαινε ότι θα έβρισκε δεύτερες εικόνες στην συνάρτηση hash,πιο συγκεκριμένα οι εικόνες θα έπρεπε να είναι της μορφής $H(M||id_T||j)$ και αυτό να το κάνει για τιμές όσο ο βαθμός του $t_{T_i}(X)$, γεγονός που θα μπορούσε να συμβεί με αμελητέα πιθανότητα λόγω ασφαλούς hash function.

β) $t_{T_i}(X) \neq t_{T^*}(X)$

Σε αυτή την περίπτωση εργαζόμαστε όπως η προηγούμενη απόδειξη βρίσκοντας ότι το α είναι ρίζα του πολυωνύμου t'(X)=pt(X)-pt*(X),γεγονός που σημαίνει ότι θα μπορούσαμε να σπάσουμε την t-SDH υπόθεση.

Για την περίπτωση (T2) λόγω του ότι μπορεί πλέον να επιλέξει εκείνος να δεσμεύσει το δικό του πολυώνυμο ,ο μόνος τρόπος να περάσει το Verify_T είναι να βρει $(T^*,\sigma_{T^*})\neq(T_i,\sigma_i)$ για κάθε i στο query step, τ.ω

$$\mathsf{DVerify}(\tau, id_{\mathcal{T}^*} \| \mathcal{C}^* \| n^* \| \mathsf{dpk_P}, \mathsf{dpk_O}) = \mathtt{true}$$

(εικόνα από **[23]**)

Γεγονός που θα σήμαινε ότι θα έσπαγε το πρωτόκολλο υπογραφής που χρησιμοποιούμε ,που λόγω υπόθεσης γίνεται με αμελητέα πιθανότητα.

Τώρα η συνθήκη (M1) ,διακρίνεται σε δυο περιπτώσεις:

α) $m_{M_{ij}}(X)=m_{M^*}(X)$

όπως και στην προηγούμενη απόδειξη αυτό θα απαιτούσε να βρεθούν hash second preimage όσο ο βαθμός του M_{ij},και συγκεκριμένα

$$H(M_l^* \| id_{\mathcal{T}_i} \| l) = H(M_{i_{j_l}} \| id_{\mathcal{T}_i} \| l)$$

(εικόνα από **[23]**)

που λόγω υποθέσεων μπορεί να συμβεί με αμελητέα πιθανότητα.

β) $m_{M_{ij}}(X)\neq m_{M^*}(X)$

Σε αυτή την περίπτωση θα αποδειχθεί ότι αν ο αντίπαλος καταφέρει να κερδίσει με μη-αμελητέα πιθανότητα ισχύοντας το παραπάνω τότε μπορεί να κατασκευαστεί αλγόριθμος B τ.ω να σπάει την υπόθεση t-SDH.

Το setup γίνεται όπως στην προηγούμενη απόδειξη για τον B, τώρα για να κερδίσει ο A ξέρουμε ότι :

$$e(\mathcal{C}_{\mathcal{M}^*}, \mathcal{C}_{\overline{\mathcal{M}}_{i_j}}) = e(\mathcal{C}_{\mathcal{M}_{i_j}}, \mathcal{C}_{\overline{\mathcal{M}}_{i_j}})$$
$$e(\mathcal{C}_{\mathcal{M}^*} - \mathcal{C}_{\mathcal{M}_{i_j}}, \mathcal{C}_{\overline{\mathcal{M}}_{i_j}}) = 1$$

(εικόνα από **[23]**)

Το $\mathcal{C}_{\bar{M}_{i_j}} \neq 0$ με πιθανότητα 1-negl(κ),προκύπτει ότι:

$$m_{\mathcal{M}^*}(\alpha)P = m_{\mathcal{M}_{i_j}}(\alpha)P$$
$$m_{\mathcal{M}^*}(\alpha)P - m_{\mathcal{M}_{i_j}}(\alpha)P = \mathcal{O}$$

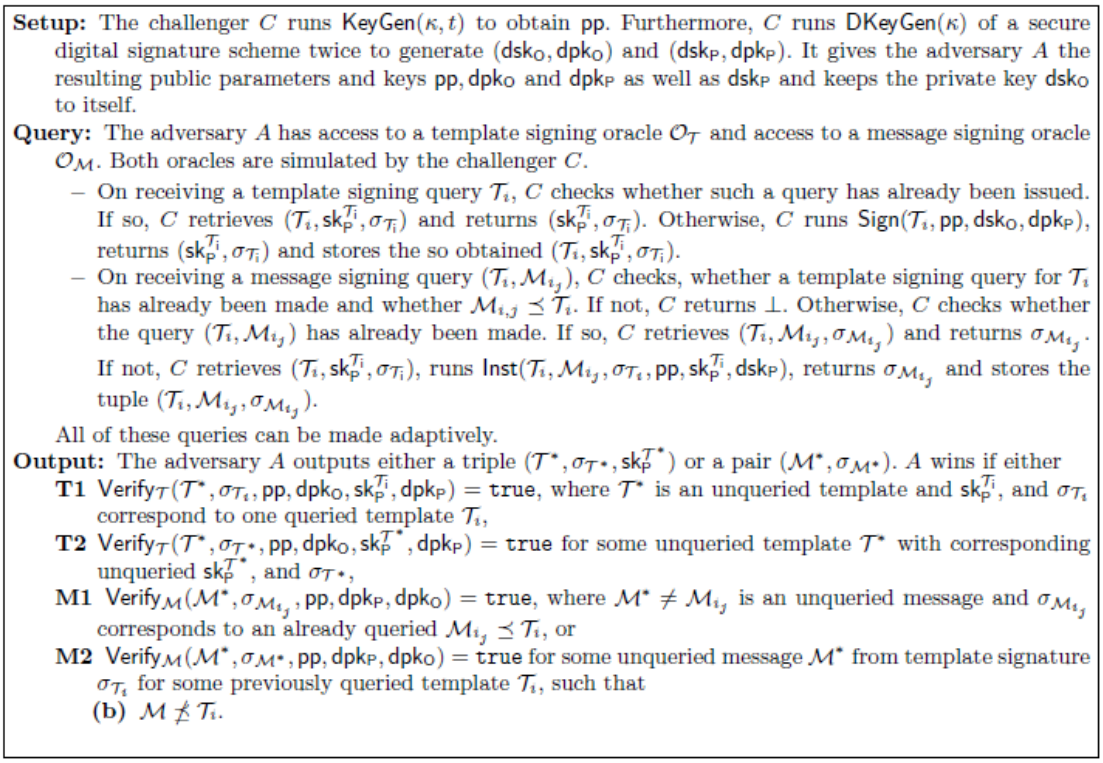
(εικόνα από **[23]**)

Συνεπώς το α είναι ρίζα του πολυωνύμου m_{M^*}(X)-$m_{M_{i_j}}$(X)∈Z_p[X]. Όπως και παραπάνω για επιλογή c∈$_R Z_p$\{-α}, ο B μπορεί να βρει μια λύση του t-SDH στο E(F_q),που από υπόθεση συμβαίνει μόνο με αμελητέα πιθανότητα.

Στις συνθήκες (M2a) και (M2b) λόγω του ότι δεν ενδιαφέρει τον αντίπαλο να χρησιμοποιήσει μια υπάρχουσα υπογραφή αλλά να φτιάξει μια καινούργια ,η δέσμευση δεν είναι πρόβλημα στο Verify,αλλά η υπογραφή που θα δώσει θα πρέπει να ανταποκρίνεται στο dpk_p,dpk_o πράγμα που σημαίνει ότι θα έσπαγε την υπόθεση για το σχήμα υπογραφής που χρησιμοποιούμε.

1.1.4.3 Immutability

Σε αυτή την περίπτωση ο αντίπαλος είναι ο proxy,και προσπαθεί να υπολογίσει υπογραφές για templates ή για μηνύματα που δεν επιθυμεί ο originator.Αυτή η έννοια είναι παρόμοια με το unforgeability μόνο που πιάνει την περίπτωση ο επιτιθέμενος είναι ο proxy,συγκεκριμένα αποτυπώνεται αυτός ο ορισμός στο παρακάτω παιχνίδι:



Setup: The challenger C runs $\mathsf{KeyGen}(\kappa, t)$ to obtain pp. Furthermore, C runs $\mathsf{DKeyGen}(\kappa)$ of a secure digital signature scheme twice to generate $(\mathsf{dsk_O}, \mathsf{dpk_O})$ and $(\mathsf{dsk_P}, \mathsf{dpk_P})$. It gives the adversary A the resulting public parameters and keys $\mathsf{pp}, \mathsf{dpk_O}$ and $\mathsf{dpk_P}$ as well as $\mathsf{dsk_P}$ and keeps the private key $\mathsf{dsk_O}$ to itself.

Query: The adversary A has access to a template signing oracle $\mathcal{O}_{\mathcal{T}}$ and access to a message signing oracle $\mathcal{O}_{\mathcal{M}}$. Both oracles are simulated by the challenger C.

- On receiving a template signing query $\mathcal{T}_i$, C checks whether such a query has already been issued. If so, C retrieves $(\mathcal{T}_i, \mathsf{sk_P}^{\mathcal{T}_i}, \sigma_{\mathcal{T}_i})$ and returns $(\mathsf{sk_P}^{\mathcal{T}_i}, \sigma_{\mathcal{T}_i})$. Otherwise, C runs $\mathsf{Sign}(\mathcal{T}_i, \mathsf{pp}, \mathsf{dsk_O}, \mathsf{dpk_P})$, returns $(\mathsf{sk_P}^{\mathcal{T}_i}, \sigma_{\mathcal{T}_i})$ and stores the so obtained $(\mathcal{T}_i, \mathsf{sk_P}^{\mathcal{T}_i}, \sigma_{\mathcal{T}_i})$.
- On receiving a message signing query $(\mathcal{T}_i, \mathcal{M}_{i_j})$, C checks, whether a template signing query for $\mathcal{T}_i$ has already been made and whether $\mathcal{M}_{i,j} \preceq \mathcal{T}_i$. If not, C returns $\perp$. Otherwise, C checks whether the query $(\mathcal{T}_i, \mathcal{M}_{i_j})$ has already been made. If so, C retrieves $(\mathcal{T}_i, \mathcal{M}_{i_j}, \sigma_{\mathcal{M}_{i_j}})$ and returns $\sigma_{\mathcal{M}_{i_j}}$. If not, C retrieves $(\mathcal{T}_i, \mathsf{sk_P}^{\mathcal{T}_i}, \sigma_{\mathcal{T}_i})$, runs $\mathsf{Inst}(\mathcal{T}_i, \mathcal{M}_{i_j}, \sigma_{\mathcal{T}_i}, \mathsf{pp}, \mathsf{sk_P}^{\mathcal{T}_i}, \mathsf{dsk_P})$, returns $\sigma_{\mathcal{M}_{i_j}}$ and stores the tuple $(\mathcal{T}_i, \mathcal{M}_{i_j}, \sigma_{\mathcal{M}_{i_j}})$.

All of these queries can be made adaptively.

Output: The adversary A outputs either a triple $(\mathcal{T}^*, \sigma_{\mathcal{T}^*}, \mathsf{sk_P}^{\mathcal{T}^*})$ or a pair $(\mathcal{M}^*, \sigma_{\mathcal{M}^*})$. A wins if either

T1 $\mathsf{Verify}_{\mathcal{T}}(\mathcal{T}^*, \sigma_{\mathcal{T}_i}, \mathsf{pp}, \mathsf{dpk_O}, \mathsf{sk_P}^{\mathcal{T}_i}, \mathsf{dpk_P}) = \mathtt{true}$, where $\mathcal{T}^*$ is an unqueried template and $\mathsf{sk_P}^{\mathcal{T}_i}$, and $\sigma_{\mathcal{T}_i}$ correspond to one queried template $\mathcal{T}_i$,

T2 $\mathsf{Verify}_{\mathcal{T}}(\mathcal{T}^*, \sigma_{\mathcal{T}^*}, \mathsf{pp}, \mathsf{dpk_O}, \mathsf{sk_P}^{\mathcal{T}^*}, \mathsf{dpk_P}) = \mathtt{true}$ for some unqueried template $\mathcal{T}^*$ with corresponding unqueried $\mathsf{sk_P}^{\mathcal{T}^*}$, and $\sigma_{\mathcal{T}^*}$,

M1 $\mathsf{Verify}_{\mathcal{M}}(\mathcal{M}^*, \sigma_{\mathcal{M}_{i_j}}, \mathsf{pp}, \mathsf{dpk_P}, \mathsf{dpk_O}) = \mathtt{true}$, where $\mathcal{M}^* \neq \mathcal{M}_{i_j}$ is an unqueried message and $\sigma_{\mathcal{M}_{i_j}}$ corresponds to an already queried $\mathcal{M}_{i_j} \preceq \mathcal{T}_i$, or

M2 $\mathsf{Verify}_{\mathcal{M}}(\mathcal{M}^*, \sigma_{\mathcal{M}^*}, \mathsf{pp}, \mathsf{dpk_P}, \mathsf{dpk_O}) = \mathtt{true}$ for some unqueried message $\mathcal{M}^*$ from template signature $\sigma_{\mathcal{T}_i}$ for some previously queried template $\mathcal{T}_i$, such that

(b) $\mathcal{M} \not\preceq \mathcal{T}_i$.

(εικόνα από **[23]**)

παρατηρούμε ότι η περίπτωση (M2a) στο συγκεκριμένο setting δεν έχει νόημα καθώς ο proxy πάντα μπορεί και υπογράφει έγκυρες αρχικοποιήσεις.

Ορισμός: Το σχήμα κενής υπογραφής έχει την ιδιότητα immutability αν για κάθε ppt αντίπαλο A, η πιθανότητα να νικήσει στο παραπάνω παιχνίδι είναι αμελητέα ως προς την παράμετρο κ .

Πρόταση: Το σχήμα κενής υπογραφής έχει την ιδιότητα immutability **[23]**.

<u>Απόδειξη:</u>

Οι αποδείξεις των περιπτώσεων (T1),(T2),(M1) είναι ανάλογες με τις προηγούμενες.

Για την περίπτωση (M2b) θα δειχτεί ότι αν ο A μπορεί να βρει ένα M* που να μην αποτελεί συγκεκριμενοποίηση κανενός template T_i για κάθε i στο query step (δηλαδή μην ισχύει ότι $M^* \leq T_i$),τότε μπορεί να κατασκευαστεί αλγόριθμος B που να σπάει την υπόθεση t-SDH στο $E(F_q)$.

Ο Β αρχικά παίρνει ένα στιγμιότυπο του προβλήματος (P,αP,...,α^tP),δημιουργεί τις δημόσιες παραμέτρους pp= (H,E(F_q),e,p, P,αP) και τα ζευγάρια κλειδιών του proxy και του originator και τα δίνει στον Α (εκτός του private key του proxy).Αν ο Α καταφέρει και νικήσει βρίσκοντας ένα M^* τ.ω το $Verify_M(M^*,\sigma_{M*},pp,dpk_o,dpk_p)$=true, θα ισχύει ότι:

$$e(\mathcal{C}_{\mathcal{M}^*},\mathcal{C}_{\overline{\mathcal{M}}^*}) = \mathcal{C},$$ (εικόνα από **[23]**)

(επίσης λόγω του ότι το M* δεν προέρχεται από το T_i θα ισχύει ότι η διαίρεση των πολυώνυμων τους θα αφήνει υπόλοιπο , δηλαδή:

$$t_{T_i} = \overline{m}_{\mathcal{M}^*} \cdot m_{\mathcal{M}^*} + \xi$$ (εικόνα από **[23]**)

με $\xi \neq 0$.)

συνεπώς το $C_{\overline{M^*}}$ θα είναι της μορφής

$$\mathcal{C}_{\overline{\mathcal{M}}^*} = \rho_i(\overline{m}_{\mathcal{M}^*}(\alpha) + \frac{\xi(\alpha)}{m_{\mathcal{M}^*}(\alpha)})P$$ (εικόνα από **[23]**)

Ο Β μπορεί να υπολογίσει το m_{M^*},και με ευκλείδεια διαίρεση με το t_{T_i} να ανακτήσει το ξ και το $\overline{m}_{M^*}$.Έτσι ο Β μπορεί να υπολογίσει το $\overline{m}_{M^*}(\alpha)P$.Στην συνέχεια προσθέτοντας τον αντίθετο του στην παραπάνω σχέση προκύπτει:

$$\rho_i^{-1}\mathcal{C}_{\overline{\mathcal{M}}^*} - \overline{m}_{\mathcal{M}^*}(\alpha)P = \frac{\xi(\alpha)}{m_{\mathcal{M}^*}(\alpha)}P.$$ (εικόνα από **[23]**)

Λόγω ότι τα templates(και κατά συνέπεια τα μηνύματα) που μπορεί να δημιουργήσει ο αντίπαλος είναι το πολύ βαθμού ένα (λόγω παραμέτρων),το $\deg(\xi) = 0$ και $\deg(\overline{m}_{M^*}) = 1$, συνεπάγεται:

$$\frac{\xi(\alpha)}{m_{\mathcal{M}^*}(\alpha)}P = \frac{\omega}{\alpha - H(M_0\|id_T\|1)}P$$ (εικόνα από **[23]**)

Τώρα ο Β μπορεί να πολλαπλασιάσει με τον αντίστροφο του ω και να πάρει την λύση $(-H(M_0||id_T||1), \frac{1}{\alpha - H(M_0||id_T||1)}P)$ του προβλήματος t-SDH στο $E(F_q)$.Παρατηρούμε ότι η πιθανότητα να συμβεί αυτό δεν είναι ακριβώς όσο η πιθανότητα να νικήσει ο Α

αλλά μειωμένη κατά κάτι αμελητέο λόγω ότι κατά την διάρκεια της απόδειξης απαιτήθηκε το $\alpha \neq H(M_0||id_T||1)$ που συμβαίνει με αμελητέα πιθανότητα.

1.1.4.4 Privacy

Σε αυτή την περίπτωση είναι επιθυμητό οποιοσδήποτε public verifier εκτός από τον originator και τον proxy,αν του δοθούν συγκεκριμενοποιήσεις ενός template T,να μην μπορεί να βρει το συμπληρωματικό πολυώνυμο του μηνύματος τους, και κατ΄επέκταση το ίδιο το template(δηλαδή τις unused options).Πιο συγκεκριμένα είναι επιθυμητό ακόμα και να έχουν φανερωθεί όλες οι πιθανές επιλογές ενός template εκτός από μια να μην μπορεί οποιοσδήποτε verifier να την βρει, αυτή η απαίτηση αποτυπώνεται στο παρακάτω παιχνίδι:

Setup: The challenger C runs $\mathsf{KeyGen}(\kappa, t)$ to obtain pp. Furthermore, C runs $\mathsf{DKeyGen}(\kappa)$ of a secure digital signature scheme twice to generate $(\mathsf{dsk_O}, \mathsf{dpk_O})$ and $(\mathsf{dsk_P}, \mathsf{dpk_P})$. It gives the adversary A the resulting public parameters and keys $\mathsf{pp}, \mathsf{dpk_O}$ and $\mathsf{dpk_P}$ and keeps the private keys $\mathsf{dsk_O}$ and $\mathsf{dsk_P}$ to itself.

Query 1: The adversary A has access to a template signing oracle $\mathcal{O}_{\mathcal{T}}$ and access to a message signing oracle $\mathcal{O}_{\mathcal{M}}$. Both oracles are simulated by the challenger C.

- On receiving a template signing query $\mathcal{T}_i$, C checks whether such a query has already been issued. If so, C retrieves $(\mathcal{T}_i, \mathsf{sk_P}^{\mathcal{T}_i}, \sigma_{\mathcal{T}_i})$ and returns $(\mathsf{sk_P}^{\mathcal{T}_i}, \sigma_{\mathcal{T}_i})$. Otherwise, C runs $\mathsf{Sign}(\mathcal{T}_i, \mathsf{pp}, \mathsf{dsk_O}, \mathsf{dpk_P})$, returns $(\mathsf{sk_P}^{\mathcal{T}_i}, \sigma_{\mathcal{T}_i})$ and stores the so obtained $(\mathcal{T}_i, \mathsf{sk_P}^{\mathcal{T}_i}, \sigma_{\mathcal{T}_i})$.
- On receiving a message signing query $(\mathcal{T}_i, \mathcal{M}_{i_j})$, C checks, whether a template signing query for $\mathcal{T}_i$ has already been made and whether $\mathcal{M}_{i,j} \preceq \mathcal{T}_i$. If not, C returns $\perp$. Otherwise, C checks whether the query $(\mathcal{T}_i, \mathcal{M}_{i_j})$ has already been made. If so, C retrieves $(\mathcal{T}_i, \mathcal{M}_{i_j}, \sigma_{\mathcal{M}_{i_j}})$ and returns $\sigma_{\mathcal{M}_{i_j}}$. If not, C retrieves $(\mathcal{T}_i, \mathsf{sk_P}^{\mathcal{T}_i}, \sigma_{\mathcal{T}_i})$, runs $\mathsf{Inst}(\mathcal{T}_i, \mathcal{M}_{i_j}, \sigma_{\mathcal{T}_i}, \mathsf{pp}, \mathsf{sk_P}^{\mathcal{T}_i}, \mathsf{dsk_P})$, returns $\sigma_{\mathcal{M}_{i_j}}$ and stores the tuple $(\mathcal{T}_i, \mathcal{M}_{i_j}, \sigma_{\mathcal{M}_{i_j}})$.

