ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΘΕΣΣΑΛΟΝΙΚΗΣ

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Πτυχιακή Εργασία

# Ανάπτυξη smart contracts στο περιβάλλον Ethereum

Μαντελάκης Χαρίτων

*Επιβλέπων:*
Μητράκος Δημήτριος
Καθηγητής

Θεσσαλονίκη 2018

# Περιεχόμενα

**Περίληψη**

Το κείμενο αυτό σκοπεύει να διερευνήσει τις βασικές αρχές της ανάπτυξης smart contracts και dapps καθώς και τα θέματα στα οποία πρέπει να δωθεί προσοχή κατα τη διάρκεια της ανάπτυξης, όπως η ασφάλεια και η ελαχιστοποίηση του κόστους λειτουργίας. Θα αναλυθούν σε μεγαλύτερο βάθος τεχνικά στοιχεία του ethereum και το πως αυτά δημιουργούν νέες δυνατότητες αλλά και απαιτήσεις στο front-end και back-end των εφαρμογών. Ταυτόχρονα θα αναπτυχθεί ως παράδειγμα μια εφαρμογή η οποία θα χρησιμοποιεί smart contracts για να υλοποιήσει μια υπηρεσία χρηματοδότησης από κοινού (crowdfunding) σε φιλανθρωπικούς σκοπούς η οποία θα εξαλείφει την ανάγκη ύπαρξης εμπιστοσύνης προς την φιλανθρωπική εταιρία ( trustless) . Η εφαρμογή θα σχεδιαστεί αρχικά σύμφωνα με τις αρχές τεχνολογίας λογισμικού, το οποίο θα χρησιμεύσει για να δείξουμε πως η ανάπτυξη σε Ethereum στηρίζεται αλλά ταυτόχρονα διαφέρει από τις παραδοσιακές μεθόδους. Τέλος, θα γίνει μια θεωρητική μελέτη στο πως η εφαρμογή θα μπορούσε να κινηθεί ώστε να επιτύχει μαζική αποδοχή, θα εξερευνήσουμε το πως θα υλοποιούνταν μια αρχική κατανομή νομισμάτων ( initial coin offering - ICO ) και στο πως θα μπορούσε να εξελιχθεί σε έναν αυτόνομο οργανισμό με την προσθήκη ενός επιχειρηματικού σχεδίου στον κώδικα της.

# Κεφάλαιο 1

# Εισαγωγή

## 1.1

Ένας καινούριος και ταχέως αναπτυσσόμενος τομέας εφαρμογών του διαδικτύου είναι αυτός των ηλεκτρονικών ή ψηφιακών νομισμάτων ( digital currencies or cryptocurrencies ). Με αρχή τη δημιουργία του bitcoin [?] το 2009, θεμελιώθηκαν οι βασικές τεχνολογίες που επιτρέπουν την δημιουργία αποκεντρωμένων ψηφιακών νομισμάτων και μαζί με αυτές ιδρύθηκαν χιλιάδες ακόμα ψηφιακά νομίσματα. Τα ψηφιακά νομίσματα είναι αποκεντρωμένα καθώς η δημιουργία τους δεν γίνεται από κάποια αρμόδια αρχή ή τράπεζα, αλλά από ένα λογισμικό που τρέχει ταυτόχρονα σε μεγάλο αριθμό υπολογιστών σε όλο τον κόσμο, οι οποίοι συμφωνούν στις λειτουργίες δημιουργίας, μεταφοράς και σε μερικές περιπτώσεις, καταστροφής των νομισμάτων.

Οι κανόνες στους οποίους συμφωνεί αυτό το διανεμημένο και αποκεντρωμένο σύστημα συνοψίζονται σε αυτό που σήμερα αποκαλείται τεχνολογίες blockchain. Καθώς όλος αυτός ο χώρος είναι ακόμα σε πρώιμο στάδιο, διάφορες ονομασίες δημιουργούνται, χρησιμοποιούνται και εγκαταλείπονται καθημερινά ώστε να δώσουν τόπο σε νέες, οπότε η επικαιρότητα κάποιων όρων, παρά τις καλύτερες προσπάθειες του συγγραφέα, δεν είναι εγγυημένη. Το bitcoin επέτρεψε την αποθήκευση, δημιουργία και μεταφορά χρημάτων μεταξύ ατόμων χωρίς τη χρήση ενδιάμεσου διαμεσολαβητή μέσω μιας λίστας ( ledger ) η οποία αντιστοιχεί διευθύνσεις (addresses) και ποσά, στην μορφή της οποίας, μέσω της τεχνολογίας blockchain συμφωνεί ανα πάσα στιγμή όλο το δίκτυο. Αυτή η ομάδα λειτουργειών ονομάστηκε στη συνέχεια blockchain 1.0 από την κοινότητα [?].

Στην συνέχεια, οι πρώτοι χρήστες αυτής της τεχνολογίας αντιλήφθηκαν ότι τα ποσά που αναγράφονται στη λίστα δεν είναι απαραίτητο να αντιστοιχούν σε χρηματική αξία , αλλά σε οτιδήποτε αντικείμενο ή πληροφορία θα μπορούσε να κατέχει κάποιος ( asset ) . Τα ψηφιακά νομίσματα που αναπτύχθηκαν με τέτοιου είδους λειτουργίες κατατάχθηκαν μετέπειτα στην κατηγορία blockchain 2.0 . Ένα χαρακτηριστικό παράδειγμα είναι το namecoin [?] το οποίο αντιστοιχεί διευθύνσεις διαδικτύου ( domain names ) μονοσήμαντα σε χρήστες, και επιτρέπει την μεταφορά τους μεταξύ αυτών. Ο χώρος των κρυπτονομισμάτων πέρασε λοιπόν σε μια περίοδο οπου κάθε νόμισμα που δημιουργούταν είχε και από μία ξεχωριστή λειτουργία. Καθώς όμως δεν υπήρχε κάποια διασυνδεσιμότητα μεταξύ τους, σε συνδυασμό με τις ήδη υπάρχουσες προκλήσεις της τεχνολογίας λόγω των προχωρημένων μεθόδων κρυπτογραφίας που χρησιμοποιεί, δημιουργήθηκε ένα μεγάλο ανάχωμα στην μαζική υιοθέτηση των ψηφιακών νομισμάτων.