All of these queries can be made adaptively.

Challenge: At some point A signals C that he is ready to be challenged by choosing two unqueried, distinct templates $\mathcal{T}_0'$ and $\mathcal{T}_1'$ of sizes less than t, which are shaped in such a way that from both templates k equal instantiations $\mathcal{M}_1', \ldots, \mathcal{M}_k'$ can be derived, and sending them to C. Then, C signs these two templates, stores the tuples $(\mathsf{sk_P}^{\mathcal{T}_0'}, \sigma_{\mathcal{T}_0'})$, $(\mathsf{sk_P}^{\mathcal{T}_1'}, \sigma_{\mathcal{T}_1'})$ and returns the corresponding template signatures $\sigma_{\mathcal{T}_0'}$ and $\sigma_{\mathcal{T}_1'}$ in a randomly permuted order to A.

Query 2: The adversary A is allowed to issue an arbitrary number of queries as in query phase 1, excluding templates $\mathcal{T}_0'$ and $\mathcal{T}_1'$.

- Additionally, A can send instantiation queries for $\mathcal{M}_l'$ with $1 \leq l \leq k$ to an additional oracle $\mathcal{O}'_{\mathcal{M}}$, which is simulated by C. On receiving such a query $\mathcal{M}_l'$, C checks whether $\mathcal{M}_l' \preceq \mathcal{T}_0'$ and $\mathcal{M}_l' \preceq \mathcal{T}_1'$. If not, C return $\perp$. Otherwise, C checks whether such a query has already been made. If so, C retrieves $(\mathcal{M}_l', \sigma_{\mathcal{M}_{0_l}'}, \sigma_{\mathcal{M}_{1_l}'})$ and returns $\sigma_{\mathcal{M}_{0_l}'}$ and $\sigma_{\mathcal{M}_{1_l}'}$ in a randomly permuted order. If not, C retrieves $(\mathsf{sk}_P^{\mathcal{T}_b'}, \sigma_{\mathcal{T}_b'})$, runs $\mathsf{Inst}(\mathcal{T}_b', \mathcal{M}_{b_l}', \sigma_{\mathcal{T}_b'}, \mathsf{pp}, \mathsf{sk_P}^{\mathcal{T}_b'}, \mathsf{dsk_P})$ for $b = 0, 1$, stores $(\mathcal{M}_l', \sigma_{\mathcal{M}_{0_l}'}, \sigma_{\mathcal{M}_{1_l}'})$ and returns $\sigma_{\mathcal{M}_{0_l}'}$ and $\sigma_{\mathcal{M}_{1_l}'}$ in a randomly permuted order.

All of these queries can be made adaptively.

Output: The adversary A outputs $(\mathcal{T}_b, \sigma_{\mathcal{T}_{b'}})$ and wins if $b = b'$.

(εικόνα από **[23]**)

Ορισμός: Ένα πρωτόκολλο κενής υπογραφής έχει την ιδιότητα privacy αν για οποιοδήποτε ppt αντίπαλο A,η πιθανότητα να νικήσει στο παραπάνω παιχνίδι είναι κοντά στο ½,και unconditional privacy αν για οποιονδήποτε αντίπαλο A(unbound) η πιθανότητα είναι ακριβώς ½.

Πρόταση: Το πρωτόκολλο της κενής υπογραφής έχει την ιδιότητα unconditional privacy **[23].**

Απόδειξη:

Αρχικά ο αντίπαλος θα έχει ζητήσει από το oracle τις υπόγραφες σ_0'και σ_1' των T_0,T_1 (θα τις έχει λάβει με τυχαία σειρά).Παρατηρείται ότι δεν μπορεί να εκμεταλλευτεί καμία πληροφορία παρά μόνο την πληροφορία που του παρέχουν οι δεσμεύσεις C_0'και C_1' (λόγω ότι το id επιλέγεται τυχαία, οι υπογραφές έχουν το ίδιο μήκος και είναι γνωστό τι υπογράφεται και στις δυο περιπτώσεις).Στην συνέχεια ο αντίπαλος μπορεί να ρωτήσει και όλες τις αρχικοποιήσεις (χωρίς βλάβη γενικότητας) που είναι κοινές και για τα δυο templates και να δημιουργήσει τις παρακάτω λίστες:

$(\mathcal{C}'_0, \mathcal{C}'_{\mathcal{M}'_{0_1}}, \ldots, \mathcal{C}'_{\mathcal{M}'_{0_k}})$ $(\mathcal{C}'_1, \mathcal{C}'_{\mathcal{M}'_{1_1}}, \ldots, \mathcal{C}'_{\mathcal{M}'_{1_k}})$,

(εικόνα από **[23]**)

Τέλος αυτό που του μένει να κάνει είναι να διαλέξει ποια λίστα προέρχεται από πιο template.Θα δειχτεί ότι κάθε λίστα κρύβει το πολυώνυμο της χωρίς προϋποθέσεις (δηλαδή η πιθανότητα να το βρει είναι ιδία με το να διαλέξει ένα πολυώνυμο στην τύχη).

Έστω template T,με δέσμευση $C = e(p \cdot t_T(\alpha)P, P)$.Υποθετουμε ότι αυτό το template έχει s αρχικοποιήσεις, με γνωστές στον αντίπαλο τις s-1 μαζί με τις δεσμεύσεις των συμπληρωματικών πολυωνύμων τους, δηλαδή:

$\mathcal{C}_{\mathcal{M}_1}, \ldots, \mathcal{C}_{\mathcal{M}_{s-1}}$ (εικόνα από **[23]**)

Παρατηρούμε ότι ένας παράγοντας του πολυώνυμου t_T δεν είναι γνωστος, ο $(X - \gamma)$ που επίσης δεν είναι γνωστός και στα συμπληρωματικά πολυώνυμα. Λόγω ότι κάθε δέσμευση έχει πολ/αστεί και με ένα άγνωστο τυχαίο p, υπάρχουν ακριβώς p-1 έγκυρα ζευγάρια $(\gamma, p) \in (Z_p^*)^2$ που να επαληθεύουν τις παραπάνω δεσμεύσεις. Συνεπώς είτε διαλέγαμε ένα γ τυχαία στο Z_p^* είτε λόγω της παραπάνω πληροφορίας θα είχαμε την ίδια πιθανότητα επιτυχίας.

Ορισμός : Ένα πρωτόκολλο κενής υπογραφής που φέρει τις ιδιότητες correctness,unforgeability,immutability,privacy είναι ασφαλές.

Πόρισμα: Το σχήμα κενής υπογραφής που περιγράφτηκε είναι ασφαλές.

**** Σε καθένα ορισμό από τους παραπάνω οι υποθέσεις είναι αυτές όπως περιγράφτηκαν στην αρχή της ενότητας.**

2 BITCOIN

Το bitcoin αποτελεί μια κρυπτοοικονομία **[3]** (cryptocurrency) που υλοποιείται σε peer-to-peer δίκτυα. Το κεντρικό πλεονέκτημα που εμφανίζει είναι ότι δεν υπάρχει κάποια έμπιστη αρχή (Trusted authority),σε αντίθεση με την πραγματική οικονομία (την μη ψηφιακή) όπου υπάρχει (π.χ τράπεζα).Τα χρήματα υπάρχουν στο δίκτυο μόνο σαν ιστορικό αρχείο το οποίο διατηρείται μέσα στο blockchain.Μπορεί να υπάρχουν όμως πολλά τέτοια blockchains, πώς θα ξέρουμε ποιο είναι το ¨πραγματικό¨?.Η συμφωνία είναι ότι το μεγαλύτερο blockchain σε μήκος θα είναι το αποδεκτό(εξάλλου μια ιστορία στην αφήγηση της με την περισσότερη λεπτομέρεια χωρίς αντιφάσεις γίνεται πάντα και η πιο πιστευτή!).Πέρα όμως από το μήκος του blockchain υπάρχει κι άλλος λόγος που κάναμε αυτή την συμφωνία, το proof of work. Για να συμβάλει κάποιος σε αυτή την αφήγηση τις ιστορίας (το ιστορικό των συναλλαγών στην περίπτωσή μας) θα πρέπει να δουλέψει. Άρα το μεγαλύτερο σε μήκος blockchain πέρα από το μήκος του είναι πιστευτό διότι έχει ¨δαπανηθεί ¨και πολύς κόπος. Τι γίνεται όμως στην περίπτωση που κάποιος θέλει να αλλάξει την ιστορία προς συμφέρον του?.Εκεί έρχεται να απαντήσει το proof of work (όπου θα αναλύσουμε παρακάτω πώς δουλεύει).Όσο αυτή η ομάδα ¨κακών¨ δεν έχει το 51% cpu power του δικτύου κάτι τέτοιο δεν μπορεί να γίνει (τι γίνεται όμως για λιγότερο από 51%(51% attack)? για 30%,είναι ασφαλές?).Επίσης υπάρχουν αρκετά ερωτήματα όπως πως στέλνουμε σε κάποιον ένα χρηματικό ποσό ?πέρα από αυτό τι άλλα πλεονεκτήματα προσφέρει το bitcoin? πώς μπορώ να βγάλω bitcoin?,είναι εύκολο? .Όλα αυτά μαζί με τον τρόπο λειτουργίας του bitcoin θα περιγραφούν σε αυτή την ενότητα.

2.1 TRANSACTIONS

Ας πούμε ότι ο Βασίλης θέλει να στείλει στην Αλίκη κάποια ποσότητα bitcoin. Αυτό που αρκεί να κάνει είναι να συμπληρώσει την παρακάτω φόρμα **[10]:**

Transaction input

Transaction id:Το hash του transaction που ο Βασίλης θα ¨εξαργυρώσει¨, η αλλιώς το reference transaction

Index:Ποιο output θέλει να εξαργυρώσει ο Βασίλης από το παραπάνω transaction,η αναφορά γίνεται με έναν ακέραιο

scriptSig: Το μισό μέρος του Script στο οποίο ο Βασίλης πληροί της προϋποθέσεις εξαργύρωσής του (για παράδειγμα την υπογραφή του πάνω σε άλλο το transaction πέρα από το signScript)

Transaction output

Value:Το πόσα χρήματα θα στείλει (σε bitcoin η και satoshi,τα οποία αποτελούν υποδιαίρεση του bitcoin, 1BTC=10^8 satoshi)

scriptPubKey:Το άλλο μισό μέρος του Script ,στο οποίο περιγράφονται οι συνθήκες εξαργύρωσης του παρόντος transaction .

timelock:Ο χρόνος μετά τον οποίο αυτό το transaction μπορεί να μπει σε block.

Επαλήθευση

Για να επαληθεύουμε ότι ένα transaction είναι έγκυρο μπορούμε να το κάνουμε απλά και αποδοτικά χάρη στον κώδικα του που είναι γραμμένο σε μια script based language (όχι Turing complete)περνώντας τιμές στο stack (LIFO) ωσότου διαβαστεί κάποια εντολή (π.χ OP_CHECKSIG
), και τις ECDS (Elliptic curve digital signature,όπου θα αναφερθούμε στην συνέχεια).

Τρόποι πληρωμής

Γενικά υπάρχουν πολλοί διαφορετικοί τρόποι να πληρώσει κάποιος/εξαργυρώσει ένα transaction **[7],[10]**. Μερικοί από αυτούς είναι οι παρακάτω:

1) **Pay-to-PubkeyHash**

scriptPubKey: OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
scriptSig: <sig> <pubKey>

εδώ ο λαμβάνων του transaction (recipient) πρέπει να εισάγει στο scriptSig την υπογραφή του (όπως θα δούμε στην συνέχεια έχουμε την επιλογή να υπογράψουμε διαφορετικά μέρη του transaction ανάλογα με την περίσταση και τους σκοπούς που αυτό θέλουμε να εξυπηρετεί ,θα ασχοληθούμε περαιτέρω στην ενότητα συμβόλαια), καθώς και το public key της υπογραφής **[5]**. Με αυτόν τον τρόπο όχι μόνο πιστοποιεί ότι είναι ο νόμιμος κάτοχος του transaction αλλά και τον προστατεύει από το υπόλοιπο δίκτυο για πιθανά modification του transaction(για την default υπογραφή,SignAll). Παρακάτω μπορούμε να δούμε την διαδικασία επαλήθευσης:

Stack	Script	Description
Empty.	<sig> <pubKey> OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG	scriptSig and scriptPubKey are combined.
<sig> <pubKey>	OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG	Constants are added to the stack.
<sig> <pubKey> <pubKey>	OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG	Top stack item is duplicated.
<sig> <pubKey> <pubHashA>	<pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG	Top stack item is hashed.
<sig> <pubKey> <pubHashA> <pubKeyHash>	OP_EQUALVERIFY OP_CHECKSIG	Constant added.
<sig> <pubKey>	OP_CHECKSIG	Equality is checked between the top two stack items.
true	Empty.	Signature is checked for top two stack items.

(εικόνα από **[7]**)

2)Pay-to-Script-Hash

A)scriptPubKey: OP_HASH160 <scriptHash> OP_EQUAL
scriptSig: ..signatures... <serialized script>

Σε αυτή την περίπτωση ο λαμβάνων πρέπει να δείξει ότι το hash του script που εισάγει ότι ταυτίζεται με το hash στο ScriptPubKey και στην συνέχεια όπως στην παραπάνω περίπτωση να δώσει έγκυρη/ες υπογραφή/ες (παρατηρούμε ότι οι δυο περιπτώσεις μοιάζουν αν αντί για την bitcoin address έχουμε το hash ενός script,(όπως θα δούμε και στα συμβόλαια το script μπορεί να είναι μια συνθήκη π.χ αν σήμερα έβρεξε στείλε 100BTC στο τάδε address για αποζημίωση.)

B) m-of-n multi-signature transaction:
scriptSig: 0 <sig1> ... <script>
script: OP_m <pubKey1> ... OP_n OP_CHECKMULTISIG

Παρόμοια, εδώ απαιτείται να παρέχουμε μια σειρά υπογραφών (συγκεκριμένα m υπογραφές από τις n, όπου n τα αναφερόμενα bitcoin address).

3)Anyone-Can-Spend Outputs

scriptPubKey: (empty)
scriptSig: OP_TRUE

Οποιοσδήποτε μπορεί να εξαργυρώσει το transaction

4)Transaction puzzle

scriptPubKey: OP_HASH256 6fe28c0ab6f1b372c1a6a246ae63f74f931e8365e15a089c68d6190000000000 OP_EQUAL
scriptSig:

Τέλος ο λαμβάνων πρέπει να εισάγει κάποια δεδομένα τ.ω το hash αυτών να δώσει το παραπάνω αποτέλεσμα (στην προκειμένη το 6fe28c0ab6f1b372c1a6a246ae63f74f931e8365e15a089c68d6190000000000)

2.1.1 Signature Hash Types

Όπως αναφερθήκαμε παραπάνω τις περισσότερες φορές για να εξαργυρώσουμε ένα transaction χρειάζεται και η υπογραφή του αναφερόμενου public key.Κάθε φορά μπορούμε να υπογράψουμε και κάτι διαφορετικό πάνω στο transaction**[5]** (μόνο για ότι υπογράφουμε εξασφαλίζουμε ότι δεν θα τροποποιηθεί μόλις γίνει broadcast στο δίκτυο),ανάλογα με την περίσταση (συμβόλαιο).Τα είδη των υπογραφών είναι τα παρακάτω:

- **SIGHASH ALL,** η προκαθορισμένη επιλογή. Εδώ υπογράφουμε όλο το transaction,χωρίς να επιτρέπουμε κανενός είδος μελλοντική τροποποίηση
- **SIGHASH NONE,** υπογράφει μόνο τα input του transaction αφήνοντας τα output ελεύθερα για πιθανές τροποποιήσεις (δηλαδή ο οποιοσδήποτε μπορεί να βάλει το δικό του bitcoin address)

- **SIGHASH_SINGLE,** υπογράφει μόνο ένα input και το αντίστοιχο output (π.χ αν υπογράψουμε το input 2 υπογράφουμε και το output 2),διασφαλίζοντας με αυτό τον τρόπο ότι το ¨δικό μας¨ μέρος στο transaction δεν θα τροποποιηθεί.
- **SIGHASH_ALL|SIGHASH_ANYONECANPAY,** υπογράφουμε ένα input και όλα τα output ,επιτρέποντας επιπλέον να προστεθούν ή να αφαιρεθούν άλλα input.
- **SIGHASH_NONE|SIGHASH_ANYONECANPAY,** τα inputs,εκτός από ένα και τα outputs αφήνονται ελεύθερα, ενώ παράλληλα επιτρέπεται να προστεθούν ή να αφαιρεθούν άλλα inputs ή outputs.
- **SIGHASH_SINGLE|SIGHASH_ANYONECANPAY,** υπογράφεται ένα input και το αντίστοιχο output , ενώ επιτρέπεται στον καθένα να προσθέσει ή να αφαιρέσει άλλα inputs.

2.1.2 Transaction Fees And Charge

Σε ένα transaction **[5]** πέρα από τα BTC που στέλνουμε, υπάρχει και ένα πεδίο που ονομάζεται transaction fee(δεν βρίσκεται κάποιο ειδικό domain στο transaction,το φανταζόμαστε νοητά),όπου το τελευταίο πληρώνεται στον miner σαν ¨αμοιβή¨ που συμπεριέλαβε το transaction μας στο block. Υπάρχει μια default αξία για το ποσό θα είναι το fee (ανάλογα με τα kb του transaction μας),αλλά πάντα υπόκειται στον νόμο αγορά και ζήτησης (τα πολύ χαμηλά fees ενδέχεται να αργήσουν να μπουν σε block λόγο ελλείψεως κίνητρου/ενδιαφέροντας του/των miners).

Πέρα από τα BTC που στέλνουμε, τι γίνεται στην περίπτωση που θέλουμε ρέστα ?Σε αυτή την περίπτωση μαζί με τα BTC που θέλουμε να στείλουμε ,θα στείλουμε και τα ρέστα που επιθυμούμε στον ίδιο μας τον εαυτό(!). Τώρα τι γίνεται αν π.χ τα συνολικά input του transaction είναι 10BTC στέλνουμε 8 στον Β και 1 πίσω σε μας, το 1 τι γίνεται? Επειδή η διαφορά είναι θετική ό,τι δεν ζητήθηκε αυτό είναι το transaction fee που παίρνει ο miner(όπως περιγράψαμε παραπάνω).

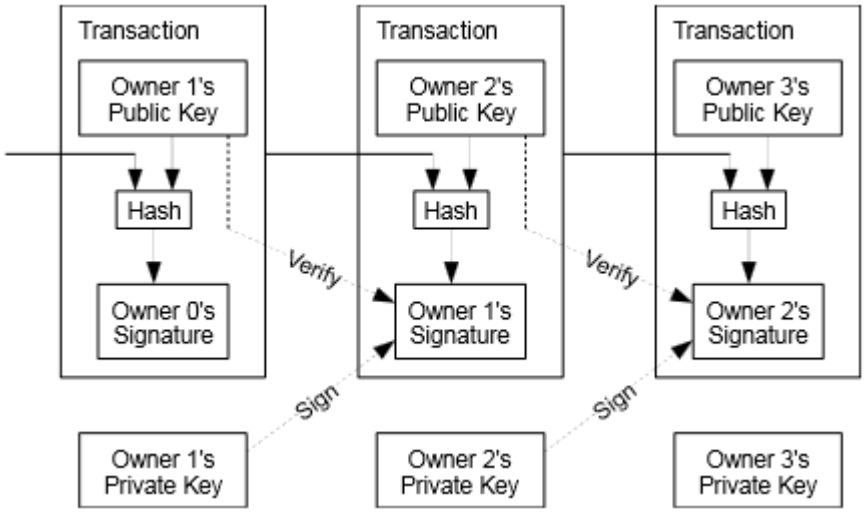
2.1.3 Transaction Malleability

Όπως αναφερθήκαμε παραπάνω στα είδη υπογραφών, σε όλες τις περιπτώσεις το κομμάτι που δεν υπογράφαμε σε ένα transaction ήταν το ScriptSign. Αυτό σαν συνέπεια έχει ότι κάποιος, όταν βρει ένα transaction στο network **[5]** , μπορεί να τροποποιήσει το data του (κάποιο κομμάτι στο ScriptSign που κατά την εκτέλεση δεν

γίνεται invalid).Το μόνο που θα αλλάξει σε αυτή την περίπτωση θα είναι το transaction id (txid).Αν π.χ ο Α συντάξει ένα tx και με αυτό στείλει κάποια BTC στον Β, από την στιγμή που μπει στο Block, δεν θα υπάρξει κάποιο πρόβλημα και ας έχει αλλάξει το txid στην πορεία. Το πρόβλημα εμφανίζεται σε περίπτωση που δυο οντότητες θέλουν να πραγματοποιήσουν μια συναλλαγή πριν ακόμα μπουν στο Block,για παράδειγμα κακόβουλος Α θέλει να στείλει στον Β κάποια BTC λόγω κάποιας ανταλλαγής υπηρεσίας (υλική/άυλη).Ο Α δημιουργεί δυο txs,ένα με το οποίο σπάει κάποια BTC σε μικρότερα ποσά (για να πληρώσει τον Β),και άλλο ένα που κάνει reference στο προηγούμενο και τα στέλνει στον Β. Τα δείχνει στον Β ότι τα έκανε broadcast,ο Β του στέλνει το αγαθό, και στην συνέχεια ο Α προσθέτει κάποια 'σκουπίδια 'στον κώδικα του πρώτου tx με αποτέλεσμα το reference του δευτέρου tx (αυτό που ουσιαστικά πληρώνεται ο Β) να μην είναι έγκυρο.

Αυτό αντιμετωπίζεται με α)Οι miner να θεωρούν ύποπτα tx με ¨παραπανίσιο¨ κώδικα, β) Ίσως το πιο αποτελεσματικό να περιμένουμε να ανακοινωθούν στο Block.

Συνοπτικά transactions **[3]** (όπως περιγράψαμε την διαδικασία υπογραφής,txid κ.α,)



(εικόνα από **[3]**)

2.1.4 Elliptic Curve Digital Signature Algorithm (ECDSA)

Όπως αναφερθήκαμε παραπάνω το είδος υπογραφής που χρησιμοποιείται είναι ECDS **[14],**πιο αναλυτικά ο αλγόριθμος:

Αρχικά ορίζουμε τις **p**ublic **p**arameters που είναι:

Α)Μια ελλειπτική καμπύλη (στην περίπτωση του Bitcoin η $\mathbf{x^2 = y^3 + 7}$)

Β)Ένα F_p (Finite field),που θα περιορίσουμε την καμπύλη πάνω σε αυτό (έτσι η καμπύλη με μια πράξη προσθετική γίνεται ομάδα)

Γ)Ένα σημείο τις καμπύλης G,όπου είναι γεννήτορας της

Δ)Ένας ακέραιος n(μεγάλος) οπού είναι η τάξη του G.

Ας πούμε ότι ο Α θέλει να στείλει ένα μήνυμα m στον Β, τότε θα κάνει τα εξής:

Ο Α διαλέγει για private key έναν αριθμό τυχαία στο [1,n-1], τον d_A και για public key υπολογίζει σημείο της καμπύλης $Q_A= d_A \times \mathbf{G}$.

1)Υπολογίζει το e=HASH(m),όπου HASH π.χ sha256

2)Ορίζει z=τα αριστερότερα bits του e τ.ω να έχουν μήκος ίσο με την τάξη n.

3)Διαλέγει $k \in_R [1,n-1]$

4)Υπολογίζει το σημείο της καμπύλης $(x_1,y_1)=k \times \mathbf{G}$

5)Υπολόγιζε $r=x_1 \bmod n$.Αν το r=0 επανέλαβε το βήμα 3)

6)Υπολόγισε $s=k^{-1}(z+rd_A) \bmod n$.Αν s=0 επανέλαβε το βήμα 3)

7)Η υπογραφή είναι το ζευγάρι (r,s)

Σε περίπτωση που διαλέγεται για δυο υπογραφές ίδιο k στο βήμα 3) εύκολα μπορεί κάποιος να ανακτήσει το private key ως εξής:

Έστω (r,s) , (r,s') δυο υπογραφές τότε s-s'= k^{-1}(z-z'). Αφού z,z' είναι γνωστά μπορώ να υπολογίσω το (z-z')$^{-1}$,αρα και το k^{-1} δηλαδή το k.

Στην συνέχεια υπολογίζουμε tor^{-1}(ks-z)=d_A, αφού πλέον το k καθώς και τα z,r(και κατά συνέπεια r^{-1}) είναι γνωστά.

Signature verification algorithm

Αρχικά ο Β κάνει check ότι το public key του Α (δηλαδή το Q_A) είναι σημείο της καμπύλης, ότι δεν είναι το (συμβατικά) ουδέτερο στοιχειό και ότι η τάξη του διαιρεί την τάξη του γεννήτορα G. Αναλυτικότερα:

1)Ελεγξε ότι το $Q_A \in CURVE(F_p)$

2) $Q_A \neq \mathbf{O}$

3)$n \times Q_A = \mathbf{O}$

Στην συνέχεια ο Β ακολουθεί τα εξής βήματα:

1)Τσέκαρε αν τα r,s ∈ [1,n-1], αν όχι επέστρεψε false

2)Υπολόγισε το e=HASH(m)

3) Υπολόγισε z=τα αριστερότερα bits του e τ.ω να έχουν μήκος ίσο με την τάξη n.

4)Υπολόγισε w=s^{-1} mod *n*

5)Υπολόγισε u_1=zw mod *n* και u_2=rw mod *n*

6)Υπολόγισε (x_1',y_1')=$u_1 \times \mathbf{G} + u_2 \times Q_A$

7)Η υπογραφή είναι έγκυρη αν r=x_1 mod *n*

Correctness of the algorithm

Το βήμα 4) υπολογίζει w=k(z+rd_A)$^{-1}$,αρα u_1=zk(z+rd_A)$^{-1}$ και u_2=r k(z+rd_A)$^{-1}$.

Τώρα , (x_1',y_1')=$u_1 \times \mathbf{G} + u_2 \times Q_A$ =(u_1+$d_A u_2$) ×**G**=(zk(z+rd_A)$^{-1}$+d_A r k(z+rd_A)$^{-1}$) ×**G**=(zk+d_A r k)(z+rd_A)$^{-1}$×**G** =k×**G**=(x_1,y_1).Αρα x_1'=x_1.

Και επειδή $r=x_1 \bmod n \overset{x1=x1'}{\Longrightarrow} r=x_1' \bmod n$

2.2 BLOCK

Σε αυτή την ενότητα θα μιλήσουμε για τα blocks **[4],[5].**Τι είναι τα blocks?Είναι μια συλλογή δεδομένων τα οποία μας πληροφορούν ποια νέα transaction έγιναν μέρος του blockchain(?),καθώς και άλλες πληροφορίες (π.χ ποιος έβγαλε το block),πιο αναλυτικά ένα block έχει την παρακάτω μορφή:

Magic no: Μια ακολουθία bytes τα οποία μας δείχνουν σε τι format διαβάζει ο υπολογιστής την πληροφορία(little Endean,big Endean)

Blocksize: Η πληροφορία σχετικά με το πόσο είναι το μέγεθος του block

Blockheader: Αποτελείται από έξι υποπεδία τα οποία θα αναλύσουμε παρακάτω

Transaction counter: Ένας μετρητής που μας πληροφορεί πόσα transactions υπάρχουν στο block

Transactions:Η λίστα με τα transactions που έχουν μπει στο block(παίζει ρόλο η σειρά)

Το blockheader αποτελείται από τα παρακάτω πεδία **[11]**:

Version: Σχετίζεται με το software του bitcoin που χρησιμοποιείται

hashPrevBlock:Το hash (sha-256) του προηγούμενου block header του προηγούμενου block (γι αυτό και η ονομασία blochchain, με αυτόν τον τρόπο τα blocks σχετίζονται μεταξύ τους)

hashMerkleRoot:Το hash (sha-256) του merkle tree (θα γίνει αναφορά στην συνέχεια)

Time:Η ώρα σε unix time (ανανεώνεται μετά από λίγα δευτερόλεπτα)

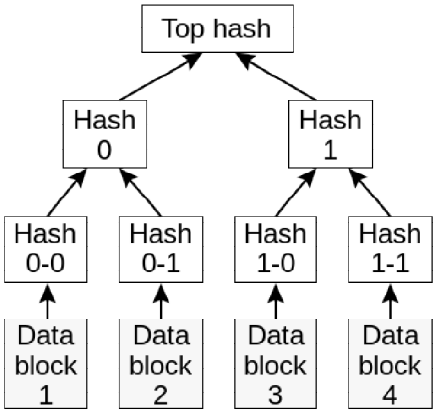
Bits: Η δυσκολία του block (θα το εξηγήσουμε στην ενότητα mining)

Nonce: Ένας 32-bit αριθμός ο οποίος αυξάνεται γραμμικά μέχρι να επιτευχτεί το proof of work (θα το εξηγήσουμε στην ενότητα mining)

2.2.1 MERKLE TREE

Τα merkle trees **[13]** αποτελούν ένα είδος data structure .Πιο συγκεκριμένα, τα leafs του δέντρου απαρτίζονται από κομμάτια data,και τα nodes (εσωτερικοί κόμβοι) από το αποτέλεσμα της hash με είσοδο το padding των δυο παιδιών τους, δηλαδή:

Parent=H(LChild/RChild)



(εικόνα από **[13]**)

Η ρίζα του δέντρου η αλλιώς root hash(top hash) περιέχει κατά κάποιον τρόπο όλη την πληροφορία του δέντρου, εξοικονομώντας έτσι χώρο (αν αλλάξει έστω και ένα bit data με πολύ μεγάλη πιθανότητα θα αλλάξει και το root hash).

Με αυτό τον τρόπο μπορούμε να ελέγχουμε αν data τα οποία λαμβάνουμε είναι σωστά ξέροντας μόνο το root hash και το branch του δέντρου (θα το εφαρμόσουμε στην συνέχεια στο bitcoin μέσω του spv **[5]**).

Για παράδειγμα θέλουμε να κατεβάσουμε ένα (νόμιμο) αρχείο από το διαδίκτυο, η σελίδα σαν έμπιστη αρχή φιλοξενεί το root hash του αρχείου. Εμείς μέσω κατάλληλου λογισμικού κοιτάμε ποιοι peers διαθέτουν αυτό το αρχείο για να το κατεβάσουμε στον υπολογιστή μας. Όμως μπορεί ο χρήστης αν είναι κακόβουλος να μας δώσει corrupted data, ζητάμε και το branch του δέντρου για ταυτοποίηση, για παράδειγμα αν θέλουμε το data block 1,θα ζητήσουμε το hash 0-1 και το hash 1.Εμεις

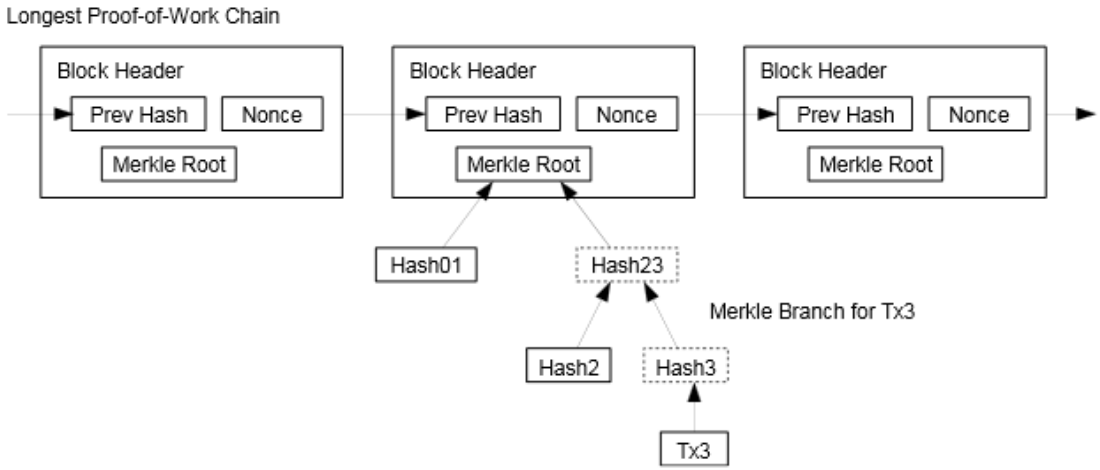
θα υπολογίσουμε το hash 0-0,στην συνέχεια το hash 0=hash(hash 0-0/hash 0-1),και τέλος το result=hash(hash 0/ hash1).

Αν το root hash≠result τότε δεν δεχόμαστε να κάνουμε download (ενδεχομένως να υπάρχει και report για τον χρήστη).

Τώρα μπορούμε να φανταστούμε πως θα χρησιμοποιήσουμε το merkle tree με τα transactions. Αρχικά βάζουμε σε σειρά τα transactions(όπως το παραπάνω παράδειγμα με τα data blocks) του block,κατασκευάζουμε το merkle tree,και το root hash το βάζουμε στο block.

2.2.2 SIMPLIFIED PAYMENT VERIFICATION (SPV)

Με αυτόν τον τρόπο **[3],[5]** ένας light node (για παράδειγμα smartphone) μπορεί να του παρέχεται ένα πιστοποιητικό ότι σε αυτό το block υπάρχουν τα transaction με το συγκεκριμένο merkle root (blockheader). Οποιοσδήποτε ισχυριστεί ότι υπάρχει ένα transaction σε ένα block που δεν υπάρχει ,ο light node θα ζητήσει απόδειξη, πράγμα που δεν γίνεται να την παρέχει ο full node κάτω από την υπόθεση της hash function(ότι δηλαδή είναι κρυπτογραφικά ασφαλές) και ότι η πλειοψηφία του cpu power του δικτύου δεν ελέγχεται από κακόβουλους χρήστες. Ο λόγος ύπαρξης light node είναι λόγω μεγάλων (σχετικά) απαιτήσεων του να είσαι full node, για παράδειγμα αυτή την στιγμή να κατεβάσει κανείς όλο το blockchain είναι περίπου 23 gb, πράγμα που το καθιστά δύσχρηστο σε αρκετές φορητές συσκευής (γενικά από τον όγκο των δεδομένων που μαζεύονται στο blockchain υποφέρουν και άλλα νομίσματα , αυτό λέγεται Scalability).Ο τρόπος επαλήθευσης φαίνεται από την παρακάτω εικόνα **[3]:**



(εικόνα από **[3]**)

2.2.3 DOUBLE SPENDING

Ένας από τους κύριους λόγους που εισήχθηκε αυτό το αποκεντρωτικό σύστημα πληρωμών **[3]** είναι και το double spending,δηλαδή να μην μπορεί κάποιος να πληρώνει με το ίδιο χρηματικό ποσό δυο οι και περισσότερους χρήστες. Αυτό λύνεται με την καταγραφή τις χρηματικής δραστηριότητας κάθε χρήστη η οποία γίνεται μέσω των transaction που υπάρχουν στα blocks της κύριας αλυσίδας. Έτσι για παράδειγμα αν ο Α στείλει 10 BTC στον Β και 10 BTC στον Γ όπου σαν reference transaction χρησιμοποίει το ίδιο και στις δυο περιπτώσεις ,τότε οι verifying node του δικτύου:

Α)Στην περίπτωση που το transaction από τον Α στον Β υπάρχει ήδη σε block,τότε θα απορρίψουν το transaction από τον Α στον Γ.

Β)Στην περίπτωση που έχουν γίνει broadcast την ίδια στιγμή, τότε θα κάνουν ένα από τα δυο δεκτά και θα ενημερώνουν παράλληλα το δυο, έτσι ένα από τα δυο θα επικρατήσει ως δεκτό και το άλλο ως double spending πριν μπουν στο block.

2.2.4 MINER'S INCENTIVE

Ένας miner πριν ανακοινώσει το block που έκανε mining (αναλυτικότερα για την διαδικασία αυτή θα δούμε στην ενότητα proof of work),μαζί με τα transactions που συμπεριέλαβε στο block του θα βάλει και ένα ειδικό transaction ,που ονομάζεται coinbase **[3],[5]** όπου θα στέλνει BTC στο address του (με αυτό τον τρόπο εξασφαλίζεται επίσης ότι κάθε miner το block που θα κάνει mining θα έχει διαφορετικό merkle root και κατά συνέπεια θα διαφέρει το nonce του block τους),στην ουσία για κάθε block που βγαίνει το σύστημα ¨κόβει¨ bitcoins,και ο miner έχει κίνητρο να συμμετέχει στην διαδικασία του mining.Επίσης για κάθε transaction που θα συμπεριλάβει στο block του θα πάρει και τα αντίστοιχα fees,οπότε όσο περισσότερα συμπεριλάβει τόσο καλύτερα γι'αυτόν αλλά και το δίκτυο (σε περίπτωση που κάποιο από αυτά δεν είναι έγκυρα τότε το block που θα βγάλει θα το απορρίψει το υπόλοιπο δίκτυο, γι αυτό πρέπει να γίνουν verify πριν τα συμπεριλάβει πράγμα το οποίο γίνεται γρήγορα λόγω τις απλής δομής του transaction και τις script base language που χρησιμοποιεί).

2.3 MINING

Σε αυτή την ενότητα θα ασχοληθούμε με το proof of work(απόδειξη εργασίας),διαφόρους τρόπους mining,καθώς και την ασφάλεια που παρέχει το bitcoin σε μια επίθεση λιγότερο του 50% cpu power του δικτύου.

2.3.1 PROOF OF WORK

Ο κάθε miner πριν ανακοινώσει το νέο block πρέπει να παρέχει μια απόδειξη στο δίκτυο **[3],[5]** ότι εργάστηκε πάνω σε αυτό και ότι το δικό του block αξίζει να το δεχτούν ως το αληθινό, αλλά τι σημαίνει αληθινό και ότι εργάστηκε?.Αληθινό block είναι ένα block το οποίο πληροί τις τυπικές προϋποθέσεις, όπως για παράδειγμα όλα τα transactions που έχει να είναι έγκυρα, το merkle root να είναι σωστό, το timestamp του block ναι είναι τουλάχιστον όσο ο μέσος όρος των timestamp των τελευταίων 11

blocks και όχι περισσότερο από 2 ώρες από το τελευταίο block κ.α. Ότι εργάστηκε σημαίνει ότι το hash του block (και συγκεκριμένα του block header),να είναι μικρότερο από έναν αριθμό ***ε*** (ο οποίος προκύπτει από τη δυσκολία του block και το πεδίο τιμών της hash),πιο αναλυτικά:

$POW(H_n,n)<\varepsilon$, όπου n το nonce του block.

Όποιος πληροί τα παραπάνω ,τα ανακοινώνει στο δίκτυο και γίνεται δεκτό στο blockchain.

Για κάθε δοκιμή με κάποια τιμή nonce ,έχει την ίδια πιθανότητα να πετύχει τον στόχο, γι αυτό και μεγαλύτερο cpu power συνεπάγεται μεγαλύτερη πιθανότητα κάποιος να πετύχει των στόχο όχι λόγω μεγαλύτερης πιθανότητας ανά δοκιμή αλλά λόγω περισσοτέρων δοκιμών στο ίδιο χρονικό διάστημα.

Το proof of work που παρέχει το bitcoin **[3]** ενώ αρχικά φαινόταν ¨δίκαιο¨ στην συνέχεια αντιμετώπισε/ζει πρόβλημα. Αρχικά το mining γινότανε μέσω cpu, στην συνέχεια μέσω vga(κάρτα γραφικών),και τέλος μέσω Asic.Τα Asic είναι εξειδικευμένο hardware το οποίο μέσω παράλληλου υπολογισμού μπορεί να παράγει έως και 700 GH/S ,σε αντίθεση ενός συμβατικού επεξεργαστή που οι τιμές ανέρχονται περίπου 10 MH/S.Ο λόγος που επιτρέπει την δημιουργία ενός τέτοιου hardware είναι ο απλός τρόπος του pow (συνεχόμενες δοκιμές hash χωρίς μεγάλες απατήσεις ram).Εστί ένα entry cost στην διαδικασία του mining ανέρχεται περίπου στα 1500 δολάριο, χωρίς να μπορεί να εξασφαλίσει κανείς αν θα βγάλει τα ¨πραγματικά ¨χρήματα που ξόδεψε ,καθώς η τιμή του bitcoin είναι αρκετή μεταβαλλόμενη.

Υπάρχουν δυο είδη mining **[5]:**

Το solo mining ,όπου ο κάθε miner κάνει mining μόνος του

Το mining pool, όπου πολλοί miners ενώνουν το cpu power τους**[6]** και δουλεύουν σαν ένας node στο δίκτυο. Στο τέλος μέσω ειδικού software δίνεται στον καθένα αμοιβή ανάλογη με την συνεισφορά του στον υπολογισμό του block.

Ο δεύτερος σήμερα είναι και ο επικρατέστερος τρόπος mining. Αρχικά ακούγεται καλός σαν πιο σίγουρο κέρδος αλλά λόγω ότι οι participant σε ένα τέτοιο pool ζητούν πληροφορίες από το ίδιο το pool για το blockchain, υπάρχει ένα είδος κεντροποίησης (δεδομένου ότι αυτή την στιγμή τα 3 μεγαλύτερα mining pools κατέχουν περίπου το 50% cpu power του δικτύου).

Η δυσκολία που πρέπει να έχει το block αυξάνεται ή μειώνεται ανάλογα με τον χρόνο που κάνει για να βγει ένα νέο block (περίπου 10 λεπτά),αν βγει πιο γρήγορα αυξάνεται ,αντίθετα μειώνεται (ο στόχος είναι κάθε 2 εβδομάδες να βγαίνουν 2016 blocks).Η δυσκολία επιτυγχάνεται δεσμεύοντας bits από αριστερά στο **ε** ως 0 (για κάθε τέτοια δέσμευση η δυσκολία αυξάνει εκθετικά).