Την λύση ήρθε να δώσει το 2014 η πλατφόρμα Ethereum [?], η οποία επέτρεψε την δημιουργία νομισμάτων τύπου blockchain 2.0 , με οποιαδήποτε λειτουργία ήθελε ο κατασκευαστής για το καθένα και τα οποία μπορούν να αλληλεπιδράσουν μεταξύ τους μέσω ενός τοπικού ψηφιακού νομίσματος, το ether , που λειτουργεί ως το τέλος το οποίο χρεώνονται οι αποκεντρωμένες αυτές εφαρμογές για να τρέξουν στην πλατφόρμα. Εκτός των άλλων, η πλατφόρμα Ethereum προσφέρει την πλήρη ασφάλεια και την αποκεντρωμένη λειτουργία της τεχνολογίας blockchain χωρίς να απαιτεί από τον προγραμματιστή προχωρημένες γνώσεις κρυπτογραφίας,καθώς και μια δική της , turing-complete γλώσσα ανάπτυξης και βελτιωμένη ταχύτητα λειτουργίας.

Ο συνδυασμός κώδικα και χρηματικών συναλλαγών δημιούργησε την έννοια των έξυπνων συμβολαίων ( smart contracts ), προγραμμάτων δηλαδή, τα οποία μετά από την εκπλήρωση κάποιων συνθηκών στέλνουν ή λαμβάνουν χρήματα. Εφαρμογές οι οποίες χρησιμοποιούν smart contracts για τις λειτουργίες τους ώστε να επιτύχουν έναν σκοπό ή να προσφέρουν μια υπηρεσία ονομάστηκαν dapps από το decentralized applications ( αποκεντρωμένες εφαρμογές) [?].

Μέσω του internet of things, οι υπηρεσίες αυτές μπορούν να περάσουν από τον κόσμο του διαδικτύου στον πραγματικό κόσμο. Μέσα στις δυνατότητες για παράδειγμα, συμπεριλαμβάνονται αυτο-οδηγούμενα αμάξια τα οποία θα μπορούν να εκτελούν μετακινήσεις μεταφορών μέσω υπηρεσιών όπως uber, θα πληρώνονται με ψηφιακά νομίσματα, θα πληρώνουν μόνα τους τα έξοδα μετακίνησης, καθαρισμού και συντήρησης και θα μοιράζουν τα όποια κέρδη τους στους μετόχους τους ( κάτοχοι και αυτοί κάποιου ειδικού νομίσματος το οποίο θα αντιπροσωπεύει ένα μέρισμα του αυτοκινήτου ) με κάποιο επιχειρηματικό μοντέλο το οποίο θα είναι ενσωματωμένο στον κώδικα λειτουργίας. Επιχειρήσεις που εν δυνάμει θα μπορούσαν να λειτουργήσουν με αυτό τον τρόπο ονομάστηκαν Daos από το dis-

tributed autonomous organizations ( διανεμημένοι αυτόνομοι οργανισμοί ).

Σύντομα μετά τη δημιουργία της λοιπόν, η πλατφόρμα Ethereum έγινε ένα δυναμικό οικοσύστημα δημιουργίας εφαρμογών [?] και οδήγησε την ανάπτυξη μιας πληθώρας νέων προγραμματιστικών τεχνικών για τη δημιουργία ασφαλών, λειτουργικών και αποτελεσματικών smart contracts. Η παρούσα πτυχιακή εργασία έχει στόχο να εξερευνήσει τις πιο πρόσφατες και λειτουργικές από αυτές παράλληλα με τη δημιουργία μιας dapp φιλανθρωπικού σκοπού.

Με τη ραγδαία ανάπτυξη που έχει παρατηρηθεί και στον χώρο των ψηφιακών νομισμάτων γενικότερα και στο Ethereum συγκεκριμένα, οι γνώσεις ανάπτυξης smart contracts είναι ήδη περιζήτητες από τη νεοδημιούργητη αυτή αγορά εργασίας.

## 1.2 Δομή διπλωματικής εργασίας

- Στο δεύτερο κεφάλαιο της εργασίας θα γίνει μια αναλυτική παρουσίαση της τεχνολογίας blockchain που υποστηρίζει το bitcoin, ώστε να αναφερθούν οι βασικές έννοιες και τεχνικές λειτουργίας και ασφάλειας των κρυπτονομισμάτων.
- Στο τρίτο κεφάλαιο της εργασίας θα εμφανιστούν μερικές πιο τεχνικές πτυχές του Ethereum και θα επεξηγηθεί μέσω παραδειγμάτων το πως μια κλασσική εφαρμογή υλοποιείται σε αυτό. Επίσης θα γίνει μια συνοπτική παρουσίαση των πιο δημοφιλών και χρηστικών εργαλείων ανάπτυξης smart contracts.
- Στο τέταρτο κεφάλαιο θα εξερευνηθούν σε βάθος μερικά από τα πιο σημαντικά ζητήματα ανάπτυξης εφαρμογών με έμφαση στην ασφάλεια.
- Στο πέμπτο κεφάλαιο θα γίνει μια εκτενέστερη περιγραφή της εφαρμογής που πρόκειται να αναπτυχθεί και θα γίνει η αρχική σχεδίαση της με βάση τις αρχές τεχνολογίας λογισμικού.
- Στο έκτο κεφάλαιο θα γίνει η περιγραφή της ανάπτυξης της εφαρμογής που σχεδιάστηκε στο τέταρτο και η επεξήγηση των ουσιωδών σημείων του κώδικα της.
- Στο έβδομο κεφάλαιο θα γίνει η τελική παρουσίαση της λειτουργίας της εφαρμογής και θα αναλυθούν μελλοντικές επεκτάσεις οι οποίες θα της επέτρεπαν να επιτύχει σε μεγαλύτερο βαθμό το σκοπό της.
- Στο όγδοο κεφάλαιο θα ενσωματωθούν τα συμπεράσματα από την ανάπτυξη.