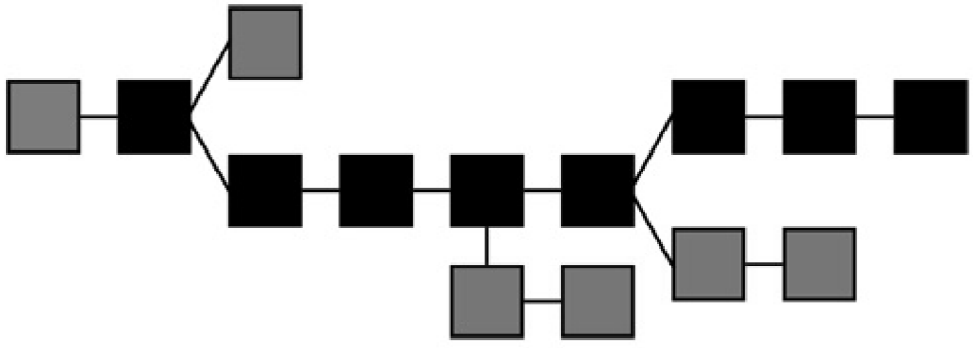
2.3.2 BLOCKCHAIN

Όπως έχει αναφερθεί ήδη παραπάνω το μεγαλύτερο blockchain στο bitcoin network είναι και το αποδεκτό. Η λογική πίσω από αυτό είναι ότι το στο μεγαλύτερο σε μήκος blockchain έχει γίνει και το μεγαλύτερο proof of work και έτσι για κάποιον που θέλει να κάνει fork από ένα σημείο και έπειτα στο blockchain ,πρέπει και να ξανακάνει όλο το αντίστοιχο proof of work,πράγμα που το κάνει υπολογιστικά μη αντιστρεπτό.

Παρακάτω θα δώσουμε κάποιους ορισμούς**[6],[5]:**

Stale Block:Ονομάζουμε το block για το οποίο ήδη κάποιος άλλος έχει ανακοινώσει το δικό του block ,έτσι λέμε ότι το block μας έχει γίνει stale(π.χ δουλεύουμε το block 3000 και πριν το ανακοινώσουμε κάποιος άλλος το ανακοινώνει, έτσι λέμε ότι εργαζόμασταν σε ένα stale block).

Orphan Block:Ονομάζουμε το block για το οποίο το δίκτυο δεν συνέχισε το chain για το οποίο ανήκε(ο πατέρας του δηλαδή δεν ανήκει κατ' ανάγκη στην κύρια αλυσίδα).

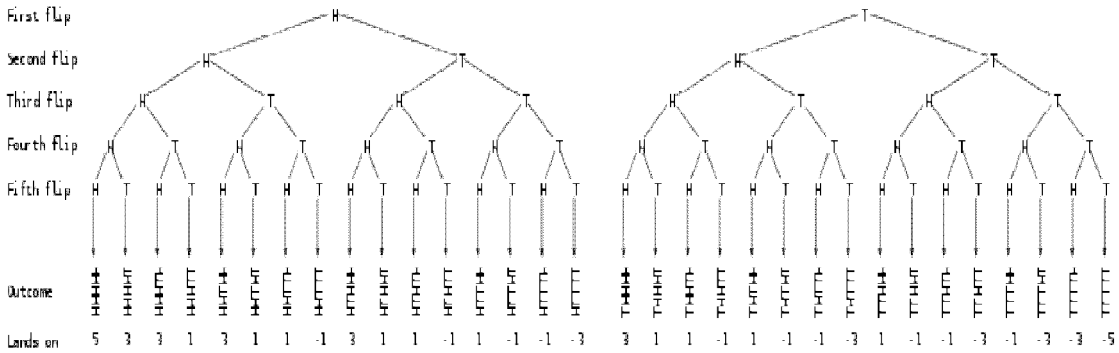


(εικόνα από en.bitcoin.it/wiki/Block_chain)

Παρατηρήσεις

Τι γίνεται στην περίπτωση που κάποιος έχει λιγότερο από 50% hash power του δικτύου? Μπορεί να προκαλέσει κάποιο πρόβλημα με μη αμελητέα πιθανότητα?(όπως αυτό τις 51 %attack). Αρχικά θα περιγράψουμε το πρόβλημα του τζογαδόρου**[1],[2]**.

Έχουμε έναν τζογαδόρο ο οποίος αρχίζει με ένα χρηματικό ποσό i,και σε κάθε ρίψη ενός όχι κατ' ανάγκη αμερόληπτου κέρματος(θα θεωρήσουμε ότι δεν είναι αμερόληπτο) αν πετυχαίνει κορόνα η περιουσία του αυξάνεται κατά μια μονάδα αλλιώς μειώνεται κατά μια μονάδα, όπως φαίνεται παρακάτω: (για πέντε δοκιμές)



(εικόνα από http://en.wikipedia.org/wiki/Random_walk)

Στόχος είναι να φτάσει την περιουσία του N μονάδες(όπου θα έχει πετύχει τον στόχο του), ή να φαλιρίσει (όπου αυτομάτως χάνει).Η πιθανότητα επιτυχίας του για κάθε δοκιμή είναι p, ενώ η πιθανότητα αποτυχίας q=1-p.Για κάθε αρχικό ποσό i που ξεκινά ο τζογαδόρος μπορούμε να φανταστούμε την διαδικασία του πειράματος σαν ένα δυαδικό δέντρο με πιθανότητες μετάβασης από τον πατέρα στα παιδιά του p για το αριστερό παιδί και q για το δεξί, με τα φύλλα του να είναι είτε επιτυχία (δηλαδή κατάφερε να κάνει την περιουσία του N),η αποτυχία (φαλίρισε, περιουσία 0).Πιο φορμαλιστικά:

R_n θα συμβολίζουμε την περιουσία του μετά την n -ιοστη δοκιμή, Δ_n η τυχαία μεταβλητή τ.ω $\Delta_n \in (-1,1)$ που μας πληροφορεί για το αποτέλεσμα της ρίψης στην n -ιοστη δοκιμή(πιο συγκεκριμένα αν είναι κορόνα το Δ=1,αλλιος -1).Άρα το $R_n = \Delta_1 + \Delta_2 + ... + \Delta_n$ και $R_0=i$.Επίσης $Pr(\Delta=1)=p$ και η $Pr(\Delta=-1)=q$.

Θα ορίσουμε έναν τελεστή $\tau_i = \min\{n \geq 0, R_n \in (0,N) | R_0=i\}$,δηλαδή το μικρότερο n για το οποίο έχουμε πετύχει τον στόχο μας (η περιουσία μας είναι N, η φαλιρίσαμε),καθώς από εκεί και έπειτα δεν μας ενδιαφέρει να συνεχιστεί το πείραμα, ξεκινώντας από το χρηματικό ποσό i.

Άρα όπως ορίσαμε το πρόβλημα αρχικά, αν $R_{\tau i}=N$ ο τζογαδόρος κερδίζει, αλλιώς αν $R_{\tau i}=0$ ο τζογαδόρος φαλίρισε. Στο πείραμα αυτό μας ενδιαφέρει να προσδιορίσουμε ποιες είναι αυτές οι πιθανότητες.

Ορίζουμε σαν $P_i=Pr(R_{\tau i}=N)$.Εύκολα παρατηρούμε ότι $P_0=0$ και $P_N=1$, για $i\in[1,N-1]$.Με αυτή την παρατήρηση και με τον τρόπο που έχουμε ορίσει τον τελεστή τ_i,αν κάθε φορά ερμηνεύαμε την πιθανότητα P_i συναρτήσει των δύο πιθανών αποτελεσμάτων (επιτυχία ,αποτυχία),κάποια στιγμή(αναδρομικά) θα μπορούσαμε να κατασκευάσουμε την πιθανότητα(θα καταλήγαμε στην πρώτη φορά που θα βλέπατε επιτυχία ή αποτυχία). Ετσι $P_i=Pr(R_{\tau i}=N)=$ $Pr(R_{\tau i}=N|\ \Delta_1=1)\ Pr(\Delta_1=1)+\ Pr(R_{\tau i}=N|\ \Delta_1=-1)\ Pr(\Delta_1=-1)\ =$ $p\cdot P_{i+1}+q\cdot P_{i-1}$. **(1)**

Λόγω ότι τα p,q είναι συμπληρωματικά(p+q=1) και τις **(1)** έχουμε:

$p\cdot P_{i+1}+q\cdot P_{i-1}=p\cdot P_i+q\cdot P_i\Rightarrow P_{i+1}-P_i=\frac{q}{p}(P_i-P_{i-1})$ (ο λόγος που επιλέξαμε αυτή την σειρά είναι ότι θα μας διευκόλυνε για τα αναπτύγματα όπως ακριβώς στις τηλεσκοπικές σειρές).

Στην συνέχεια αθροίζοντας κατά μελή τις παρακάτω εξισώσεις προκύπτει:

$P_2-P_1=\frac{q}{p}(P_1-P_0)=\frac{q}{p}P_1$

$P_3-P_2=\frac{q}{p}(P_2-P_1)=(\frac{q}{p})^2 P_1$

$P_{i+1}-P_i=(\frac{q}{p})^i P_1$

Προσθέτοντας κατά μελή τις παραπάνω σχέσεις προκύπτει ότι:

$P_{i+1}-P_1=\sum_{k=1}^{i}(\frac{q}{p}^k P_1)\Leftrightarrow P_{i+1}=\sum_{k=0}^{i}(\frac{q}{p}^k)P_1\overset{p\neq q}{\Longleftrightarrow}P_{i+1}=P_1\frac{1-(\frac{q}{p})^{i+1}}{1-\frac{q}{p}}$ **(2)**

Λόγω ότι το P_N είναι γνωστό, αντικαθιστούμε όπου i=N-1, και έτσι έχουμε:

$P_1=\frac{1-\frac{q}{p}}{1-(\frac{q}{p})^N}$, έτσι αντικαθιστώντας στην **(2)** έχουμε:

$P_{i+1}=\frac{1-(\frac{q}{p})^{i+1}}{1-(\frac{q}{p})^N}\Leftrightarrow P_i=\frac{1-(\frac{q}{p})^{i}}{1-(\frac{q}{p})^N}$

Τώρα για p>q , παρατηρούμε ότι για $N\to\infty$ (θέλουμε το κέρδος μας να είναι άπειρο πριν σταματήσουμε να παίζουμε) αυτή η πιθανότητα συγκλίνει(από το γεγονός ότι $\frac{q}{p}<1$),πιο συγκεκριμένα:

$\lim_{N\to\infty} P_i = 1 - (\frac{q}{p})^i$

Παρατηρούμε ότι τώρα η πιθανότητα μας εξαρτάται μόνο από το αρχικό budget.

Έτσι η πιθανότητα να βγάλουμε άπειρο κέρδος είναι $1 - (\frac{q}{p})^i$, ενώ να φαλιρίσουμε $(\frac{q}{p})^i$.

Τώρα στο αρχικό μας πρόβλημα **[2],[3],** έστω ότι ένας κακός χρήστης στο bitcoin network έχει πιθανότητα να βρει το επόμενο block q, και το υπόλοιπο network πιθανότητα p,με q+p=1 και p>q (το event εδώ θεωρείται το να βγει κάποιο block,γι αυτό είναι και συμπληρωματικά τα ενδεχόμενα, και ότι ο κακός έχει λιγότερο από 50% του hash power του δικτύου)

Ας σκεφτούμε ότι κάποιος χρήστης Α πραγματοποιεί ένα transaction ,και αυτό στην συνέχεια μπαίνει στο block(αφότου πραγματοποιηθούν οι διαδικασίες που περιγράψαμε στις προηγούμενες ενότητες).Στην συνέχεια ο Α για να βεβαιωθεί ότι το block στο οποίο βρίσκεται το transaction του θα παραμείνει στην κύρια αλυσίδα (δες παραπάνω stale block,orpahn block),περιμένει να μπουν και αλλά z-1 blocks ακόμα. Τώρα ο κακός με το που κάνει ο χρήστης Α broadcast το transaction του στο δίκτυο, ξεκινά και δουλεύει στο δικό του private chain μη έχοντας συμπεριλάβει πουθενά το transaction που πραγματοποίησε ο χρήστης Α. Μας ενδιαφέρει να μελετήσουμε την πιθανότητα ο κακός χρήστης να κάνει catch up το main chain.

Παρατηρούμε ότι αυτό το πρόβλημα μοιάζει με το πρόβλημα του τζογαδόρου που περιγράψαμε με αρχικό budget να είναι η απόσταση του private chain με το main chain,q η πιθανότητα να προχωρήσει το private chain κατά ένα block (δηλαδή η απόσταση να μείνει κατά 1,αρα -1),p η πιθανότητα το main chain να προχωρήσει κατά ένα block (δηλαδή η απόσταση να αυξηθεί κατά ένα, άρα +1).

Ορίζουμε ως q_z την πιθανότητα ο κακός να κάνει catch up το main chain αφότου έχουν μπει z blocks.

Συμφώνα με την ανάλυση στο πρόβλημα του τζογαδόρου αυτή η πιθανότητα ισούται με: $q_z = (\frac{q}{p})^z$.

Παρατηρούμε ότι καθώς το z αυξάνει η πιθανότητα ο επιτιθέμενος να φτάσει την κύρια αλυσίδα μειώνεται εκθετικά.

Τώρα μας ενδιαφέρει αφότου z blocks έχουν μπει στην κύρια αλυσίδα τι πρόοδο έχει κάνει ο επιτιθέμενος στην δική του αλυσίδα, σε αυτό το διάστημα. Θεωρούμε ότι η πιθανότητα επιτυχίας του επιτιθέμενου προσεγγίζεται από την κατανομή Poisson με παράμετρο λ=nq**.** **(1)** Επειδή και ο επιτιθέμενος αλλά και το honest party ακολουθούν επίσης διωνυμική κατανομή μπορούμε να προσδιορίσουμε το n ως εξής:

Θεωρούμε ότι το honest party τα z block που βρήκε είναι τα expected,δηλαδή $\rightarrow z=np \Leftrightarrow n=\frac{z}{p}$ **(2).**

Άρα από **(1)** και **(2),** $\boldsymbol{\lambda=\frac{q}{p}z.}$

Έτσι ορίζοντας ως A το ενδεχόμενο να νικήσει ο επιτιθέμενος, και A_i να νικήσει έχοντας βρει i blocks (τα z blocks του main chain τα κρατάμε σταθερά),η πιθανότητα που μας ενδιαφέρει να υπολογίσουμε είναι:

$Pr(A)=Pr(A_1 \cup A_2 \cup A_3 \cup \ldots\ldots \cup A_z \cup \ldots)=\sum_{i=1}^{\infty} Pr(A_i)$

Ισχύει ότι $Pr(A_i)$=Pr(Να βρει i blocks)Pr(Να κάνει catch up)$=\frac{\lambda^i e^{-\lambda}}{i!}(\frac{q}{p})^{z-i}$ για $i< z$

Άρα $Pr(A)= \sum_{i=0}^{\infty} \frac{\lambda^i e^{-\lambda}}{i!} \begin{cases} (\frac{q}{p})^{z-i} & \text{για } i < z \\ 1 & \text{αλλιος} \end{cases}$

Γνωρίζουμε ότι $\sum_{i=0}^{\infty} \frac{\lambda^i e^{-\lambda}}{i!}=1 \Leftrightarrow \sum_{i=0}^{z} \frac{\lambda^i e^{-\lambda}}{i!} + \sum_{i=z+1}^{\infty} \frac{\lambda^i e^{-\lambda}}{i!} =1$
$\Leftrightarrow \sum_{i=z+1}^{\infty} \frac{\lambda^i e^{-\lambda}}{i!}=1-\sum_{i=0}^{z} \frac{\lambda^i e^{-\lambda}}{i!}$ **(3)**

Έτσι από την **(3)** $Pr(A)=1-\sum_{i=0}^{z} \frac{\lambda^i e^{-\lambda}}{i!}(1-(\frac{q}{p})^{z-i})$. (Η (3) έγινε για να μην αθροίσουμε την άπειρη ουρά Poisson).

Παρακάτω παρατίθενται κάποιοι υπολογισμοί, σχετικά με το πόσο είναι αυτή η πιθανότητα ανάλογα το z και το q:

q=0.1
z=0 P=1.0000000

z=1 P=0.2045873
z=2 P=0.0509779
z=3 P=0.0131722
z=4 P=0.0034552
z=5 P=0.0009137
z=6 P=0.0002428
z=7 P=0.0000647
z=8 P=0.0000173
z=9 P=0.0000046
z=10 P=0.0000012

q=0.3
z=0 P=1.0000000
z=5 P=0.1773523
z=10 P=0.0416605
z=15 P=0.0101008
z=20 P=0.0024804
z=25 P=0.0006132
z=30 P=0.0001522
z=35 P=0.0000379
z=40 P=0.0000095
z=45 P=0.0000024
z=50 P=0.0000006

2.3.3 SELFISH MINING

Σε αυτού του είδους mining **[8]** ο κακόβουλος miner δουλεύει στο δικό του block όπως ο κάθε miner με την διαφορά ότι για κάθε block που βρίσκει δεν το ανακοινώνει στο δίκτυο, συνεχίζοντας έτσι να κάνει mining στο δικό του block που δεν το έχει ανακοινώσει ακόμα στο δίκτυο. Με αυτό τον τρόπο κρατάει τους άλλους miner στο σκοτάδι μη ξέροντας αν το block το οποίο δουλεύουν ανήκει στην κύρια αλυσίδα ή τελικά θα γίνει orphan.

Πιο συγκεκριμένα στο**[9]**, έδειξαν μια νέα είδους επίθεση(οικονομική επίθεση). Κάποιος μπορεί να επιτεθεί στο δίκτυο διαθέτοντας μόνο το 25% hashpower του δικτύου. Πιο συγκεκριμένα ο αλγόριθμος που ακολουθεί ο κακός είναι ο εξής(όπου q η πιθανότητα ο κακόβουλος να βρει το επόμενο block,p η πιθανότητα το υπόλοιπο δίκτυο να βρει το επόμενο block,και Z η πιθανότητα το υπόλοιπο δίκτυο να κάνει mining (on top)στο block του κακόβουλου:

Στάδιο 0:Ο κακόβουλος δουλεύει πάνω στο τελευταίο block του main chain.Με πιθανότητα q,θα βρει το επόμενο block,τότε θα αυξήσει το κέρδος του κατά 1 (25 BTC),και θα πάει στο στάδιο 1.Με πιθανότητα p,το υπόλοιπο δίκτυο θα βρει το επόμενο block και θα το δημοσιεύσει, τότε ο επιτιθέμενος επαναλαμβάνει το στάδιο 0.

Στάδιο 1:Ο επιτιθέμενος με πιθανότητα q,βρίσκει και άλλο block στο private chain (αυξάνοντας το κέρδος του κατά 1) και πάει στο στάδιο 2, αλλιώς με πιθανότητα p το υπόλοιπο δίκτυο βρίσκει το επόμενο block και το ανακοινώνει, τότε ο επιτιθέμενος πάει στο στάδιο 0΄

Στάδιο 0΄:Ο επιτιθέμενος ανακοινώνει το block του. Με πιθανότητα q ο επιτιθέμενος θα βρει το επόμενο block κανόνας το υπόλοιπο δίκτυο να εργάζεται στο δικό του chain (εδώ το private chain είναι κατά 1 μεγαλύτερο από το public chain),και αυξάνοντας το κέρδος του κατά 2.Με πιθανότητα p·Z το δίκτυο θα βρει το επόμενο block κτισμένο πάνω στο block του επιτιθέμενου, έτσι το δίκτυο θα έχει κέδρος 1 και ο επιτιθέμενος 1.Αλλιως με πιθανότητα p·(1-Z) το δίκτυο θα βρει το επόμενο block κτισμένο πάνω στο δικό του block,έτσι το δίκτυο θα έχει κέρδος 2 ενώ ο επιτιθέμενος 0 και θα ξένα γυρίσει στο στάδιο 0.

Σταδιο2:Με πιθανότητα q ο επιτιθέμενος βρίσκει το επόμενο block (αυξάνοντας το κέρδος του κατά 1),πηγαίνοντας έτσι στο στάδιο 3,αλλιώς με πιθανότητα p το υπόλοιπο δίκτυο βρίσκει το επόμενο block και το ανακοινώνει, έτσι και ο επιτιθέμενος ανακοινώνει τα δυο δικά του block,αυξάνοντας έτσι το κέρδος του κατά 2.

Στάδιο n: Με πιθανότητα q ο επιτιθέμενος βρίσκει το επόμενο block,πηγαίνοντας έτσι στο στάδιο n+1, αλλιώς με πιθανότητα p το υπόλοιπο δίκτυο βρίσκει το επόμενο block και το ανακοινώνει, έτσι ο επιτιθέμενος επιστρέφει στο στάδιο n-1.

Το Z ουσιαστικά το μελετάμε μόνο για το στάδιο 0΄.Ποσο είναι το Z όμως?

Λόγω του ότι ο αλγόριθμος του επιτιθέμενου είναι reactive (δηλαδή πρέπει πρώτα να προβεί σε δημοσίευση block το υπόλοιπο δίκτυο για να δημοσιεύσει και ο επιτιθέμενος),το Z στο στάδιο 0΄είναι πάντα 0,καθος από τον αλγόριθμο του bitcoin το πρώτο block που θα πάρουν οι miners, σε αυτό θα κάνουν mining.Όμως ο επιτιθέμενος μπορεί να εισάγει αρκετούς κόμβους στο δίκτυο(Sybil attack),και να κάνουν το δικό του block broadcast πρώτα. Μια εγγύηση ότι το Z δεν θα φτάσει ποτέ 1 είναι ότι κάθε miner λαμβάνει πρώτα το block του mining poll που ανήκει ,και λόγω ότι το μεγαλύτερο hashpower ενός mining pool αγγίζει το 20%,το $Z \leq 0,8$.Μια τακτική θα ήταν οι miners να διάλεγαν τυχαία πάνω σε πιο block θα κάνουν mining,πηγαίνοντας έτσι το Z=0,5.Οι Ittay Eyal και Emin Gun Sirer έδειξαν ότι για Z=0,5 και $q \geq 0,25$ ο επιτιθέμενος γίνετε πιο αποδοτικός από το υπόλοιπο δίκτυο ,ενώ για $q \geq \frac{1}{3}$ γίνεται πιο αποδοτικός για οποιοδήποτε Z(πιο αποδοτικός σε σχέση με

τον λόγο ενέργειας, κέρδους από το υπόλοιπο δίκτυο).Έτσι σε περίπτωση που ένας τέτοιος κακός δηλώσει τις προθέσεις του (δηλαδή να επιτεθεί με αυτόν τον τρόπο στο δίκτυο),όλοι οι miner που είναι μεν τίμιοι αλλά επιθυμούν να βελτιστοποιήσουν το κέρδος τους θα συνεισφέρουν και αυτή στην επίθεση (δηλαδή θα γίνουν ¨κακοί¨).Κάτι τέτοιο θα είχε σαν συνέπεια τα mining pool να θέτουν δελεαστικές προσφορές ώστε να μην φύγουν οι miners(χωρίς να γνωρίζουμε στα σίγουρα τι θα γινόταν).Συνοψίζοντας ,αν ο κακός στο δίκτυο κατείχε το 25% του hashpower και ένα ¨κάλο¨ budget για το Sybil attack,θα ανακοίνωνε τις προθέσεις του για να φτάσει hashpower$\geq \frac{1}{3}$,έτσι ώστε η απόδοση του να μην εξαρτάται πλέον από το Z(χωρίς να χρειάζεται πλέον να κάνει Sybil attack).

Μέχρι στιγμής οι miner δρουν αλτρουιστικά ,και θέλουν να συνεισφέρουν στο δίκτυο έτσι ώστε να μην χάσει τον αποκεντρωτικό του χαρακτήρα.

2.4 CONTRACTS

Σε αυτή την τελευταία ενότητα του bitcoin θα αναφέρουμε μερικά από τα συμβόλαια **[12]** που για την υλοποίηση τους χρησιμοποιούν τον μηχανισμό του bitcoin (blockchain,proof of work,transaction, κ.α).

2.4.1 Παροχή πειστηρίου

Ας υποθέσουμε ότι ένας χρήστης θέλει να ανοίξει κάποιον λογαριασμό σε μια ιστοσελίδα(π.χ Wikipedia).Σε αυτή την περίπτωση η σελίδα θέλει κάποιο πειστήριο ότι ο χρήστης δεν είναι spammer (δηλαδή να μπορεί να φτιάξει χιλιάδες account στην σελίδα ,με αποτέλεσμα να κάνει κακό στον παροχέα).Μια ιδέα θα ήταν ότι όσο ο χρήστης έχει ένα ενεργό account να έχει δεσμευτεί γι'αυτό ένα χρηματικό ποσό στην σελίδα ,έτσι ώστε να μην μπορεί να ανοίγει όσα account θέλει .Σε αυτή την περίπτωση ερχόμαστε αντιμέτωποι με άλλα προβλήματα ,όπως αν θα του επιστραφούν τα χρήματα μετά το πέρας λειτουργιάς του account?,δηλαδή ερχόμαστε αντιμέτωπη με προβλήματα εμπιστοσύνης όχι από την πλευρά του παροχέα αλλά από την πλευρά του χρήστη προς τον παροχέα .Ένα συμβόλαιο που λύνει αυτό το πρόβλημα είναι το παρακάτω:

1)Ο χρήστης και ο παροχέας ανταλλάζουν bitcoin adress(μπορεί να ανακτηθεί το public key)

2)Ο χρήστης φτιάχνει το transaction Tx1 στο οποίο στέλνει το χρηματικό ποσό που έχουν συμφωνήσει με τον παροχέα να δεσμευτεί ,ας πούμε 10 BC,στις διευθύνσεις και τον δυο (δηλαδή για να εξαργυρωθεί το ποσό απαιτείται τόσο η υπογραφή του χρήστη αλλά και του ίδιου του παροχέα),αλλά δεν το κάνει broadcast(διότι όπως θα δούμε παρακάτω αν το έκανε μπορεί τα χρήματα του να εγκλωβίζονταν για πάντα).Στην συνέχεα στέλνει το $Tx1_{id}$ στον παροχέα.

3)Ο παροχέας φτιάχνει το Tx2 μη ξέροντας το Tx1 παρά μόνο το id του και το ποσό που δεσμεύεται ο χρήστης. Στέλνει το ποσό πίσω στον χρήστη θέτοντας το nlocktime στο Tx2 στο συμφωνηθέν (π.χ 6 μήνες),στην συνέχεια βάζει την υπογραφή του και το στέλνει πίσω στον χρήστη (δεν μπορεί να το κάνει broadcast και να ήθελε διότι λείπει η υπογραφή του χρήστη αλλά εκτός αυτού, το βασικότερο ,δεν έχει δημοσιευτεί το Tx1 που αυτό είναι για την ασφάλεια του χρήστη).

4)Ο χρήστης ελέγχει αν όλα τα πεδία του Tx2 είναι σωστά (το nlocktime,ότι τα χρήματα θα επιστρέψουν πίσω σε αυτόν κ.α),και στην συνέχεια το υπογράφει.

5)Ο χρήστης δημοσιεύει το Tx1,και στην συνέχεια το Tx2.

Αν για οιοδήποτε λόγο ο χρήστης επιθυμεί να τερματίσει τον λογαριασμό του νωρίτερα ,τότε φτιάχνουν μια νέα έκδοση του Tx2(κανονικά θα έπρεπε να το πούμε Tx3),με nlocktime μειωμένο σε σχέση με το αρχικό ,η αν ο χρήστης επιθυμεί να το ανανεώσει μετά το πέρας του χρονικού ορίου που έχουν συμφωνήσει, ξανά επαναλαμβάνουν την διαδικασία (κανονικά αυτό μπορούσε να γίνει χωρίς να επαναλάβουν όλη την διαδικασία με την χρήση των sequence numbers που σου

επιτρέπουν να τροποποιήσεις αρκετές φορές ένα transaction πριν την τελική του μορφή αντικαθιστώντας έτσι την προηγούμενη του έκδοση στο memory pool,αλλά έχει καταργηθεί αυτή η λειτουργιά από το λογισμικό του bitcoin για λόγους ασφάλειας).

2.4.2 Χρησιμοποιώντας έναν εξωτερικό παράγοντα

Σε αυτού του είδους τα συμβόλαια μας ενδιαφέρει να καταλήξουμε σε μια συμφωνία χρησιμοποιώντας κάποια συνθήκη που είναι εκτός των πλαισίων του συστήματος του bitcoin ,για παράδειγμα το party A θα πάρει 20 BC εφόσον η θερμοκρασία την τετάρτη θα είναι μικρότερη τον 25 c^o.Κάτι τέτοιο εκ πρώτης όψεως μοιάζει δύσκολο να πραγματοποιηθεί με το σύστημα του bitcoin),παρ όλα αυτά το παρακάτω συμβόλαιο εισάγει μια γενικότερη τεχνική για αυτού του είδους τα συμβόλαια.

Ας υποθέσουμε ότι ένας παππούς θέλει να δώσει στον εγγονό του κάποιο χρηματικό ποσό για τα 18 του γενέθλια η αφότου πεθάνει ,ότι γίνει πρώτα (ελπίζουμε να γίνει το πρώτο αλλά είναι υποχρέωση μας να είμαστε καλυμμένοι και στις δυο περιπτώσεις!)

Η συνθήκη τον 18 επιτυγχάνεται εύκολα. Ο παππούς φτιάχνει ένα transaction με το ακριβές ποσό που θέλει να δώσει στον εγγονό του (υποθέτουμε ότι είναι ένας παππούς γνώστης της τεχνολογίας!) βάζοντας σαν nlocktime την χρονική περίοδο που γίνετε 18.Στην συνέχεια το δίνει στον εγγονό του(θα μπορούσε να το δημοσιοποιήσει αλλά λόγω του μεγάλου χρονικού διαστήματος που θα περνούσε θα έμενε αρκετό καιρό στο memory pool με αποτέλεσμα να μην ήταν υψηλής προτεραιότητας transaction και θα μπορούσε να περίμενε αρκετά blocks μέχρι να το βάλει κάποιος miner),ο οποίος όταν γίνει 18 θα το δημοσιεύσει και θα πάρει το χρηματικό ποσό.

Η συνθήκη του θανάτου είναι πιο περίπλοκη .Όπως και με το παράδειγμα με την θερμοκρασία ,πως στο σύστημα του bitcoin μπορεί να ελέγξει/πιστοποιήσει κάποιος ένα γεγονός του εξωτερικού κόσμου ?(στην συγκεκριμένη περίπτωση σχετικά με το αν ζει ο παππούς η όχι).Η λύση που προτείνεται εδώ είναι η χρήση Oracle (μαντείου).

Το Oracle είναι ένας server ο οποίος μπορεί να υπογράφει transaction αφότου αξιολογεί ότι η πρόταση που του έστειλαν είναι αληθής .Συγκεκριμένα η πρόταση εδώ είναι η διαθήκη του παππού που λέει:

if (has_died('john smith', born_on=1950/01/02))

return (10.0,1JxgRXEHBi86zYzHN2U4KMyRCg4LvwNUrp);

όπου **1JxgRXEHBi86zYzHN2U4KMyRCg4LvwNUrp** το bitcoin address του εγγονού και 10,το ποσό που επιθυμεί να του σταλεί.

Το Oracle αν αξιολογήσει ότι η πρόταση αυτή είναι αληθής (θα μπορούσε να είναι συνδεδεμένο σε μια σελίδα που ανακοινώνει τέτοια συμβάντα ,και να δει ότι είναι αληθές η ψευδές η να έψαχνε να κάνει match στο google κ.α),τότε θα υπογράψει το transaction που θα του έχει δοθεί.

Τώρα σχετικά με το πώς ο παππούς θα συντάξει το transaction του ,εδώ η συνθήκη εξαργύρωσης είναι η ακόλουθη:

<hash> OP_DROP 2 <sons pubkey> <oracle pubkey> CHECKMULTISIG

Όπου hash,το hash τις διαθήκης .Εδώ παρατηρούμε ότι μόλις τρέξουμε το script θα πετάξει από το stack το <hash>.Ο λόγος που το hash μπήκε είναι γιατί μόνο το Scriptpubkey θα δοθεί στο Oracle,έτσι θα πρέπει να γνωρίζει ότι όντως αυτή ήταν η διαθήκη του παππού (μπορεί να το ελέγξει απλά συγκρίνοντας τα hash).Επίσης με το Scriptpubkey που του παραχωρείται θα μπορεί να δει ότι όντως τα χρήματα στέλνονται στην διεύθυνση που λέει και η διαθήκη.(Θεωρητικά θα μπορούσε το transaction του παππού να μην υπήρχε καν ,καθώς το oracle το μόνο πράγμα που βλέπει είναι το Scriptpubkey,αλλά κάτι τέτοιο δεν μας απασχολεί καθώς μελετάμε ζητήματα ασφάλειας ,και έτσι αν υπέγραφε ένα transaction για το οποίο το id δεν υπήρχε απλά δεν θα το δέχονταν οι miners.)

Ο εγγονός έχει το transaction του παππού(δεν το έχει κάνει broadcast ακόμα),και το transaction που παίρνει εκείνος τα χρήματα από το πρώτο ,αλλά δεν μπορεί να το δημοσιεύσει καθώς λείπει μια υπογραφή ,αυτή του oracle.

Ο αλγόριθμος του oracle θα πήγαινε ως εξής:

1)Το oracle δέχεται το Scriptpubkey,την διαθήκη ,και ένα transaction που το μόνο που λείπει είναι η υπογραφή του oracle.

2)Ελέγχει αν το hash της διαθήκης ταυτίζεται με το hash του Scriptpubkey,αν όχι επιστρέφει false.

3)Στην συνέχεια ελέγχει ότι το address του transaction ταυτίζεται με αυτό στο return τις διαθήκης ,αν όχι επιστρέφει false.

4)Τέλος ελέγχει ότι η διαθήκη είναι αληθής ,αν ναι υπογράφει το transaction και επιστρέφει την υπογραφή του ,αλλιώς επιστρέφει false.

Έτσι ο εγγονός δημοσιεύει το transaction του παππού του και στην συνέχεια το δικό του.

2.4.3 Συνάλλαγμα

Σε αυτό το συμβόλαιο ας υποθέσουμε ότι παράλληλα με την αλυσίδα του bitcoin,έχει αναπτυχτεί και μια άλλη αλυσίδα ακριβώς με τον ίδιο τρόπο όπως του bitcoin,αλλά κρατώντας μια διαφορετική καταγραφή συναλλαγών (σαν να έχουμε ένα άλλο νόμισμα),αυτή την αλυσίδα θα την λέμε alternative chain.Το πρόβλημα εδώ είναι το εξής :Ο Α επιθυμεί να δώσει 10BC στον Β στο main chain,σε αντάλλαγμα 5 μονάδες του άλλου νομίσματος στο alternative chain.Το πρόβλημα εδώ είναι ότι ο Α έτσι και στείλει στον Β το συμφωνηθέν ποσό δεν ξέρει ότι ο Β θα του στείλει και αυτός με την σειρά του το αντίστοιχο στο alternative chain (η και το αντίθετο).Γι'αυτό έχουμε το παρακάτω συμβόλαιο:

1)Ο Α επιλέγει $x \in_R Z_p^*$

2)Ο Α δημιουργεί το Tx1 όπου στέλνει 10 BC σε όποιον του παρέχει δυο υπογραφές ,του Α και του Β η΄του Β και το μυστικό x (το οποίο εμφανίζεται μόνο το hash του),πιο συγκεκριμένα:

```
IF
  2 <key A> <key B> 2 CHECKMULTISIGVERIFY
ELSE
  <key B> CHECKSIGVERIFY SHA256 <hash of secret x> EQUALVERIFY
ENDIF
```

(εικόνα από **[12]**)

Στην συνέχεια φτιάχνει ένα δεύτερο transaction,το Tx2 όπου παίρνει πίσω αυτά τα χρήματα μετά από κάποιο χρονικό διάστημα θέτοντας το ntimelock στην επιθυμητή τιμή. Το στέλνει στον Β για να το υπογράψει ,στην συνέχεια το υπογράφει και ο Α και δημοσιεύει το Tx1 και Tx2.

Ο Β κάνει ακριβώς την ίδια δουλειά αλλά στο alternative chain,δηλαδή φτιάχνει το Tx1' με συνθήκη εξαργύρωσης την ίδια με προηγουμένως εκτός της δεύτερης συνθήκης που ζητάει την υπογραφή του Α και το μυστικό x.

Ο Α οποιαδήποτε στιγμή μπορεί να πάρει τα χρήματα στο alternative chain φανερώνοντας όμως το μυστικό x,έτσι ο Β θα μάθει το μυστικό και στην συνέχεια θα κατασκευάσει ένα transaction στο main chain στέλνοντας τα χρήματα στον εαυτό του (επικαλούμενος την δεύτερη συνθήκη εξαργύρωσης αφού πια ξέρει το x).Με αυτό τον τρόπο μπορούν δυο χρήστες να συναλλάζουν ποσά στα δυο chains με ασφάλεια.

3 ETHEREUM

Στην προηγούμενη ενότητα είδαμε έναν αποκεντρωμένο τρόπο πληρωμών όπως περιγράφεται αυτός στο σύστημα του bitcoin. Πέρα από πληρωμές στο σύστημα του bitcoin μπορούμε να φτιάξουμε και smart contracts (έξυπνα συμβόλαια),όπως και αναφέρθηκαν κάποια από αυτά, παρόλα αυτά είχαμε κάποιους περιορισμούς, αναφορικά**[21]**:

Blockchain Blindness:Το transaction στο bitcoin δεν μπορεί να αξιοποιήσει πληροφορίες από το block που θα μπει, όπως π.χ το hash του block,με αποτέλεσμα να στερούμαστε μια πολύ καλή πηγή τυχαιότητας (θα μπορούσαμε να το χρησιμοποιήσουμε σαν seed) για κάποια συμβόλαια (αν το hash(block) είναι μικρότερο του ε τότε το ποσό πάει στον Α, αλλιώς στον Β).

Value Blindness:Σε οποιαδήποτε συνθήκη εξαργύρωσης (δες ScriptPubKey σε προηγούμενες ενότητες) δεν έχουμε ικανοποιητικό έλεγχο πάνω στο ποσό που θέλουμε να πάρουμε, π.χ ισχύει ή όλα ή τίποτα (δηλαδή ή παίρνουμε ολόκληρο ένα output αν ικανοποιούμε το script,ή δεν το χρησιμοποιούμε καν).Ας φανταστούμε ότι υπάρχει ένα συμβόλαιο όπου ο Α μετά από ένα μήνα πρέπει να πάρει BC αξίας 100 $,όπως και με το συμβόλαιο με την διαθήκη σίγουρα θα χρειαστούμε και κάποιο oracle που να καθορίζει ποια είναι η ισοτιμία BC και $ έτσι ώστε να υπογράψει και να πάει το ποσό στον Α. Εδώ ένα εμφανές πρόβλημα είναι η κεντροποίηση αλλά αν το ξεπεράσουμε αυτό και πάλι για να γίνει αυτό το συμβόλαιο θα έπρεπε εξαρχής να φτιάχναμε όλα τα δυνατά ποσά που θα πάνε στο Α μετά το πέρας του μήνα (δεν ξέρουμε την ισοτιμία πριν ένα μήνα) διότι από ένα unspent output ή μπορούμε να το ξοδέψουμε όλο ή τίποτα. Έτσι πρέπει να δημιουργήσουμε όλα τα δυνατά ποσά , και στην συνέχεια το oracle να διαλέξει το σωστό για να το υπογράψει και να σταλεί στον Α, πράγμα που είναι ασύμφορο.

Lack of turing completeness:Όπως είχαμε αναφέρει στην προηγούμενη ενότητα, η γλώσσα που χρησιμοποιεί το bitcoin είναι non Turing complete.Σαν αποτέλεσμα έχουμε μια ¨ελαφριά¨ γλώσσα για το σύστημα, αλλά με κόστος τις περιορισμένες δυνατότητες ,π.χ για έναν επαναλαμβανόμενο υπολογισμό μπορεί να χρειαστεί να γράψουμε εκατοντάδες γραμμές κώδικα ενώ θα μπορούσαμε να κάναμε την ίδια δουλειά με ένα for.

Πέρα από αυτές τις δοκιμές αδυναμίες του bitcoin υπάρχουν και άλλες, όπως το ότι ο αλγόριθμος του proof of work που χρησιμοποιεί το bitcoin δεν είναι Asic resistant.Αυτό έχει σαν συνέπεια το mining να ελέγχεται από τα Asics τα οποία έχουν και υψηλό κόστος αλλά εκτός αυτού αποτελούν μορφή κεντροποίησης. Ένα επίσης από τα σημαντικότερα προβλήματα είναι η κεντροποίηση που υπάρχει στα mining pools.Καθώς ένας miner έχει πιο μεγάλο αναμενόμενο κέρδος να βγάλει αν συμμετέχει σε ένα poll (μικρό κέρδος, αλλά σταθερό),παρά να κάνει solo mining (μεγάλο κέρδος αλλά πολύ μικρή πιθανότητα),οι περισσότεροι miners σήμερα είναι polol miner.Αυτή οι miner συνδέονται μέσω ειδικού λογισμικού με τον admin του pool,των οποίο το υπόλοιπο δίκτυο του bitcoin τον αντιλαμβάνεται σαν έναν full node,ενώ οι υπόλοιποι miners του pool το δίκτυο τους αντιλαμβάνεται μόνο σαν cpu power(συγκεκριμένα hash power) του admin του pool(ουσιαστικά δεν υπάρχουν στο δίκτυο).Σαν συνέπεια έχει την πλήρη κεντροποίηση του miner από το poll,καθώς ο miner όχι μόνο δεν έχει το blockchain,αλλά δεν ξέρει τι κάνει mining,με αποτέλεσμα να αναθέτει όλη του την εμπιστοσύνη στον admin του pool (μη ξέροντας αν έχει καλούς η κακούς σκοπούς για το δίκτυο), η μόνη ύπαρξη που έχει για το δίκτυο είναι σαν light (spv) node (για αν ελέγχει αν πληρώθηκε από το mining,να δημιουργεί transaction κ.α).