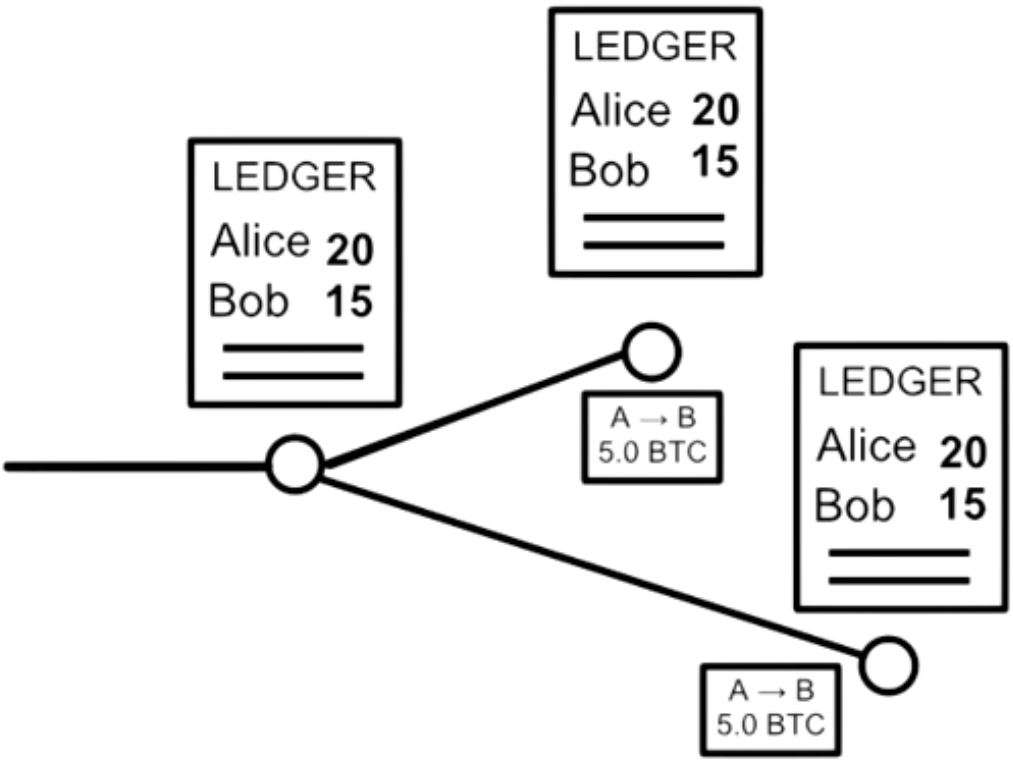
# Κεφάλαιο 2

# Πασουσίαση του BlockChain και των Κρυπτονομισμάτων

## 2.1 Τι είναι τα κρυπτονομίσματα

Στον πυρήνα τους, τα κρυπτονομίσματα είναι ένα ψηφιακό αρχείο που καταγράφει λογαριασμούς και ποσά σε λογιστικό επίπεδο. Ένα αντίγραφο αυτού του αρχείου διατηρείται σε κάθε υπολογιστή που υποστηρίζει το δίκτυο. Να τονιστεί ότι δεν είναι απαραίτητη η κατοχή αυτού του αρχείου για τις συναλλαγές. Αυτά τα νούμερα δεν αντιπροσωπεύουν στοιχεία του πραγματικού κόσμου, αλλά έχουν χρηματική αξία επειδή οι άνθρωποι είναι διατεθειμένοι να εκτελούν συναλλαγές για πραγματικά προϊόντα και υπηρεσίες με αντάλλαγμα την προσθήκη ενός ποσού στο λογαριασμό τους. Με την λογική ότι και οι υπόλοιποι χρήστες του δικτύου θα επιδιώξουν να εκτελέσουν τέτοιες συναλλαγές. Τα νούμερα αυτά έχουν αξία επειδή οι άνθρωποι πιστεύουν ότι έχουν αξία, οπως τα fiat νομισματα.

Για την αποστολή χρημάτων, αναγγέλουμε στο δίκτυο ότι το ποσό στο λογαριασμό μας θα πρέπει να μειωθεί, και το ποσό του παραλαμβάνοντος θα πρέπει να αυξηθεί. Οι κόμβοι, υπολογιστές, στο δίκτυο του κρυπτονομίσματος εφαρμόζουν την συναλλαγή στο αντίγραφο του αρχείου τους, και μεταβιβάζουν την συναλλαγή στους επόμενους κόμβους. Αυτή η επικοινωνία, σε συνδυασμό με μεθόδους κρυπτογραφίας, είναι όλο το υποσύστημα του κρυπτονομίσματος. Ένα σύστημα που επιτρέπει σε ένα σύνολο υπολογιστών να διατηρήσουν ένα αρχείο συναλλαγών.




Σχήμα 2.1: Μετάδοση συναλλαγής και ενημέρωση λίστας

Ακούγεται παρόμοιο με τη μέθοδο που μια τράπεζα διατηρεί ένα αρχείο με λογαριασμούς,αλλά διαφέρει στο γεγονός ότι αυτό το αρχείο διατηρείται από ένα σύνολο οντοτήτων σε αντίθεση με μια μοναδική οντότητα. Αυτό το γεγονός εισάγει μερικές πολύ σημαντικές διαφορές. Σε αντίθεση με μια τράπεζα στην οποια γνωρίζουμε μόνο τις δικές μας συναλλαγές, στα κρυπτονομίσματα έχουμε την δυνατότητα να γνωρίζουμε τις συναλλαγές κάθε λογαριασμού στο δίκτυο. Επίσης, με τις τραπεζες υπάρχει ένα επίπεδο εμπιστοσύνης για τις συναλλαγές και μπορούν να γίνουν νομικές κινήσεις εναντίων τους σε περιπτώσεις λάθους ή απάτης. Στα κρυπτονομίσματα οι συναλλαγές είναι μεταξύ αγνώστων, συνεπώς δεν υπάρχει κανένα επίπεδο εμπιστοσύνης. Το σύστημα των κρυπτονομισμάτων είναι σχεδιασμένο με αυτό τον τρόπο ώστε να μην χρειάζεται να υπάρχει εμπιστοσύνη μεταξύ των χρηστών. Ειδικές μαθηματικές συναρτήσεις προστατεύουν καθε πλευρα της συναλλαγής.

## 2.2 Αποστολή χρημάτων μέσω κρυπτονομισμάτων

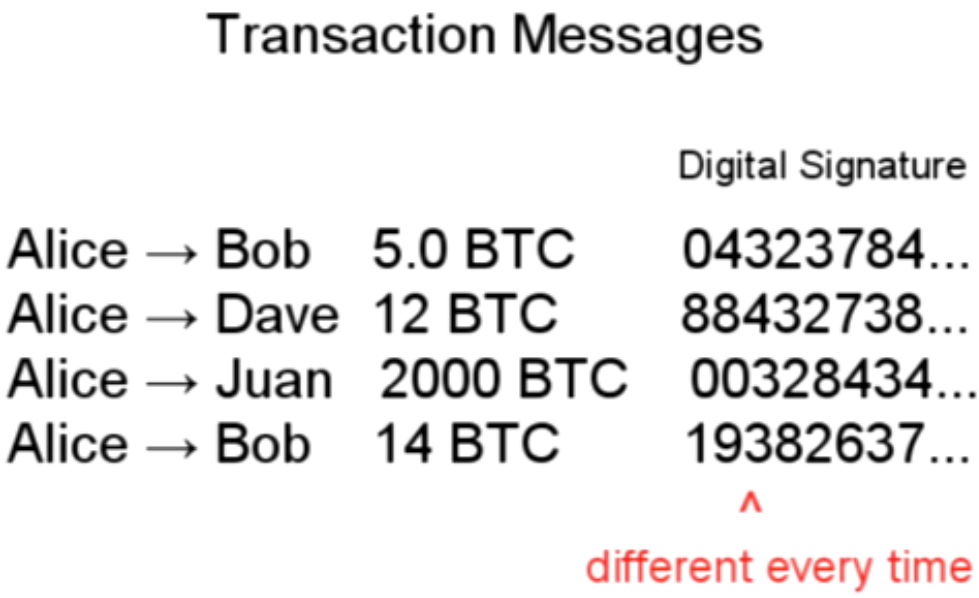
Σε ένα αρχικό επίπεδο, για να στείλει χρήματα η Alice στον Bob εκπέμπει απλά ένα μήνυμα με τους λογαριασμούς και το ποσό της συναλλαγής.

”Μετέφερε 5 BTC απο την Alice στον Bob”

Κάθε κόμβος που θα το λάβει θα ανανεώσει το αντιγραφο του αρχείου του και θα στείλει το μήνυμα της συναλλαγής στον επόμενο κόμβο. Σε αυτό το σημείο μπαίνει το ερώτημα της αυθεντικότητας του μηνύματος και πως οι κόμβοι μπορούν να διαπιστώσουν την εγκυρότητα του αποστολέα. Οι κανόνες του blockchain ορίζουν ότι για κάθε συναλλαγή είναι απαραίτητη η παροχή ενός μοναδικού κωδικού, ο οποίος αποτελείται ψηφιακή υπογραφή. Με τη χρήση του αλγορίθμου ελλειπτικής κρυπτογραφίας, η ψηφιακή υπογραφή πιστοποιεί τον αποστολέα του μηνύματος, αποκλείοντας με μαθηματικό τρόπο την αντιγραφή η πλαστογράφηση στον ψηφιακό κόσμο. Σε αντίθεση με έναν σταθερό κωδικό, για κάθε συναλλαγή χρειάζεται μια διαφορετική ψηφιακή υπογραφή. Έχοντας υπολογίσει ότι οι συναλλαγές γίνονται με αγνώστους, υπάρχει ανάγκη να μην αποκαλυφθεί ένας κωδικός που μπορεί να αντιγραφεί από κάποιον μέσα στο δίκτυο. Μια ψηφιακή υπογραφή δουλεύει χρησιμοποιώντας δύο διαφορετικά, συνδεδεμένα μεταξύ τους, κλειδιά. Ένα ”ιδιωτικό” κλειδί για τη δημιουργία της υπογραφής και ενα ”δημόσιο” κλειδί για την επαλήθευση της υπογραφής κάθε φορά. Το ιδιωτικό κλειδί αντιπροσωπεύει τον πραγματικό κωδικό και η ψηφιακή υπογραφή αντιπροσωπεύει έναν μεσάζοντα, που αποδεικνύει την ιδιοκτησία του ιδιωτικού κλειδιού, χωρίς όμως να το αποκαλύπτει.

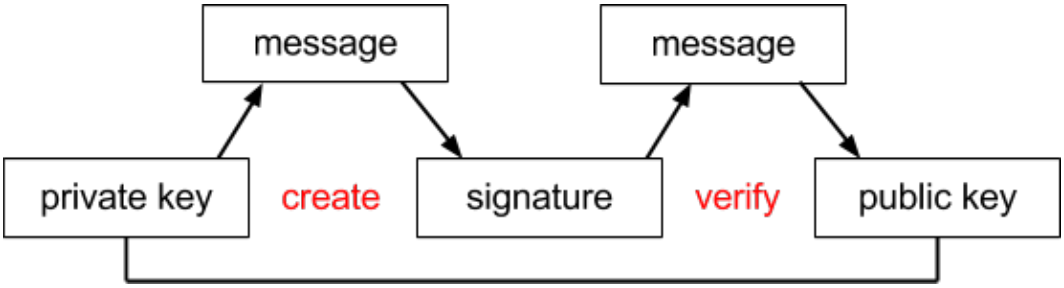
Τα δημόσια κλειδιά είναι στην ουσία οι διευθύνσεις αποστολής σε πολλά κρυπτονομίσματα, δηλαδη οταν αποστέλλονται χρήματα σε κάποιον, στέλνονται στο δημόσιο κλειδί τους.

Για να γίνει μια συναλλαγή, πρέπει ο αποστολέας να αποδείξει ότι είναι ο πραγματικός κάτοχος της διεύθυνσης δημόσιου κλειδιού από την οποία θα αποσταλούν τα χρήματα. Αυτό γίνεται με την δημιουργία μιας ψηφιακής υπογραφής από το μήνυμα της συναλλαγής και το ιδιωτικό κλειδί του λογαριασμού. Ψηφιακή υπογραφή = f (μήνυμα, ιδιωτικό κλειδί) Άλλοι κόμβοι στο δίκτυο μπορούν να χρησιμοποιήσουν αυτή την υπογραφή σε μια διαφορετική συνάρτηση για να επαληθεύσουν ότι αντιστοιχεί στο δημόσιο κλειδί της συναλλαγής.




Σχήμα 2.2: Διαφορετική Ψηφιακή Υπογραφή για κάθε συναλλαγή

```
1 1 =? f (message, public key, digital signature)
```




Σχήμα 2.3: Σχεδιάγραμμα λειτουργίας

Από τους μαθηματικούς τύπους που σχετίζονται με την ψηφιακή υπογραφή, οι κόμβοι είναι σε θέση να επαληθεύσουν ότι ο αποστολέας έχει στην κατοχή του το ιδιωτικό κλειδί χωρίς να το χρειάζεται να το δουν.

Είναι ιδιαίτερα σημαντικό το γεγονός ότι η παραγόμενη ψηφιακή υπογραφή εξαρτάται από το μήνυμα της συναλλαγής, το οποίο σημαίνει ότι θα είναι διαφορετική για κάθε συναλλαγή και επομένως δεν μπορεί να επαναχρησιμοποιηθεί από κάποιον άλλον σε διαφορετική συναλλαγή. Αυτή η εξάρτηση από το μήνυμα σημαίνει επίσης ότι κανείς δεν μπορεί να τροποποιήσει το μήνυμα κατά τη μετάδοσή του κατά μήκος του δικτύου, καθώς τυχόν αλλαγές στο μήνυμα καθιστούν την υπογραφή άκυρη.