Έτσι δημιουργείται η ανάγκη ένα νόμισμα που σε πρώτη φάση να μην έχει τις παραπάνω αδυναμίες, αλλά και το πιο βαθυστόχαστο, ένα νόμισμα που να μας επιτρέπει με ευκολία να μπορούμε να δημιουργούμε smart contracts με όσο το δυνατόν λιγότερους περιορισμούς.

3.1 STATE

Θα μπορούσαμε να φανταστούμε το σύστημα του bitcoin να περιγράφεται με μια κατάσταση **[21]**, την οποία μπορούμε να ονομάσουμε word state.Αυτή η κατάσταση θα λειτουργεί ως ένας λογαριασμός που θα μας λέει ποιος έχει τι. Πιο συγκεκριμένα για κάθε transaction,Tx1 και για κάθε state,s θα ορίζαμε μια συνάρτηση apply η οποία σαν είσοδο θα είχε το Tx1 και το s και θα μας έδινε το s'(apply(Tx1,s)->s').Το

s’ στο bitcoin θα είναι μια συλλογή από τα UTXO(unspent output).Πιο συγκεκριμένα η apply θα λειτουργεί ως εξής:

1)Ελεγξε αν τα UTXO που χρησιμοποιεί το Tx1 υπάρχουν στο s,και ότι πληρούνται οι συνθήκες εξαργύρωσης(π.χ υπάρχουν οι σωστές υπογραφές),αν ένα από τα δυο δεν ίσχυε επέστρεψε error

2)Ελεγξε αν το άθροισμα των ποσών που αναφέρονται στα inputs είναι μικρότερο από το άθροισμα των ποσών που αναφέρονται στα outputs,αν όχι επέστρεψε error.

3)Τέλος αφαίρεσε από το s όλα τα UTXO που χρησιμοποιήθηκαν στο input του Tx1 και προσέθεσε τα νέα UTXO (τα unspent output του Tx1) στο s(s’).

3.2 BLOCKCHAIN APPLICATIONS

Μετά την εμφάνιση του bitcoin **[16]**, έχουν εμφανιστεί πολλά νομίσματα χρησιμοποιώντας την ιδέα του blockchain όπως αυτή περιγράφεται στο σύστημα του bitcoin.Μια ιδέα είναι να χτίσουμε ένα διαφορετικό νόμισμα αλλά σε διαφορετικό blockchain,ανεξάρτητο από το bitcoin,όπως εχει γίνει με το namecoin ,πράγμα που εχει κόστος για το δίκτυο, δηλαδή οι miners πλέον θα δουλεύουν σε δυο ή και περισσότερα blockchains(ανάλογα με το πόσα τέτοια νομίσματα θα ακολουθήσουν) με αποτέλεσμα κάποια από αυτά να μην έχουν καν τον επαρκούμενο αριθμό miners για να συνεχίσουν την λειτουργία τους .Μια άλλη ιδέα για να γλιτώσουμε αυτό το κόστος θα ήταν να φτιάξουμε ένα νέο νόμισμα που να συνυπάρχει στο ίδιο blockchain με το bitcoin (όπως τα metacoins).Κάτι τέτοιο δεν θα είχε το κόστος της προηγούμενης περίπτωσης αλλά επειδή το block validation που γίνεται από τους miners του bitcoin δεν ελέγχει το data του ¨παρασιτικού¨ νομίσματος, δεν γνωρίζουμε κάτι για την εγκυρότητα του (π.χ ένα transaction αυτού του νομίσματος δεν ξέρουμε ότι είναι έγκυρο και ας υπάρχει στο block).Αυτό δημιουργεί πρόβλημα στους light spv nodes,διότι και να τους ¨αποδείξει¨ κάποιος full node ότι το transaction τους (μέσω merkle tree όπως είχαμε δει στην προηγούμενη ενότητα) ότι το transaction τους βρίσκεται στο block δεν γνωρίζουν κάτι για την εγκυρότητα του με αποτέλεσμα ή να χρειάζονται όλο το blockchain η κάποια έμπιστη αρχή.

3.3 MODIFIED MERKLE-PATRICIA TRIE

Σε αυτή την ενότητα θα αναφερθούμε στην λειτουργία και στην υλοποίηση του modified merkle patricia trie**[15],[17],[19],[20]**, που στις μετέπειτα ενότητες θα γίνει αντιληπτή η εφαρμογή του στο ethereum.

Αρχικά ,τι είναι ένα radix(patricia) trie?,είναι μια μορφή data structure(list) που σου επιτρέπει να αποθηκεύεις ζευγάρια της μορφής (key,value).Μπορούμε να το φανταστούμε σαν ένα δέντρο (πιο ρεαλιστικά σαν λίστα),όπου κάθε κορυφή είναι της μορφής [value, i0, i1 ... in],όπου value η τιμή που υπάρχει σε αυτή την κορυφή, και ij με j∈{Αλφαβητο key),η θέση που υπάρχει η αναμενόμενη κορυφή (στην συγκεκριμένη περίπτωση ένα reference τις κορυφής αυτής, ποιο συγκεκριμένα το hash της. Για το λόγο ότι η hash είναι μη αντιστρεπτή κάθε φορά που κάνουμε reference μέσα σε μια κορυφή αποθηκεύουμε σε ένα lookup table ζευγάρια της μορφής (node,hash(node)),έτσι ώστε σε περίπτωση αναζήτησης να μπορούμε να ανακτήσουμε τις κορυφές που γίνονται reference.

Ας υποθέσουμε ότι θέλουμε να βρούμε για το key=dog το value με το οποίο έχει αποθηκευτεί. Αρχικά θα μετατρέψουμε το dog **[20]** στο hex (δεκαεξαδικό σύστημα) που αντιστοιχεί στην τιμή 646f67,στην συνέχεια θα ανακτήσουμε την ρίζα του radix tree από την βάση δεδομένων μας, έστω η:

```
> a = memory_lookup(root)
[ NULL, NULL, NULL, 'af1259bb', '10bcf892', 'fe824bca', '725db325', '6b6c782e, '8924bce3', NULL,
NULL, NULL, NULL, NULL, NULL, NULL ]
```

Καθώς το πρώτο ψηφίο είναι το 6,μας ενδιαφέρει το περιεχόμενο τις θέσεις i6 όπως περιγράφεται και παραπάνω η δομή του node,όπου εκεί υπάρχει η τιμή 6b6c782e.Αυτή η τιμή είναι η αναφορά ή αλλιώς το δακτυλικό αποτύπωμα μιας άλλης κορυφής που πρέπει να μεταβούμε ώστε να πάρουμε την επόμενη πληροφορία μεταβάσης (κοιτώντας στην i4 θέση αυτή την φορά) και συνεχίζοντας μέχρι να εξαντλήσουμε το key μας, συγκεκριμένα:

```
> a['6']
'6b6c782e'
> b = memory_lookup(a['6'])
[ NULL, '18724bce', NULL, NULL , NULL, 'd8e7b12f', NULL, NULL, '87ac6db3', '8a71bcc2',
NULL, NULL, NULL, NULL, NULL, NULL, 'cef18742', 'ce82625a' ]
> b['4']
'd8e7b12f'
> c = memory_lookup(b['4'])
> c['6']
```

```
'789265b4'
> d = memory_lookup(c['6'])
[ NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
NULL, NULL, 'ab72cb25' ]
> e = memory_lookup(d['f'])
[ 'or do not there is no try', NULL, NULL, NULL, NULL, NULL, '345d78bf', NULL, NULL, NULL,
NULL, NULL, NULL, NULL, NULL, NULL ]
> f = memory_lookup(e['6'])
[ NULL, NULL, NULL, NULL, NULL, NULL, NULL, '12b302cc', NULL, NULL, NULL, NULL,
NULL, NULL, NULL, NULL ]
> g = memory_lookup(f['7'])
[ 'cat', NULL, NULL, NULL, NULL, NULL, NULL, '83b88608', NULL, NULL, NULL, NULL,
NULL, NULL, NULL, NULL, NULL ]
```

Όπως τελικά φαίνεται η τιμή ήταν cat!

Μια πρώτη παρατήρηση είναι ο χρόνος αναζήτησης είναι καλός (log(n)), αλλά θα μπορούσε να βελτιωθεί αν για παράδειγμα ανά ένα lookup περνάμε παραπάνω από ένα nibble πληροφορία, όπου αυτό ήταν εφικτό (π.χ για 3 συνεχόμενα βήματα ούτως αλλιώς υπήρχε μια πιθανή μετάβαση),βελτίωση που θα δούμε στο modified merkle-patricia trie.

Για να εισάγουμε μια κορυφή στο radix tree(trie)**[20]** , ανά nibble κλειδιού που διαβάζουμε μεταβαίνουμε στις κατάλληλες θέσεις ξεκινώντας από το root node μέχρι το κλειδί να εξαντληθεί. Στην συνέχεια διορθώνουμε το value (η το συμπληρώνουμε αν αυτό ήταν κενό),καθώς δημιουργούμε και nodes όπου αυτό είναι αναγκαίο (για παράδειγμα δεν προϋπήρχαν τόσο μεγάλα κλειδιά στο δέντρο).Τέλος επιστρέφοντας bottom(leaf node)-up(root node) διορθώνουμε τα references ή τα συμπληρώνουμε με τα νέα hashes,όπως ο αλγόριθμος περιγράφεται παρακάτω:

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

το κακό όπως φαίνεται(πάλι) είναι ότι μένουν ανεκμετάλλευτες πολλές nodes με αποτέλεσμα να χρειάζεται να αποθηκεύουμε πιο πολλές nodes στην βάση δεδομένων. Για παράδειγμα, αν αποθηκεύαμε μόνο ένα (key,value) pair θα χρειαζόταν να μεταβούμε αρκετά επίπεδα για την αποθήκευση του, και πιο πρακτικά ένα (key,value) πιο ‘’μακρύ’’ απ ότι υπήρχε στον δέντρο πριν. Παρόμοια αν θέλουμε να

απομακρύνουμε ένα κλειδί από το δέντρο μας **[20]** (και τα παιδιά του),αλγόριθμος είναι ο ακόλουθος:

```
def delete(node,key):
   if key == '' or node is NULL:
      return NULL
   else:
      curnode = memory_lookup(node)
      newnode = curnode.copy()
      newindex = delete(curnode[key[0]],key[1:])
      newnode[key[0]] = newindex
      if len(filter(x -> x is not NULL, newnode)) == 0:
         return NULL
      else:
         memory_save(hash(newnode),newnode)
         return hash(newnode)
```

3.3.1 HP(HEX PREFIX)

Στην συνέχεια θα αναφερθούμε στην κωδικοποίηση HP **[18]** που χρησιμοποιείται στο modified merkle patricia tree(trie).Ο λόγος είναι ότι καθώς θα ορίσουμε και νέα είδη κορυφών όπως θα δούμε στην συνέχεια για να αποφεύγουμε τις άσκοπες μεταβάσεις όπως αναφέραμε προηγουμένως, πρέπει να γνωρίζουμε πότε ένα key τελειώνει (για να αναζητούμε value) και πότε όχι (για να αναζητούμε reference,και στην συνέχεια να ανακτάμε την κορυφή από την βάση δεδομένων). Επίσης επειδή η διαδικασία κωδικοποίησης γίνεται αναδρομικά πρέπει να γνωρίζουμε αν έχουμε κωδικοποίηση για π.χ 0f48c4 η για το f48c4(δεν είναι το ίδιο, καθώς προηγούνται και άλλα nibbles), γι αυτό πρέπει να έχουμε σαν πληροφορία και το μήκος του string(αν είναι άρτιο η περιττό) εκτός από την τερματική του κατάσταση. Πιο συγκεκριμένα ο αλγόριθμος κωδικοποίησης είναι ο εξής:

```
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
```

Αν ο πίνακας που θα δεχτεί έχει το ειδικό τερματικό σύμβολο T (16),τότε στο nibble 0=(0,0,0,0),αλλάζει από το δεξιό το μέρος το περισσότερο σημαντικό από 0 ->1,δηλαδη (0,0,1,0).Στην συνέχεια το λιγότερο σημαντικό bit αν είναι άρτιο το αφήνει 0,αλλιως το κάνει 1,δηλαδη (0,0,1,1).Στην συνέχεια αν ήταν περιττό προσθέτουμε στον πινάκα αυτό το nibble (το καταχωρούμε από αριστερά θα ήταν σωστότερο),αλλιώς βάζουμε και το nibble 0,ετσι ώστε να διατηρήσουμε άρτιο το μήκος του ούτως ώστε στην εκτύπωση να μπορεί να αναπαρασταθεί από bytes.π.χ

```
Input: [ 1, 2, 3, 4, 5 ]
'\x11\x23\x45'
Input: [ 0, 1, 2, 3, 4, 5 ]
'\x00\x01\x23\x45'
Input: [ 0, 15, 1, 12, 11, 8, T ]
'\x20\x0f\x1c\xb8'
Input: [ 15, 1, 12, 11, 8, T ]
'\x3f\x1c\xb8'
```

3.3.2 RLP (Recursive Length Prefix) ENCODING

Πέρα από την HP που περιγράψαμε για το key,θα χρειαστούμε μια κωδικοποίηση ώστε να σειρικοποιήσουμε (serialized) τα δεδομένα μας(value,nodes),δηλαδή να μπορεί να διαβάσει ο υπολογιστής μια σειρά από bytes γνωρίζοντας τι πρόκειται να διαβάσει στην συνέχεια **[18]**. Σαν είσοδο στην rlp μπορεί να έχουμε ένα string (π.χ την λέξη 'dog'),μια λίστα(π.χ μια άλλη (key,value) κορυφή, η όπως θα δούμε στις παρακάτω ενότητες, ένα branch).

Η διαδικασία είναι η εξής:

1)Για ένα byte με τιμές από **[0x00, 0x7f],**τότε rlp(byte)=byte
2)Για ένα string με μήκος από 0-55,τοτε η κωδικοποίηση του ξεκινά με το byte **0x80 αθροίζοντας και το μήκος του string, ακολουθημένου(padding)από το ίδιο το string.Οι τιμές εδώ για το πρώτο byte κυμαίνονται στο διάστημα** [0x80, 0xb7].

3)Για string με μήκος μεγαλύτερα του 55,στην κωδικοποίηση τους το πρώτο byte είναι το **0xb7 αθροίζοντας το μήκος του μήκους του string στο δυαδικό(π.χ το string έχει μήκος 400 bytes ,το 400 σε hex είναι 190,δηλαδη \x01\x90,αρα μήκος 2),ακολουθημένου από τα byte που δείχνει το μήκος του(για το προηγούμενο**

παράδειγμα το \x01\x90)και στην συνέχεια ακολουθουμένου από το string.Το εύρος τιμών εδώ για το πρώτο byte είναι [0xb8, 0xbf](δηλαδή μπορούμε να κωδικοποιήσουμε μέχρι και string μήκους 256^7 bytes).

Στην περίπτωση που έχουμε λίστα (**π.χ** [key,value]), χρησιμοποιούμε αναδρομικά rlp σε κάθε αντικείμενο της λίστας (ωσότου δούμε byte η string) και στην συνέχεια κωδικοποιούμε την λίστα ως εξής:

4)Για λίστα όπου το μήκος της (μετρημένες και τα bytes της rlp) έχει μήκος από 0-55 bytes,στην κωδικοποίηση το πρώτο byte είναι το **0xc0 αθροίζοντας και το μήκος της λίστας, ακολουθουμένου από την λίστα. Το εύρος του byte εδώ είναι [0xc0, 0xf7].**

5)Για λίστα όπου το μήκος της είναι περισσότερο από 55 bytes,όπως και στην περίπτωση του string ,το πρώτο byte είναι το **0xf7 αθροίζοντας το μήκος του μήκους της λίστας στο δυαδικό, ακολουθουμένου από το byte που δείχνει το μήκος της λίστας, ακολουθουμένου από την λίστα. Το εύρος εδώ είναι [0xf8, 0xff]** (δηλαδή παρόμοια με τα string μπορούμε να κωδικοποιήσουμε λίστες έως 256^7 bytes).

Ο κώδικας για την κωδικοποίηση rlp είναι ο παρακάτω:

```
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)
```

Η συνάρτηση **isinstance** αξιολογεί αν η πρώτη καταχώρηση ταυτίζεται με το είδος που αναφέρουμε στην δεύτερη καταχώρηση, και επιστρέφει αναλόγως 0 η 1.Στην συνέχεια ελέγχει για το τι είδους κωδικοποίηση χρειάζεται (στην πρώτη περίπτωση αν έχουμε μήκος 1 και χαρακτήρα έως 128, όπως έχουμε περιγράψει απλά επιστρέφει τον χαρακτήρα),για τις άλλες περιπτώσεις θα χρειαστούμε και την συνάρτηση **encode_lenght**.

Η συνάρτηση **encode_lenght** ανεξαρτήτως τι είδος ήταν το input ελέγχει σε ποια από τις δυο κατηγορίες του ανήκει (μικρότερο από 55 bytes ή μεγαλύτερο),και ανάλογα επιστρέφει την κωδικοποίηση όπως αυτή περιγράφεται στις παραπάνω κατηγορίες. Αν το μήκος ξεπερνά το 256^7,επιστρεφει "input too long".

Τέλος ,η **to_binary**,δέχεται σαν είσοδο το μήκος του string (στο δεκαδικό),τον μετατρέπει σε bytes(το κάθε byte μπορεί να αναπαραστήσει 256 τιμές στο δεκαδικό),όπως ακριβώς μετατρέπουμε έναν δεκαδικό σε δυαδικό (αντί για διαδοχικές διαιρέσεις με το 2 και συμπλήρωση των υπόλοιπων από δεξιά, με το 256),και τον επιστρέφει.

Παράδειγμα

The string "dog" = [0x83, 'd', 'o', 'g']

The list ["cat", "dog"] = [0xc8, 0x83, 'c', 'a', 't', 0x83, 'd', 'o', 'g']

The empty string ('null') = [0x80]

The empty list = [0xc0]

The encoded integer 15 ('\x0f') = [0x0f]

The encoded integer 1024 ('\x04\x00') = [0x82, 0x04, 0x00]

The set theoretical representation of two, [[], [[]], [[], [[]]]] = [0xc7, 0xc0, 0xc1, 0xc0, 0xc3, 0xc0, 0xc1, 0xc0]

The string "Lorem ipsum dolor sit amet, consectetur adipisicing elit" = [0xb8, 0x38, 'L', 'o', 'r', 'e', 'm', ' ', ... , 'e', 'l', 'i', 't']

3.3.3 DESCRIPTION OF THE ALGORITHM

Τώρα μπορούμε να περιγράψουμε πώς λειτουργεί το modified merkle patricia tree(trie)**[22].**

Αρχικά η ιδέα πίσω από αυτό μοιάζει με του radix trie,αλλάζοντας βεβαία τα είδη κορυφών αυτού του δέντρου (εισάγοντας και κάποιες νέες κορυφές),και αυξάνοντας λίγο την πολυπλοκότητα για την δημιουργία του, τόσο λίγο που τα πλεονεκτήματα του είναι πολύ μεγαλύτερα, πριν μιλήσουμε γι αυτά ας δούμε τα είδη κορυφών που μπορεί αν έχει αυτό το δέντρο.

1)**Leaf node**. Αποθηκεύεται ως **(key,value),**με το key **να έχει** το ειδικό τερματικό σύμβολο όπως αναφέρθηκε στην hp κωδικοποίηση

2)**Extension node.**Αποθηκεύεται ως **(key,value),**με το key σε αυτή την περίπτωση **να μην εχει** το ειδικό τερματικό σύμβολο της hp,και το value να είναι απλώς μια αναφορά(hash),μια άλλης κορυφής(brance node),και το key να είναι κάποιο κοινό μέρος που έχουν δυο ή περιστέρα κλειδιά(common prefix),ή κάποιο key στο στάδιο του update ψάχνοντας το value του στην αναφερόμενη κορυφή (συγκεκριμένη στην $17^{η}$ θέση).

3)**Branch node.**Αποθηκεύεται ως **($item_1$,$item_2$,...,$item_{15}$,value),**με item να είναι η κάποιο άλλο είδος κορυφής ή reference αυτής(ανάλογα τι δείχνει η rlp κωδικοποίηση),και value είναι η το value ενός ζευγαριού που κάνουμε με αυτό update το δέντρο η μια αναφορά μιας άλλης κορυφής (αναλόγως αν εξαντλήθηκε κάποιο κλειδί στην προηγούμενη κορυφή πριν μεταβούμε εδώ).

Σαν είσοδο υποθέτουμε ότι έχουμε ζευγάρια της μορφής (key,value),πιο γενικά:

$$\mathfrak{I} = \{(\mathbf{k}_0 \in \mathbb{B}, \mathbf{v}_0 \in \mathbb{B}), (\mathbf{k}_1 \in \mathbb{B}, \mathbf{v}_1 \in \mathbb{B}), ...\}$$

(εικόνα από **[22]**)

Κάθε δυάδα (k_0,v_0),θα την αναφέρουμε σαν I (αναφερόμενη στο tube),και με I_0= k_0 και I_1= k_1.