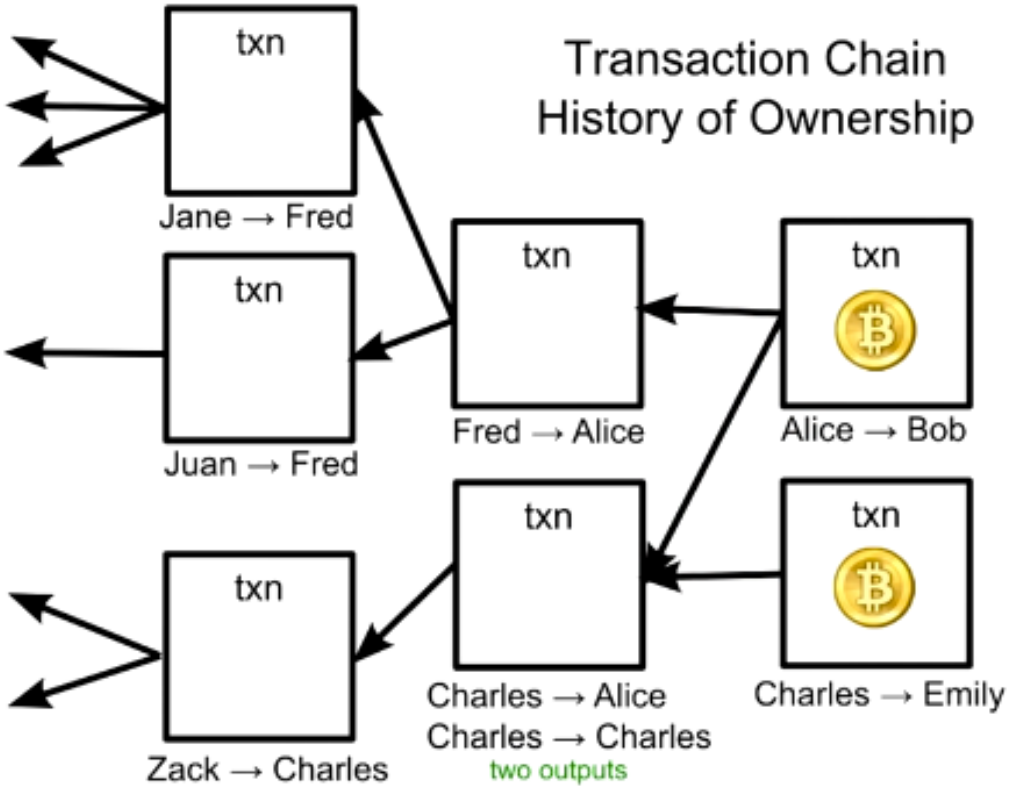
## 2.3 Συναλλαγές κρυπτονομισμάτων και αρχείο συναλλαγών

Έχει γίνει γνωστό ότι οι ψηφιακές υπογραφές χρησιμοποιούνται για να εξασφαλιστεί η εγκυρότητα μιας συναλλαγής, αλλά δεν έχει εξηγηθεί σε βάθος ο τρόπος με τον οποίο οι κόμβοι του δικτύου παρακολουθούν τα ποσά του λογαριασμού. Στην πραγματικότητα, δεν τηρούνται καθόλου αρχεία ισοζυγίου λογαριασμών. Αντί για παρακολούθηση των υπολοίπων σε κάθε λογαριασμό, η κυριότητα των ποσών επαληθεύεται μέσω συνδέσεων με προηγούμενες συναλλαγές.

Για παράδειγμα, για να αποσταλούν 5.0 BTC στον Bob, η Alice πρέπει να κάνει αναφορά σε προηγούμενες συναλλαγές μέσω των οποίων έλαβε συνολικά 5 ή περισσότερα Bitcoin. Αυτές οι αναφερόμενες συναλλαγές ονομάζονται "εισροές". Άλλοι κόμβοι που επαληθεύουν αυτή τη συναλλαγή θα ελέγξουν αυτές τις εισροές για να εξασφαλίσουν ότι η Alice ήταν στην πραγματικότητα ο παραλήπτης τους και επίσης ότι οι εισροές ανέρχονται τουλάχιστον σε 5 Bitcoin.

Μέσω αυτών των εισροών, η ιδιοκτησία των Bitcoins μεταδίδεται σε ένα είδος αλυσίδας, όπου η εγκυρότητα κάθε συναλλαγής εξαρτάται από προηγούμενες συναλλαγές. Και για κάθε συναλλαγή ελέγχεται η εγκυρότητα όλων των προηγούμενων εισροών. Όταν εγκατασταθεί για πρώτη φορά το λογισμικό Bitcoin, κατεβάζει κάθε συναλλαγή που έγινε ποτέ και ελέγχει την εγκυρότητα του καθενός μέχρι την πρώτη συναλλαγή. Με αυτό τον τρόπο διασφαλίζεται η ασφάλεια και εγκυρότητα του συστήματος. Εφόσον οι συναλλαγές γίνονται μεταξύ αγνώστων, είναι σημαντικό να ελέγχεται κάθε συναλλαγή. Αυτή η διαδικασία μπορεί να διαρκέσει πάνω από 24 ώρες, αλλά πρέπει να γίνει μόνο μία φορά.

Μόλις μία συναλλαγή χρησιμοποιηθεί μία φορά, θεωρείται από το σύστημα ότι δαπανήθηκε και δεν μπορεί να χρησιμοποιηθεί ξανά. Διαφορετικά, κάποιος μπορεί να ξοδέψει δύο φορές μια εισροή αναφερόμενη σε πολλαπλές συναλλαγές. Έτσι, κατά την επαλήθευση μιας συναλλαγής, επιπλέον των άλλων ελέγχων, οι κόμβοι βεβαιώνουν ότι οι εισροές δεν έχουν ήδη δαπανηθεί. Για να είναι σαφές, για κάθε είσοδο, οι κόμβοι ελέγχουν κάθε άλλη συναλλαγή που έγινε ποτέ για να βεβαιωθείτε ότι η είσοδος δεν έχει χρησιμοποιηθεί σε προηγούμενη συναλλαγή. Ενώ αυτό μπορεί να φαίνεται ως ιδιαίτερα χρονοβόρα διαδικασία, καθώς υπάρχουν πλέον πάνω από 200 εκατομμύρια συναλλαγές, γίνεται γρήγορα με έναν πίνακα μη δαπανημένων συναλλαγών. Έτσι, αντί για ένα αρχείο ισοζυγίων, οι κόμβοι των κρυπτονομισμάτων παρακολουθούν έναν τεράστιο κατάλογο συναλλαγών. Η κατοχή κρυπτονομισμάτων σημαίνει ότι υπάρχουν συναλλαγές σε αυτό τον κατάλογο που δείχνουν τον λογαριασμό του δικαιούχου και δεν έχουν δαπανηθεί ή, με άλλα λόγια, χρησιμοποιούνται ως εισροές σε άλλες συναλλαγές.




Σχήμα 2.4: Αλυσίδα κυριότητας

Μια ενδιαφέρουσα συνέπεια αυτής της ιδιοκτησιακής διάρθρωσης είναι ότι η για να υπολογιστεί το ποσό που κατέχει ένας λογαριασμός απαιτεί την αναδρομή κάθε συναλλαγής που έγινε ποτέ και τη συγκέντρωση όλων των αχρησιμοποίητων εισροών τους.

Μια άλλη ενδιαφέρουσα σημείωση σχετικά με τις συναλλαγές είναι ότι το σύστημα μπορεί να υποστηρίξει πιο σύνθετες λειτουργίες από απλά να μεταφέρει χρήματα μεταξύ λογαριασμών.

Αποδεικνύεται ότι οι έξοδοι του συστήματος μοιάζουν περισσότερο με παζλ που πρέπει να λυθούν παρά με απλές ”διευθύνσεις”. Η αποστολή χρημάτων μέσω κρυπτονομισμάτων μοιάζει περισσότερο με την τοποθέτηση χρημάτων σε μία δημόσια θυρίδα και την επισύναψη ενός μαθηματικού παζλ που πρέπει να λυθεί για να ανοίξει. Το παζλ καθορίζεται χρησιμοποιώντας μια ειδική γλώσσα προγραμματισμού και, ενώ είναι συνήθως σχεδιασμένη έτσι ώστε να μπορεί να το λύσει μόνο ο κάτοχος ενός δημόσιου κλειδιού, είναι δυνατόν να προστεθούν πιο περίπλοκες συνθήκες. Για παράδειγμα, θα μπορούσαν να απαιτηθούν 2 από

τις 3 υπογραφές για μια συναλλαγή μεσεγγύησης. Ένα άλλο παράδειγμα είναι η πρώτη συναλλαγή bitcoin που έγινε ποτέ, η οποία ήταν ένα παζλ που μπορούσε οποιοσδήποτε να λύσει.

Ενώ το μεγαλύτερο μέρος του κώδικα της διεκδίκησης των κρυπτονομισμάτων αποκρύπτεται από τον χρήστη, ο καθένας μπορεί να γράψει το δικό του λογισμικό και τις προϋποθέσεις διεκδίκησης, αν και αυτό μπορεί να είναι επικίνδυνο. Πάνω από 2600 BTC χάθηκαν σε μία παρτίδα συναλλαγών εξαιτίας μιας διεύθυνση με λανθασμένη μορφή.

Αυτό τονίζει ένα σημαντικό μέρος των κρυπτονομισμάτων. Δεδομένου ότι δεν υπάρχει τράπεζας ή κάποια πιστωτική εταιρεία να στηρίζει το σύστημα, όπου μπορεί να απευθυνθεί κάποιος σε περιπτώσεις σφαλμάτων, μπορεί να οδηγήσει στη μόνιμη απώλεια κρυπτονομισμάτων, όχι μόνο από έναν λογαριασμό αλλά από την οικονομία των κρυπτονομισμάτων συνολικά. Αν χαθεί το ιδιωτικό κλειδί, τα χρήματα που σχετίζονται με το αντίστοιχο δημόσιο κλειδί θα χαθούν για πάντα. Επειδή οι άνθρωποι είναι πολύ πιθανό να χάσουν τα ιδιωτικά κλειδιά λόγω σφαλμάτων του σκληρού δίσκου και των ανεπαρκών αντιγράφων ασφαλείας, αυτό σημαίνει ότι το κρυπτονόμισμα θα είναι τελικά αποπληθωριστικό.

## 2.4 Αλγόριθμος Ψηφιακής Υπογραφής

Όπως εξηγήθηκε παραπάνω, μια ψηφιακή υπογραφή επιτρέπει σε ένα χρήστη να αποδείξει ότι κατέχει ένα ιδιωτικό κλειδί και ότι το χρησιμοποίησε για να υπογράψει ένα συγκεκριμένο μήνυμα, χωρίς ποτέ να αποκαλύψει το ιδιωτικό του κλειδί. Εδώ είναι μια απλοποιημένη μέθοδος, που αν και ανασφαλής, μοιράζεται κάποια από τη δομή ενός πραγματικού Αλγόριθμου Ψηφιακής Υπογραφής: RSA.

Εξετάζοντας τις ακόλουθες μεταβλητές:

```
1 p = public key
2 q = private key
3 p * q = N also public
4 m = message
```

Αν δημιουργηθεί μια υπογραφή από το ιδιωτικό κλειδί και το μήνυμα, για παράδειγμα, s = q * m, τότε ένα τρίτο μέρος μπορεί να επαληθεύσει την υπογραφή ελέγχοντας s * p = m * N. Αυτό ισχύει επειδή m * q * ρ = m * N και N = q * ρ.

Με τον τρόπο αυτό, το τρίτο μέρος χρειάζεται μόνο να γνωρίζει το δημόσιο διαθέσιμο κλειδί, N, το μήνυμα και την υπογραφή. Το μόνο μειονέκτημα αυτής

της τεχνικής είναι ότι κάποιος που ξέρει πώς να διαιρέσει θα είναι σε θέση να υπολογίσει το ιδιωτικό μου κλειδί με q = N / p. Τα πραγματικά μαθηματικά ανήκουν στην κατηγορία της γενικής άλγεβρας και περιλαμβάνει modular αριθμητική και την δυσκολία παραγοντοποιηθούν εξαιρετικά μεγάλους αριθμούς. Για παράδειγμα το N = q * p, όπου q και p είναι μεγάλοι αριθμοί (300+ ψηφία). Δεδομένου ότι μόνο το N, η εύρεση q και p θα απαιτούσε τεράστια ποσότητα εικασίας και ελέγχου, αλλά γνωρίζοντας το q εκ των προτέρων επιτρέπει να βρεθεί το p με απλή διαίρεση. Αυτή η trapdoor συνάρτηση της παραγοντοποίησης επιτρέπει τον αλγόριθμο RSA. Ο αλγόριθμος δημόσιου κλειδιού / ιδιωτικού κλειδιού που χρησιμοποιείται στο Bitcoin καλείται Elliptic Curve DSA [?]και βασίζεται στη δυσκολία εύρεσης ενός διακριτού λογαρίθμου. Το πλεονέκτημα της ECDSA έναντι του RSA είναι ότι το μέγεθος κλειδιού που απαιτείται για μια συγκεκριμένη τιμή ασφάλειας είναι πολύ μικρότερο.