Καθορίζουμε το δέντρο ως την παρακάτω συνάρτηση:

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

(εικόνα από **[22]**)

Όπως φαίνεται θα επιστρέψει το hash της ρίζας του δέντρου (που όπως θα δούμε/είδαμε και στο radix tree είναι κρυπτογραφικά ασφαλές, καθώς με οποιαδήποτε μελλοντική διαμόρφωση κάποιας κορυφής του δέντρου θα επέλθει και αλλαγή της ρίζας του (εκτός αμελητέας πιθανότητας) λόγω ότι κάθε κορυφή περιέχει κάποιο αποτύπωμα για τις επόμενες).

Η c(J,i) περιγράφεται όπως παρακάτω:

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

(εικόνα από **[22]**)

Όπου J κάθε φορά το εναπομένων σύνολο (θα φανεί καλυτέρα από την κλήση μέσω της n(J,i)), i το σημείο και έπειτα στα bytes του/των I_0 που μας ενδιαφέρει να διαβάσουμε.

Η c(J,i) ουσιαστικά βλέπει τι κορυφή πρέπει να φτιάξει αναλόγως, αν π.χ στο εναπομένων σύνολο J έχει ένα I ,τότε φτιάχνει μια leaf node(πρώτη περίπτωση),άνω αν υπάρχει κοινό μέρος μεταξύ όλων των κλειδιών (I_0 ,∀ I∈ J) (common prefix) φτιάχνει μια extension node,με στην θέση του key την hp κωδικοποίηση του common prefix (το true και false αναφέρονται για να ξέρει η hp αν υπάρχει terminal η όχι). Τέλος, στην τρίτη περίπτωση, αν δεν έχουμε ούτε ένα στοιχείο στο J,αλλά ούτε υπάρχει common prefix για όλα τα στοιχεία του, τότε φτιάχνει μια branch node που σε κάθε θέση καλή αναδρομικά την n(J΄,i) (όπως θα περιγράψουμε παρακάτω) με Jκάθε φόρα όλα εκείνα τα I∈ J,τέτοια ώστε το πρώτο nibble του key τους να είναι το αντίστοιχο ψηφίο σε hex ανάλογα με την θέση του branch.

Η n(J,i) περιγράφεται ως εξής:

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

(εικόνα από **[22]**)

Το n(J,i) κοιτάει κάθε φόρα αν το μήκος της κορυφής σε byte είναι μικρότερο από 32 bytes έτσι ώστε στο reference να το συμπεριλάβει όλο ή αν είναι μεγαλύτερο ώστε να συμπεριλάβει το hash της κορυφής (συγκεκριμένη το hash(rlp(node)) λαμβάνοντας υπόψη και τον ρόλο της c(J,i)).

Έτσι πέρα από την ρίζα του δέντρου πρέπει σε μια βάση δεδομένων να φυλάμε και τα ζευγάρια (hash(rlp(node)),rlp(node)) για ||node||>32.Για node μήκους μικρότερα από 32 bytes δεν χρειάζεται καθώς το είναι το ίδιο το node (rlp(node)).

Όπως καταλαβαίνουμε τα refrences φτιάχνονται bottom-up,όπως και στο radix tree.

Το σημαντικότερο πλεονέκτημα του σε σχέση με το radix tree είναι ο μηχανισμός του common prefix ,χωρίς να είναι απαραίτητο να κατεβαίνουμε άσκοπα πολλά επίπεδα στο δέντρο, έτσι ώστε για μια μόνο λέξη να χρειαζόμαστε αρκετά επίπεδα, κάνοντας και το search,το update,αλλά και το delete μη αποδοτικά.

Ας δούμε ένα παράδειγμα για το πώς λειτουργεί **[17]**.

Έστω τα ακόλουθα ζεύγη (key,value) που θέλουμε να συμπεριλάβουμε το δέντρο μας (ξεκινάμε από blank node):

('dog', 'puppy'), ('horse', 'stallion'), ('do', 'verb'), ('doge', 'coin')

μετατρέποντας τα keys σε hex βάζοντας και το ειδικό τερματικό σύμβολο T(16),περνούμε τον παρακάτω πίνακα:

[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'

Αρχικά λόγω του ότι το common prefix εδώ είναι 1 nibble (το 6), φτιάχνουμε ένα extension node για root,με key το hp_encode(6,no)=\x16,και value το reference τις επόμενης κορυφής (όπου θα χτιστεί bottom-up).

ROOT: ['\x16', A]

Στην συνέχεια λόγω του ότι δεν υπάρχει άλλο common prefix,φτιάχνουμε μια branch node, βάζοντας σε κάθε αντίστοιχη θέση του branch το reference της επόμενης κορυφής (ή την ίδια την κορυφή, ανάλογα το μήκος της),στην συγκεκριμένη περίπτωση στην θέση 4 και 8.

A: ['', '', '', '', B, '', '', '', C, '', '', '', '', '', '', '']

Φτιάχνοντας την κορυφή B (έχουμε ήδη διαβάσει 2 nibble, δηλαδή είμαστε για i=3),υπάρχει common prefix εδώ, το 6,15,αρα με την ίδια λογική με το root φτιάχνουμε την:
B: ['\x00\x6f', D]

Με παρόμοια λογική το δέντρο που προκύπτει είναι:

ROOT: ['\x16', A]
A: ['', '', '', '', B, '', '', '', C, '', '', '', '', '', '', '']
B: ['\x00\x6f', D]
D: ['', '', '', '', '', '', E, '', '', '', '', '', '', '', '', 'verb']
E: ['\x17', F]
F: ['', '', '', '', '', '', G, '', '', '', '', '', '', '', '', 'puppy']
G: ['\x35', 'coin']
C: ['\x20\x6f\x72\x73\x65', 'stallion']

Κανονικά το αποτέλεσμα που περνούμε από την συνάρτηση Trie(),είναι το sha(root).
Όπου A,B,C,D,E,F,G είναι οι κορυφές αυτές αν το μήκος σε bytes είναι μικρότερο από 32,ειδαλλως το hash αυτόν.(όπως αναφέρθηκε και παραπάνω, αν είναι το hash αυτόν, τότε έχουμε αποθηκευμένο και το ζεύγος (hash(rlp(node))),rlp(node)),για να την ανακτήσουμε). Λόγω απλότητας στο παραπάνω παράδειγμα, δεν αναφέραμε ότι κάθε value περνά και rlp κωδικοποίηση, π.χ "coin" = [0x84, 'c', 'o', 'i' , 'n'].
Θα μπορούσαμε να εισάγουμε και πιο σύνθετα αντικείμενα αντί για τα παραπάνω string,όπως θα γίνει κατανοητό και στην παράγραφο που θα περιγράψουμε τον τρόπο λειτουργίας του ethereum.

3.4 BlockChain

Στόχος στο ethereum **[22]** δεν είναι τόσο η ύπαρξη μιας εναλλακτικής οικονομίας όπως αυτή του bitcoin,αλλά όσο η ύπαρξη ενός «δικαστηρίου»(smart contracts) ούτως ώστε να υπάρχει εμπιστοσύνη .Συνεπώς το νόμισμα εδώ, ο κύριος ρόλος του είναι για να εξασφαλίζει το «καύσιμο» για την εκτέλεση τέτοιου διαδικασιών παρά για αγοραπωλησίες.

Αρχικά θα μπορούσαμε να σκεφτούμε τον κόσμο του ethereum σαν μια κατάσταση **σ,** όπου για κάθε transaction **T** μεταβάλλεται .Άρα θα χρειαστούμε να ορίσουμε μια συνάρτηση **Υ** τ.ω να πραγματοποιεί αυτή την μετάβαση ,δηλαδή:

$\sigma_{T+1}=Y(\sigma_T,T)$

Τώρα μετά από μια πεπερασμένη εφαρμογή της **Y** παίρνουμε την ημιτελική κατάσταση του κόσμου μας ,το μόνο που μας μένει είναι η εφαρμογή ενός ακόμα γεγονότος ,όπου αυτό είναι το mining reward (όπως στο bitcoin,to coinbase transaction).Αυτή την δουλειά θα την κάνει η συνάρτηση **Ω,** δηλαδή:

$\sigma_{final}=\Omega(B,Y(Y(\sigma,T_0),T_1),....)$, όπου **B** το block με ότι πληροφορίες περιέχει (θα αναφερθούμε αναλυτικότερα παρακάτω)

Συνοπτικά για όλη αυτή την διαδικασία πρέπει να ορίσουμε μια συνάρτηση **Π** τ.ω με είσοδο το **σ** και το block,**B,**να μπορεί να μας παρέχει την νέα κατάσταση του κόσμου.(παρακάτω θα δούμε αναλυτικότερα πως λειτουργούν αυτές οι συναρτήσεις).

$$\begin{aligned} \sigma_{t+1} &\equiv \Pi(\sigma_t, B) \\ B &\equiv (..., (T_0, T_1, ...)) \\ \Pi(\sigma, B) &\equiv \Omega(B, \Upsilon(\Upsilon(\sigma, T_0), T_1)...) \end{aligned}$$

(εικόνα από **[22]**)

VALUE

Στον κόσμο του ethereum,όπως και στο bitcoin πέρα από το νόμισμα μας που είναι το ether έχουμε και υποδιαιρέσεις αυτού ,με μικρότερη υποδιαίρεση το Wei.Ένα ether αντιστοιχεί σε 10^{18} Wei,και γενικότερα ισχύει ο παρακάτω πίνακας:

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

(εικόνα από **[22]**)

3.5 WORD STATE

Παραπάνω αναφερθήκαμε στην κατάσταση του κόσμου του ethereum χωρίς όμως να περιγράψουμε τι ακριβώς περιλαμβάνει αυτή.Η κατάσταση περιλαμβάνει λογαριασμούς που ελέγχονται είτε από πραγματικές οντότητες(simple η external account) είτε από τον ίδιο τον κώδικα τους(contract account).Αυτή η κατάσταση μπορεί να αποτυπωθεί μέσω του modified merkle patricia trie που είδαμε παραπάνω με key το address του account και value το περιεχόμενο του(όπου και αυτό θα υπόκειται σε κάποιο data structure).

Πιο συγκεκριμένα ένα account θα έχει τα εξής πεδία **[22]**(σίγουρα τα πρώτα δυο ,και πιθανόν τα τελευταία δυο ,είναι under development):

nonce:Ένας ακέραιος που χρησιμοποιείται ως μετρητής των transactions που στείλαμε από αυτό το account.Για ένα account **α** σε word state **σ**, αυτό θα συμβολίζεται με $\sigma[\alpha]_n$.

balance:Μια τιμή που θα μας δείχνει το πόσα Wei έχει αυτό το account,όπου θα τα συμβολίζουμε με $\sigma[\alpha]_b$.

stateRoot:Το αποτέλεσμα μιας hash-256(π.χ sha-256) του root node ενός data structure που περιλαμβάνει το storage του account (π.χ merkle tree) ,όπου θεωρείται ένας πίνακας με μεγέθους 32 που στην συνέχεια περνάει από RLP encode.Θα το συμβολίζουμε με $\sigma[\alpha]_s$.

codeHash:Το hash του κώδικα του account (αν είναι contract account) που εκτελείται από το EVM(ethereum virtual machine).Ο κώδικας αυτός ,όπως θα δούμε και στην συνέχεια ,εκτελείται αφού δεχτεί κάποιο μήνυμα(συνήθως),και ο κώδικας δεν μπορεί να αλλάξει(η τουλάχιστον μια τέτοια λειτουργία θα ήταν φανερή ,καθώς είναι public).Το hash αυτό το συμβολίζουμε με $\sigma[\alpha]_c$,και αν τον κώδικα τον συμβολίσουμε με **b**,τότε έχουμε sha3(**b**)= $\sigma[\alpha]_c$.

Σε περίπτωση που το πεδίο του codehash είναι κενό ,αυτό σημαίνει ότι αυτό το account είναι external.

Στην συνέχεια θα ορίσουμε μια συνάρτηση**[22]** η οποία σαν είσοδο θα δέχεται την κατάσταση **σ** και σαν έξοδο θα εμφανίζονται τα περιεχόμενα όλων των λογαριασμών που υπάρχουν ως τώρα στο world state,πιο φορμαλιστικά:

$$L_S(\sigma) \equiv \{p(a) : \sigma[a] \neq \varnothing\}$$ (εικόνα από **[22]**)

Όπου *p(a)*:

$$p(a) \equiv \left(a, \mathtt{RLP}\left((\sigma[a]_n, \sigma[a]_b, \sigma[a]_s, \sigma[a]_c)\right)\right)$$ (εικόνα από **[22]**)

Αυτή η συνάρτηση θα χρησιμοποιηθεί μαζί με το trie structure (merkle patricia trie) για να παραγάγει ένα αποτύπωμα του ethereum world state.

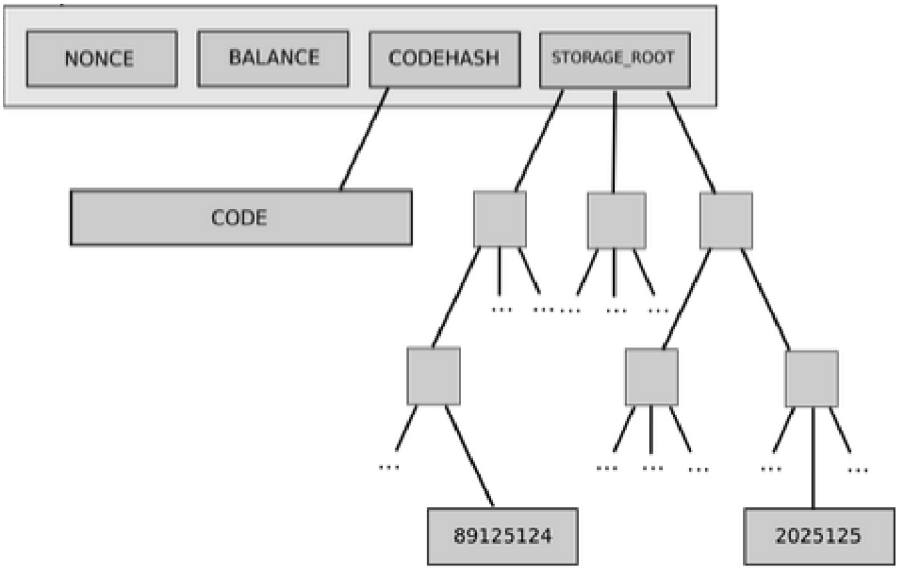
Συγκεκριμένα για κάθε address **α** ισχύει:

$$\forall a : \sigma[a] = \varnothing \ \vee \ (a \in \mathbb{B}_{20} \ \wedge \ v(\sigma[a]))$$ (εικόνα από **[22]**)

Όπου ν(σ[α]) είναι:

$$v(x) \quad \equiv \quad x_n \in \mathbb{P} \wedge x_b \in \mathbb{P} \wedge x_s \in \mathbb{B}_{32} \wedge x_c \in \mathbb{B}_{32}$$ (εικόνα από **[22]**)

Δηλαδή για κάθε address α, είτε που δεν υπάρχει στο state(ακόμα),είτε ¨υπάρχει¨, δηλαδή(ο ρόλος της συνάρτησης u που θα το αξιολογήσει αυτό) αν το address αυτό έχει integer στο nonce του account και έχει κάποιο balance (έστω και 0) και έχει κάποιο hash που κάνει point στο storage του και κάποιο hash του κώδικα του (αν είναι contract,αλλιώς κενό).
Συνοπτικά το account έχει την παρακάτω μορφή **[16]**:



(εικόνα από **[16]**)

3.6 TRANSACTION

Τα transaction στο ethereum **[21],[22]** είναι κάποιες οδηγίες με στοιχεία ταυτοποιήσης που είτε τα στέλνει μια πραγματική οντότητα (external actor) είτε κάποιο συμβόλαιο(contract account).Τα είδη τους είναι δυο :Τα κανονικά (message call) που μπορεί να είναι είτε μια απλή πληρωμή είτε κάποιο input για ένα contract account (που είδη υπάρχει),είτε contract creation transaction όπου σαν αποτέλεσμα έχει να δημιουργείται ένα contract με κώδικα που παράγεται από το transaction όπως θα δούμε και στην συνέχεια πιο αναλυτικά.

Τα κοινά πεδία και των δυο τύπων transaction είναι τα εξής:

nonce:Το πλήθος των transaction που έχουν σταλθεί από τον αποστολέα ,το συμβολίζουμε με $\mathbf{T_n}$.

gasPrice:Την τιμή για κάθε computational step που θα κάνει ο miner καθώς τρέχει το EVM,που είναι σε Wei(πιο αναλυτικά στην ενότητα mining), το συμβολίζουμε με $\mathbf{T_p}$.

gasLimit:Το μέγιστο ποσό που θα πληρωθεί ο miner για να τρέξει αυτό το transaction,το οποίο θα πληρωθεί απ΄την αρχή(πιο αναλυτικά στην ενότητα mining), το συμβολίζουμε με $\mathbf{T_g}$.

to:Μια 160-bit address που δείχνει τον παραλήπτης του transaction(κενό στην περίπτωση για contract creation),θα το συμβολίζουμε με $\mathbf{T_t}$.

value:Τα χρήματα σε Wei που είτε θα μεταφερθούν στον προορισμό τους είτε στην περίπτωση του contract creation θα αποτελούν κεφάλαιο του contract.θα το συμβολίζουμε με $\mathbf{T_v}$.

v,r,s:Τιμές που έχουν να κάνουν με την αυθεντικοποίηση του transaction,σχετικά με το ποιος ήταν ο αποστολέας και ποιος ο παραλήπτης ,θα τα συμβολίζουμε με $\mathbf{T_w}$, $\mathbf{T_r}$, και $\mathbf{T_s}$ (θα τα δούμε αναλυτικότερα παρακάτω).

Εκτός από τα παραπάνω που είναι κοινά ,τα contract creation transaction έχουν και τα εξής πεδία:

init:Είναι ένας πίνακας απεριορίστων διαστάσεων(fragment) όπου στην συνέχεια εκτελείται μέσω του EVM για την παραγωγή του κώδικα του account(main body).Το init εκτελείται μόνο μια φορά και στην συνέχεια γίνετε drop,καθώς έχουμε πλέον τον κώδικα του account όπου για κάθε μήνυμα που θα λάβει στο μέλλον θα ενεργοποιείται ή λόγω δικών του διαδικασιών (π.χ μετά από 10 blocks στείλε 10 ether στον Νίκο μαζί με το μήνυμα ‘’happy birthday’’). Θα το συμβολίζουμε με $\mathbf{T_i}$

Τέλος ένα απλού τύπου transaction (message call),περιέχει και το εξής:

data:Ένας πίνακας απεριορίστων διαστάσεων που έχει κάποια δεδομένα (π.χ το input για να εκτελέσει μια διαδικασία ένα contract account). Θα το συμβολίζουμε με $\mathbf{T_d}$

Τέλος ορίζουμε την συνάρτηση $\mathbf{L_T}$,η οποία δέχεται ένα transaction T, και το απεικονίζει σε ένα διάνυσμα αναλόγως την τιμή T_t (δηλαδή αν είναι contract creation η απλά message call),πιο φορμαλιστικά:

$$L_T(T) \equiv \begin{cases} (T_n, T_p, T_g, T_t, T_v, T_\mathbf{i}, T_w, T_r, T_s) & \text{if } T_t = 0 \\ (T_n, T_p, T_g, T_t, T_v, T_\mathbf{d}, T_w, T_r, T_s) & \text{otherwise} \end{cases}$$ (εικόνα από **[22]**)

Τα πεδία όπου ανήκουν τα επιμέρους στοιχεία είναι το ***N***, μετά από rlp encode,εκτός από το $\mathbf{T_i}$,και το $\mathbf{T_d}$ (που όπως αναφέραμε είναι unlimited size array) και το address $\mathbf{T_t}$.Πιο συγκεκριμένα:

$$\begin{array}{lllllll} T_n \in \mathbb{P} & \wedge & T_v \in \mathbb{P} & \wedge & T_p \in \mathbb{P} & \wedge \\ T_g \in \mathbb{P} & \wedge & T_w \in \mathbb{P} & \wedge & T_r \in \mathbb{P} & \wedge \\ T_s \in \mathbb{P} & \wedge & T_\mathbf{d} \in \mathbb{B} & \wedge & T_\mathbf{i} \in \mathbb{B} & \wedge \\ T_t \in \mathbb{B}_{20} & & & & & \end{array}$$ (εικόνα από **[22]**)

3.7 INSIDE THE BLOCK

Ένα block στο ethereum αποτελείται από 3 συνιστώσες **[22]** .Το block header **H,**κάποιες πληροφορίες σχετικές με τα transactions που βρίσκονται στο block **R,**και τέλος κάποια λίστα και από άλλα block header για τα οποία ο πατέρας αυτού του block και τα αναφερόμενα blocks είχαν κοινό πατέρα (τα λέμε uncle block,**U**).