## 2.5 Ανωνυμία

Το δίκτυο είναι διαθέσιμο με πλήρη ανωνυμία. Μπορεί να γίνει πρόσβαση στο δίκτυο του κρυπτονομίσματος μέσω δικτύου TOR που αποκρύπτει τη διεύθυνση IP του χρήστη. Επίσης μπορεί να χρησιμοποιηθεί χωρίς ποτέ να αποκαλυφθεί περισσότερη πληροφορία για τον λογαριασμο από το δημόσιο κλειδί του. Και για να αποφευχθεί η σύνδεση των συναλλαγών με ένα λογαριασμό υπάρχει η δυνατότητα να δημιουργείται ένα νέο δημόσιο κλειδί για κάθε εισερχόμενη συναλλαγή.

Ωστόσο, είναι δυνατόν να συνδεθούν αθέλητα τα δημόσια κλειδιά μαζί. Στη συναλλαγή που παρουσιάστηκε νωρίτερα, χρησιμοποιήθηκαν ως πηγές 6 συναλλαγές "εισροών" και παρά το γεγονός ότι όλες αυτές οι εισροές έχουν αποσταλεί σε διαφορετικές διευθύνσεις, όλες έχουν συνδεθεί σε αυτή τη συναλλαγή. Ο αποστολέας απέδειξε ότι ανήκε σε όλες τις διευθύνσεις, παρέχοντας την ψηφιακή υπογραφή για να ξεκλειδώσει το καθένα. Έρευνες έχουν χρησιμοποιήσει αυτούς τους συνδέσμους για να μελετήσουν τη συμπεριφορά των χρηστών των κρυπτονομισμάτων.

Το ερώτημα είναι αν η δημιουργία ενός δημόσιου κλειδιού ως "διεύθυνση εισροών" θα μπορούσε ενδεχομένως να δημιουργήσει έναν σύνδεσμο με την πραγματική ταυτότητά του χρήστη, αλλά ακόμη και αυτό το βήμα είναι ανώνυμο και μπορεί να γίνει χωρίς σύνδεση με το δίκτυο. Επειδή υπάρχουν τόσες πολλές διαφορετικές πιθανές διευθύνσεις, δεν υπάρχει κανένας λόγος να ελεγχθεί αν κάποιος άλλος έχει ήδη το κλειδί. Στην πραγματικότητα, αν πάρετε το κλειδί κάποιου άλλου, θα έχετε πρόσβαση στα χρήματά τους.

Αυτός είναι ο συνολικός αριθμός πιθανών διευθύνσεων του κρυπτονομίσματος Bitcoin:

$$1461501637330902918203684832716283019655932542976 (1,46 \cdot 10^{48} \text{ ή } 2^{160})$$

Αυτοί οι αριθμοί προστατεύουν το σύστημα των κρυπτονομισμάτων με διάφορους τρόπους, γι 'αυτό είναι χρήσιμο να εκτιμηθεί πόσο πολύπλοκο είναι. Ορισμένες εκτιμήσεις για τον αριθμό των σπόρων άμμου σε ολόκληρο τον κόσμο είναι περίπου

$$7,5 \cdot 10^{18} \text{ ή } 7,500,000,000,000,000$$

Σύμφωνα με αυτό το παράδειγμα, αντίστοιχα ισχύει ότι κάθε κόκκος άμμου αντιπροσώπευε μια ολόκληρη άλλη Γη με πρόσθετους κόκκους. Η πιθανότητα αντιστοίχισης δύο διευθύνσεων μειώνεται καθώς ο αριθμός των χρηστών αυξάνεται με βάση το Πρόβλημα των Γενεθλίων, αλλά εξακολουθεί να έχει την εμβέλεια των

$$2,9 \cdot 10^{39}$$

με ένα δισεκατομμύριο διευθύνσεις.

## 2.6 Διπλή δαπάνη στα κρυπτονομίσματα

Επαληθεύοντας την ψηφιακή υπογραφή, το σύστημα γνωρίζει ότι μόνο ο πραγματικός κάτοχος θα μπορούσε να έχει δημιουργήσει την συναλλαγή μηνύματος. Και για να βεβαιωθεί ότι ο αποστολέας έχει πραγματικά χρήματα για να υλοποιήσει την συναλλαγή, ελέγχει επίσης κάθε εισροή, διασφαλίζοντας ότι δεν έχει δαπανηθεί. Εδώ υπάρχει ένα μεγάλο θέμα ασφάλειας στο σύστημα που μπορεί να κάνει αυτό το "unspent check" αναξιόπιστο, και αυτό έχει να κάνει με τη σειρά των συναλλαγών.

Λαμβάνοντας υπόψη ότι οι συναλλαγές μεταφέρονται από κόμβο σε κόμβο μέσω του δικτύου, δεν υπάρχει καμία εγγύηση ότι η σειρά με την οποία λαμβάνονται αντιπροσωπεύει τη σειρά με την οποία δημιουργήθηκαν. Και δεν πρέπει να εμπιστεύεται την χρονική σήμανση, επειδή θα μπορούσε κάποιος εύκολα να την τροποποιήσει για να εμφανίζει διαφορετική στιγμή δημιουργίας μια συναλλαγή. Σε αντίθεση με ένα κεντρικό σύστημα όπως το paypal, όπου είναι εύκολο για έναν κεντρικό υπολογιστή να παρακολουθεί τη σειρά των συναλλαγών.

Επομένως, δεν υπάρχει κανένας τρόπος να διαπιστωθεί αν μια συναλλαγή ήρθε πριν από μια άλλη και αυτό ανοίγει την πιθανότητα για απάτη στο σύστημα.

Ένας κακόβουλος χρήστης, η Alice, θα μπορούσε να στείλει μια συναλλαγή δίνοντας χρήματα στον Bob, περιμένοντας τον Bob να στείλει ένα προϊόν και έπειτα να στείλει μια άλλη συναλλαγή που αναφέρει την ίδια "εισροή" πίσω στον εαυτό της. Λόγω διαφορών στους χρόνους διάδοσης, ορισμένοι κόμβοι στο δίκτυο θα λάβουν τη 2η συναλλαγή "διπλής δαπάνης" πριν από την πρώτη για τον Bob. Και όταν φτάσει η συναλλαγή του Bob, θα τη θεωρούσαν άκυρα επειδή προσπαθεί να επαναχρησιμοποιήσει μια εισροή. Οπότε ο Bod θα χάσει τόσο το παραδοθέν προϊόν όσο και τα χρήματά του. Γενικά, θα υπάρχει διαφωνία στο δίκτυο για το αν ο Bob ή η Alice έχουν τα χρήματα, επειδή δεν υπάρχει τρόπος να αποδειχθεί ποια συναλλαγή είναι πρώτη.