Το block header αποτελείται από τα εξής πεδία:

parentHash:Όπως και στο bitcoin,το sha3 256 bit του πατέρα αυτού του block,θα το συμβολίζουμε με H_p.

unckeHash:Το sha3 256 bit τις λίστας των header των θείων ,θα το συμβολίζουμε με H_u.

coinbase:Η διεύθυνση αυτού που θα πληρωθεί για αυτό το block (του miner), θα το συμβολίζουμε με H_b.

stateRoot:Το sha3 256 bit της κορυφής του merkle patricia trie αφότου όλα τα transaction που περιέχονται σε αυτό το block εφαρμοστούν, θα το συμβολίζουμε με H_r.

transhactionTrie: Το sha3 256 bit της κορυφής του data structure που θα χρησιμοποιήσουμε για να κάνουμε insert τα transaction του block(το αντίστοιχο merkle root στο bitcoin), θα το συμβολίζουμε με H_t.

difficulty:Η δυσκολία του block(παρακάτω θα δούμε πως αυτή υπολογίζεται), θα το συμβολίζουμε με H_d.

number:Μια τιμή που μας δείχνει το πλήθος όλων τον προγόνων του block, θα το συμβολίζουμε με H_i.

minGasPrice:Μια τιμή που θα μας δείχνει την χαμηλότερη επιτρεπτή τιμή που μπορεί να πληρώνει ένα transaction για κάθε computational step(όπως θα αναφερθούμε για αποφυγή κάποιον επιθέσεων) για το οποίο πρέπει να ισχύει ούτως ώστε να συμπεριληφθεί στο block, θα το συμβολίζουμε με H_m.

gasLimit:Το συνολικό gas που επιτρέπεται να έχει καταναλωθεί σε αυτό το block(καθορίζεται από το ποια και πόσα transaction βάλαμε στο block αυτό), θα το συμβολίζουμε με H_l.

gasUsed:Πόσο τελικά gas χρησιμοποιήθηκε για όλα τα transaction που συμπεριλάβαμε στο block(φυσικά πρέπει να ισχύει gasUsed≤gasLimit), θα το συμβολίζουμε με H_u.

timestamp:Η ώρα σε unix,από την έναρξη mining αυτού του block(με πιθανή ανανέωση ανά κάποιο χρονικό διάστημα), θα το συμβολίζουμε με H_s.

extraData:Ένας τυχαίος πίνακας από bytes,που με εξαίρεση το genesis block πρέπει να μην ξεπερνά τα 32 bytes, θα το συμβολίζουμε με H_x.

nonce:Όπως και στο bitcoin,η τιμή που μας δείχνει πόσοι υπολογισμοί γίνανε στο block(pow).

Στην συνέχεια, η λίστα με τα uncle blocks **U** είναι απλά μια λίστα με block headers (όπως ακριβώς περιγράψαμε παραπάνω) τα οποία έχουμε συμπεριλάβει.

Τέλος το **R**, για κάθε transaction που έχουμε συμπεριλάβει στο block μας, αποτελείται από τρεις συνιστώσες(διάνυσμα):

$$R \equiv (R_T, R_{\sigma}, R_u)$$ (εικόνα από **[22]**)

R_T:Είναι το transaction

R_{σ}:Η κατάσταση του word state μετά από την εφαρμογή του αναφερόμενου transaction (hash κορυφής)

R_u:Το συνολικό used gas μετά την εφαρμογή του αναφερόμενου transaction (αυξητικά)

Τώρα για να κωδικοποιηθούν αυτά στο block πρέπει να χρησιμοποιήσουμε την RLP όπως περιγράφεται παραπάνω .Έτσι υπάρχει η ανάγκη για ορισμό μιας συνάρτησης ,τις $L_{\mathbf{R}}$, η οποία θα ετοιμάσει αυτό το διάνυσμα για να περάσει από κωδικοποίηση (καθώς η RLP αναγνωρίζει μόνο string ,λίστες από string και αναδρομικά τα αναφερόμενα).

Πιο φορμαλιστικά:

$$L_R(R) \equiv (L_T(R_T), \texttt{TRIE}(L_S(R_{\sigma})), R_u)$$ (εικόνα από **[22]**)

Για λόγους συμβολισμού(θα αναφερθούμε αργότερα) συμβολίζουμε την λίστα με τα transaction που κωδικοποιούνται στο block με :

$$B_{\mathbf{T}} \equiv (B_{\mathbf{R}}[0]_T, B_{\mathbf{R}}[1]_T, ...)$$ (εικόνα από **[22]**)

Παρατηρούμε ότι αυτή η λίστα δεν εμφανίζεται με αυτή την μορφή στο block.

Τώρα για την κωδικοποίηση του block ,θα χρειαστούμε την βοήθεια δυο ακόμα συναρτήσεων πριν περάσουν από RLP και αποθηκευτούν τοπικά στον υπολογιστή μας η γίνουν broadcast στο δίκτυο ,αυτές είναι οι L_H, και η L_B,για τα headers και τα transaction αντίστοιχα ,όπως περιγράφονται παρακάτω:

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

(εικόνα από **[22]**)

Όπου L^* είναι το διάνυσμα της συνάρτησης L,και τα πεδία τα οποία ανήκουν τα στοιχεία είναι:

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

(εικόνα από **[22]**)

3.8 BLOCK VALIDATION

Αρχικά θα ορίσουμε την δυσκολία του block ως εξής**[22]**:

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

(εικόνα από **[22]**)

Όπου για H_i=0 είναι το genesis block, και P(H) το block header του προγωνου(πατερα) του H.

Παρατηρουμε ότι αν το timestamp του block που θα γινει broadcast είναι μικροτερο από το timestamp του προγόνου του +42 (δηλαδη +42 sec),τοτε το difficulty θα μεγαλωσει,αλλιως θα μικρύνει,σε αντίθεση με το bitcoin οπου η αλλαγή difficulty γινεται κάθε 2016 blocks.

Στην συνέχεια θα ορίσουμε το gas limit του block,δηλαδή πόσο gas επιτρέπεται να γινει spent μέσω των transaction που φιλοξενεί για αν γινει αποδεκτό από το δίκτυο (computetional unit efford) ,ως εξής:

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

(εικόνα από **[22]**)

Όπου L΄(H):

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

(εικόνα από **[22]**)

Παρατηρούμε ότι το gas limit αλλάζει σύμφωνα με το limit και το gas used του προηγούμενου block με μεγαλύτερο συντελεστή στο limit.Έτσι αν χρησιμοποιείται πολύ λιγότερο gas απ'ότι το limit,το επόμενο limit θα τείνει να πέφτει (ποτέ κάτω από 125000),αλλιώς το αντίθετο.

Τέλος για το validation του block πρέπει να ισχύουν τα εξής:

$$\begin{aligned} H_u &\equiv \texttt{SHA3}(\texttt{RLP}(L^*_H(B_{\mathbf{U}}))) \quad \wedge \\ H_t &\equiv \texttt{TRIE}(\{\forall i < \|B_{\mathbf{T}}\|, i \in \mathbb{P} : p(i, L_R(B_{\mathbf{R}}[i])\})) \quad \wedge \\ H_r &\equiv \texttt{TRIE}(L_S(\Pi(\sigma, B))) \end{aligned}$$

(εικόνα από **[22]**)

(Δηλαδή η λίστα με τα uncle header που αναφέρονται να ταυτίζονται με το πεδίο H_u (αφότου είχαν ετοιμαστεί και στην συνέχεια περαστεί με RLP),το root του data structure με (**key**,**value**) ζευγάρια το index και την τριάδα που χαρακτηρίζει κάθε transaction αντίστοιχα να ταυτίζεται με το H_t,και το τέλος αν στην προηγούμενη κατάσταση κάνουμε apply μέσω της Π τα transaction να προκύπτει το ίδιο root hash,H_r,που αναφέρεται στο block.)

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

(εικόνα από **[22]**)

Όπου pow,η συνάρτηση proof of work που θα χρησιμοποιηθεί ,και το timestamp του block να είναι σίγουρα μεγαλύτερο από του προγόνου του.

3.9 TRANSACTION EXECUTION

Σε αυτή την ενότητα θα αναπτύξουμε τον τρόπο εκτέλεσης ενός transaction αναλόγως το είδος του (contract creation,message call)**[22],**καθώς και τα στάδια ενημέρωσης του word state.

Αρχικά θα αναφερθούμε ξανά στο gas limit και το gas price όπως ορίστηκαν παραπάνω.

Το **gas limit** είναι τα συνολικά computational units που έχουν καθοριστεί από τον sender για να φέρει ο miner εις πέρας αυτό το transaction(compute), δηλαδή για να το εκτελέσει στο EVM.Σε περίπτωση που το spent gas ξεπεράσει το limit,τότε το transaction και πάλι θεωρείται έγκυρο(περνάει το validation του block από τους άλλους miner),ο miner παίρνει την αμοιβή του και το execution γίνεται reverse στο σημείο που ξεκίνησε το transaction (το word state,**σ**, δεν ενημερώνεται εκτός κάποιον περιπτώσεων όπως π.χ κατά την εκτέλεση του init code όπως θα δούμε παρακάτω).

Τέλος το **gas price** καθορίζει πόσα ether θα είναι η αμοιβή του miner για κάθε computational unit.

Πριν αναφερθούμε στο transaction validation,θα ορίσουμε το εγγενές gas,g_0,οπως αυτό περιγράφεται παρακάτω:

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

(εικόνα από **[22]**)

όπου $\|T_i\|$ και $\|T_d\|$ είναι το πλήθος των bytes σε contract creation η σε message call transaction αντίστοιχα, και G κάποια fix fee value που υπολογίζεται σαν gas spent πριν ακόμα αρχίσει το execution (π.χ το G_{txdata}=1, $G_{transaction}$=500).

Τώρα το εγγενές validation περιλαμβάνει:

1)Οι υπογραφές του transaction είναι έγκυρες

2)Το nonce του transaction ισούνται με το nonce του sender account

3)Το gas limit είναι μεγαλύτερο από το εγγενές gas,g_0

4)Το account του sender έχει τουλάχιστον τόσα ether όσο το προπληρωμένο ποσό u_0,όπως καθορίζεται από την παρακάτω σχέση:

$$v_0 \equiv T_g T_p + T_v$$

(εικόνα από **[22]**)

Τέλος το validation είναι:

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

(εικόνα από **[22]**)

όπου αυτό σημαίνει ότι το address του sender να είναι συμπληρωμένο ,να υπάρχει στο word state,το nonce του transaction να ταυτίζεται με αυτό του sender,το εγγενές gas να είναι μικρότερο από το gas limit,να ισχύει το 4),και το άθροισμα του gas που έγινε spent γι'αυτό το transaction μαζί με τα άλλα που βρίσκονται στο block να μην ξεπερνούν το gas limit του block.

Ένα transaction που πληροί τα παραπάνω θεωρείται έγκυρο (ανεξαρτήτως αν το gas δεν είναι αρκετό για την περάτωση του).

Στην συνέχεια για όλα τα έγκυρα transactions,καθορίζουμε το checkpoint state σ_0 όπου δεν είναι τίποτε άλλο παρά ένα update του word state **σ**, αυξάνοντας το nonce του sender account κατά ένα ,και αφαιρώντας από το balance του το εγγενές κόστος,πιο φορμαλιστικά:

$$
\begin{aligned}
\sigma_0 &\equiv \sigma \quad \text{except:} \\
\sigma_0[S(T)]_b &\equiv \sigma[S(T)]_b - v_0 \\
\sigma_0[S(T)]_n &\equiv \sigma[S(T)]_n + 1
\end{aligned}
$$

(εικόνα από **[22]**)

Έπειτα θα ορίσουμε την μετά-προσωρινή κατάσταση ,σ_p ,η οποία αποτελεί ενημέρωση της σ_0 μετά το πέρας του transaction **T**.Οι συναρτήσεις που κάνουν αυτή την μετάβαση είναι οι Λ και Θ_3 για contract creation και message call αντίστοιχα .Πιο φορμαλιστικά:

$$
(\sigma_P, g', \mathbf{s}) \equiv \begin{cases} \Lambda(\sigma_0, S(T), T_o, g, T_p, T_v, T_\mathbf{i}) & \text{if} \quad T_t = 0 \\ \Theta_3(\sigma_0, S(T), T_o, T_t, g, T_p, T_v, T_\mathbf{d}) & \text{otherwise} \end{cases}
$$

(εικόνα από **[22]**)

Όπου g΄το remaining gas,και **s** η λίστα αυτοκτονίας (αναφέρεται σε account που δεν θα είναι πλέον ενεργά ,π.χ σε contract creation όπου μείναμε outofgas στην φάση εκτέλεσης του init code,οπότε αυτό το account μπήκε στην λίστα αυτοκτονίας),και $g=T_g-g_0$.

Στην συνέχεια θα ορίσουμε την ημιτελική κατάσταση ,σ*, στην οποία επιστρέφεται το remaining gas στο sender(αν υπάρχει),και προστίθενται στο balance του miner το used gas(το address που αναφέρεται στο πεδίο coinbase του block),πιο φορμαλιστικά:

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

(εικόνα από **[22]**)

Τέλος ορίζουμε την τελική κατάσταση σ΄, ως την κατάσταση όπου σβήνουμε από την σ* τα accounts που βρίσκονται στην λίστα αυτοκτονίας ,δηλαδή:

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

(εικόνα από **[22]**)

Τώρα μπορούμε να ορίσουμε ως Y^g το συνολικό gas που χρησιμοποιήθηκε για την εκτέλεση του Τ, ως εξής:

$Y^g(\sigma,T\alpha)=T_g-g'$

3.9.1 CONTRACT CREATION

Όπως αναφέρθηκε παραπάνω η συνάρτηση **Λ,** χρησιμοποιείται σε περίπτωση που το είδος του transaction είναι contract creation

$$(\sigma', g', \mathbf{s}) \equiv \Lambda(\sigma, s, o, g, p, v, \mathbf{i})$$ (εικόνα από **[22]**)

όπου σ η αρχική κατάσταση έχοντας αυξήσει το nonce κατά ένα ,και μειώνοντας το balance του sender όπως περιγράφεται παραπάνω ,s το address του sender,o το address του originator,g το gas,p το gasprice,u το κεφάλαιο του contract,και I το init code.

Αρχικά δημιουργείται το address του contract που δεν είναι παρά τίποτε άλλο από τα πιο δεξιά 160-bit του hash του ζευγαριού που αποτελείται από το address του sender και του nonce έχοντας περάσει rlp-encoding.Σε περίπτωση που το address χρησιμοποιείται αυξάνει κατά ένα μέχρι να βρει κάποιο διαθέσιμο:

$$\begin{aligned} A(s,n) &\equiv a \quad \text{where:} \\ a &= \arg\min_x : x \geqslant a' \wedge \sigma[x] = \varnothing \\ a' &= \mathcal{B}_{96..255}\Big(\texttt{SHA3}\Big(\texttt{RLP}\big(\ (s,n)\ \big)\Big)\Big) \end{aligned}$$ (εικόνα από **[22]**)

Παρατηρούμε ότι το nonce εδώ είναι το μειωμένο κατά ένα από το τρέχον ,λόγω του ότι από προηγούμενο βήμα έχει υποστεί προσαύξηση.

Στην συνέχεια πριν τρέξουμε στην evm το init code,θα ορίσουμε την κατάσταση σ*, στην οποία γίνεται initialization το contract account με μηδέν στην θέση nonce,u στο balance,TRIE(∅) στο storage,και SHA3( ) στο code:

$$\sigma^* \equiv \sigma \quad \text{except:}$$

$$\sigma^*[A(s, \sigma[s]_n - 1)] \equiv \big(0, v, \texttt{TRIE}(\varnothing), \texttt{SHA3}(())\big)$$ (εικόνα από **[22]**)

Τέλος μέσω της συνάρτησης **Ξ**, υπολογίζουμε το body του code από το init code.Αν κατά αυτή την διαδικασία ξεμείνουμε από gas,τότε λέμε ότι μια OutOfGas exception συνέβη κατά την εκτέλεση (και έτσι δεν είχαμε normal halting).Σαν συνέπεια έχουμε το gas που επιστρέφεται να είναι μηδέν (g΄=0,δηλαδη ο miner πληρώνεται),και η κατάσταση επιστρέφει πριν την δημιουργία του contract(δηλαδή το address του

contract μπαίνει στην λίστα αυτοκτονίας).Σε οποιαδήποτε άλλη περίπτωση το remaining gas επιστρέφεται στον sender,και το πεδίο code του account ενημερώνεται:

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

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

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

(εικόνα από **[22]**)

3.9.2 MESSAGE CALL

Εκτός από τα contract creation transaction υπάρχουν και τα message call.Σε αυτήν την κατηγορία ανήκουν απλά transaction(όπως στο bitcoin) που απλά μεταφέρουν ether από το ένα account στο άλλο,η transaction που σαν συνέπεια έχουμε την εκτέλεση κάποιου κώδικα(π.χ αν κάποιο transaction έχει input data και παραλήπτης είναι κάποιο contract account,θα εκτελεστεί ο κώδικας του).Μια extra παράμετρος που έχουμε εδώ σε σχέση με τα contract creation(output της συνάρτησης **Λ**) είναι ο output code (**o**) που στην περίπτωση ενός απλού transaction δεν λαμβάνεται υπόψη ,συγκεκριμένα:

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

(εικόνα από **[22]**)

Ορίζουμε την κατάσταση σ_1 να είναι η κατάσταση πριν ακριβώς από το code execution (δηλαδή να έχουμε μεταφέρει απλά τα ether από το ένα account στο άλλο):

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

(εικόνα από **[22]**)

Στην συνέχεια χρησιμοποιώντας την συνάρτηση **Ξ**(code execution) παίρνουμε την κατάσταση **σ**** .Αν δεν είχαμε normal halting κατά την διάρκεια εκτέλεσης του κώδικα (π.χ out of gas,αντί για return),τότε το state που επιστρέφει η **Ξ** είναι Ø και η νεα κατασταση είναι ιδια με την **σ_1**(δηλαδή επιστρέφουμε αμέσως πριν την εκτέλεση

του κώδικα),αλλιώς κοιτάμε την **σ**** για recent state(δηλαδή την κατάσταση που έχουμε αφότου έχει εκτελεστεί ο κώδικας):

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

(εικόνα από **[22]**)

Τέλος το Ethereum Virtual Machine(EVM) **[21]**,αποτελείται από ένα διάνυσμα που περιέχει την κατάσταση του block που κάνουμε mining,το transaction,τον κώδικα που πρόκειται να τρέξουμε αν απευθυνόμαστε σε contract account,την προσωρινή μνήμη memory,το stack που δεν είναι τίποτε άλλο παρά ένας LIFO σωρός ,τον μετρητή **pc** που μας δείχνει σε ποιο computational step είμαστε στον κώδικα και τέλος το διαθέσιμο gas:

(block_state, transaction, message, code, memory, stack, pc, gas)

3.10 ETHEREUM CONSESUS

Θα μπορούσαμε να φανταστούμε τα block που γίνονται broadcast στο δίκτυο με reference σε κάποιο προηγούμενο block ότι τελικά η εικόνα που θα έχουμε θα είναι ένα δέντρο με ρίζα το genesis block και φύλλα τα τελευταία blocks που έχουν γίνει mining,αλυσίδα ονομάζουμε οποιοδήποτε μονοπάτι από την ρίζα σε οποιοδήποτε φύλλο .Το ζήτημα εδώ είναι ,ποια αλυσίδα είναι η σωστή?

Στο bitcoin η πιο «βαριά» αλυσίδα είναι και η σωστή **[3]** ,δηλαδή η πιο μακριά ,αυτή που απεικονίζει το μεγαλύτερο proof of work.Στο ethereum **[21],[22]** χρησιμοποιείται μια παραλλαγή του GHOST,δηλαδή ακολουθούμε το πιο βαρύ block αλλά στο difficulty προσμετράμε και τα uncle blocks,blocks τα οποία στο bitcoin είναι απλά stale blocks.Με αυτό τον τρόπο δεν πάει χαμένο το work που έχει γίνει σε αυτά τα

blocks και επίσης ενισχύουμε και την ασφάλεια του main chain.Η σχέση που μας δίνει το work του chain δίνεται παρακάτω:

$$B_t \equiv B'_t + B_d + \sum_{U \in B_U} U_d$$
$$B' \equiv P(B_H)$$

(εικόνα από **[22]**)

όπου B_t είναι το ολικό difficulty , B_d το block difficulty, B'_t του parent block το ολικό difficulty,και B_u η λίστα με τα difficulties των uncle blocks.

3.11 MINING REWARD AND BLOCK FINALIZATION

Ολοκληρώνοντας, η αμοιβή για ένα έγκυρο block δύνεται στο coin base address του block και ενημερώνεται η κατάσταση **σ** μέσω της συνάρτησης **Ω,** όπως φαίνεται παρακάτω:

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

(εικόνα από **[22]**)

οπου R_b=1500 Finney

Με αυτό τον τρόπο δίνεται κίνητρο και στον miner να συμπεριλάβει uncle blocks,καθώς το reward του αυξάνεται (ανά 8 uncle blocks διπλασιάζεται),αλλά και το computational effort των uncle blocks δεν πάει χαμένο καθώς αμείβονται αν κάποιο block τα αναφέρει.

4 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]Elliptic_Curve_Digital_Signature_Algorithm,
" http://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm "

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

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

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

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

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

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

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

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

[23] Christian Hanser and Daniel Slamanig, "Blank Digital Signatures", 2013

[24] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg, "Constant-Size Commitments to Polynomials and Their Applications" , 2010

[25] Joachim Von Zur Gathen and Daniel Panario,"Factoring Polynomials Over Finite Fields: A Survey",2001

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

[27] Α.Κιαγιάς: Τεχνικές Σύγχρονης Κρυπτογραφίας