Υπό αυτό το πρίσμα, πρέπει να υπάρχει ένας τρόπος για το σύνολο του δικτύου να συμφωνήσει σχετικά με τη σειρά των συναλλαγών, πράγμα που αποτελεί μια πολύ δύσκολη πρόκληση σε ένα αποκεντρωμένο σύστημα. Η λύση που δόθηκε είναι ένας έξυπνος τρόπος για να καθοριστεί και να διασφαλιστεί η σειρά προτεραιότητας μέσα από ένα είδος μαθηματικό πρόβλημα με βάση τον χρόνο.

## 2.7 Ταξινόμηση των συναλλαγών στο BlockChain

Το σύστημα Bitcoin ταξινομεί τις συναλλαγές τοποθετώντας τις σε ομάδες που ονομάζονται block και συνδέοντας αυτά τα μπλοκ μαζί σε μια αλυσίδα που ονομάζεται BlockChain. Να σημειωθεί ότι αυτό είναι διαφορετικό από την αλυσίδα συναλλαγών που έχει αναφερθεί. Το BlockChain χρησιμοποιείται για την ταξινόμηση των συναλλαγών, ενώ η αλυσίδα συναλλαγών παρακολουθεί τον τρόπο που αλλάζει η ιδιοκτησία των κρυπτονομισμάτων.

Κάθε block στο BlockChain περιέχει μια αναφορά στο προηγούμενο block , και αυτό τοποθετεί το ένα μπλοκ μετά το άλλο στον χρόνο. Το σύστημα μπορεί να διαβάσει τις αναφορές προς τα πίσω μέχρι την πρώτη ομάδα συναλλαγών που έχει κάνει ο χρήστης. Οι συναλλαγές στο ίδιο μπλοκ θεωρείται ότι έχουν συμβεί ταυτόχρονα και οι συναλλαγές που δεν βρίσκονται ακόμη σε ένα μπλοκ ονομάζονται "μη επιβεβαιωμένες" ή "αταξινόμητες".

Οποιοσδήποτε κόμβος μπορεί να συμπεριλάβει ένα σύνολο μη επιβεβαιωμένων συναλλαγών σε ένα block και να το μεταδώσει στο υπόλοιπο δίκτυο ως πρόταση για το ποιό θα είναι το επόμενο block στο BlockChain. Επειδή πολλοί χρήστες θα μπορούσαν να δημιουργήσουν ένα block ταυτόχρονα, μπορούν να υπάρχουν αρκετές επιλογές από τις οποίες το σύστημα πρέπει να διαλέξει το επόμενο block. Για αυτή την απόφαση δεν μπορεί να βασιστεί στην σειρά με την οποία φτάνουν τα block, επειδή οι συναλλαγες μπορούν να φτάσουν με διαφορετική σειρά από

διαφορετικά σημεία του δικτύου.

Μέρος της λύσης που χρησιμοποιείται από το Bitcoin είναι ότι κάθε έγκυρο block πρέπει να περιέχει την απάντηση σε ένα πολύ ιδιαίτερα πολύπλοκο μαθηματικό πρόβλημα. Οι υπολογιστές τρέχουν ολόκληρο το κείμενο ενός block μαζί μια πρόσθετη τυχαία τιμή σε έναν αλγόριθμο που ονομάζεται κρυπτογραφικός κατακερματισμός έως ότου η έξοδος είναι μικρότερη από ένα συγκεκριμένο όριο.

Μια συνάρτηση hash δημιουργεί ένα σύντομο digest από οποιοδήποτε αυθαίρετο μήκος κειμένου, στην περίπτωσή μας, το αποτέλεσμα είναι ένας αριθμός 32 byte. Ακολουθούν ορισμένα παραδείγματα της συγκεκριμένης συνάρτησης κατακερματισμού SHA256:

```
SHA256("short sentence")
0x0acdf28f4e8b00b399d89ca51f07fef34708e729ae15e85429c5b0f403295cc9

SHA256("The quick brown fox jumps over the lazy dog")
0xd7a8fbb307d7809469ca9abcb0082e4f8d5651e46d3cdb762d02d0bf37c9e592

SHA256("The quick brown fox jumps over the lazy dog.")
0xef537f25c895bfa782526529a9b63d97aa631564d5d789c2b765448c8635fb6c
```

Να σημειωθεί πόσο διαφορετική είναι η έξοδος της συνάρτησης με της προσθήκη μιας μόνο επιπλέον περιόδου στο τέλος του τρίτου παραδείγματος. Η έξοδος είναι εντελώς απρόβλεπτη, οπότε ο μόνος τρόπος για να βρεθεί μια συγκεκριμένη τιμή εξόδου είναι να γίνονται τυχαίες εικασίες. Μοιάζει ιδιαίτερα με την προσπάθεια να μαντευτεί ο συνδυασμός σε μια κλειδαριά. Κάποιος υπολογιστής μπορεί να το βρει με την πρώτη προσπάθεια, αλλά κατά μέσο όρο χρειάζονται πολλές επαναλήψεις. Στην πραγματικότητα, ένας τυπικός υπολογιστής θα χρειαζόταν αρκετά χρόνια να βρει την λύση σε ένα block.

Με κάθε υπολογιστή σε ολόκληρο το δίκτυο Bitcoin, χρειάζονται περίπου 10 λεπτά κατά μέσο όρο για να βρεθεί κάποια λύση. Ο πρώτος υπολογιστής που επιλύει το μαθηματικό πρόβλημα μεταδίδει το block και το σύνολο των συναλλαγών του γίνεται αποδεκτό ως το επόμενο στο BlockChain. Η τυχαιότητα στο μαθηματικό πρόβλημα το καθιστά απίθανο να το λύσουν δύο διαφορετικοί υπολογιστές ταυτόχρονα. Περιστασιακά, ωστόσο, θα λυθούν ταυτόχρονα περισσότερα από ένα block, οδηγώντας σε διαφορετικά πιθανά κλαδιά του συστήματος. Σε αυτή την περίπτωση, απλά προστίθενται πάνω από το πρώτο που λήφθηκε. Άλλοι κόμβοι μπορεί να έχουν λάβει τα block με διαφορετική σειρά και θα προσθέτουν πάνω στο πρώτο block που έλαβαν.

Αυτή η περίπτωση θα επιλυθεί όταν κάποιος άλλος λύσει ένα καινούργιο block. Ο γενικός κανόνας είναι ότι πάντα να προστεθεί στο μεγαλύτερο διαθέσιμο

παρακλάδι του BlockChain. Τα μαθηματικά καθιστούν σπάνια την ταυτόχρονη επίλυση των block και ακόμη πιο σπάνια να συμβεί πολλές φορές στη σειρά. Το τελικό αποτέλεσμα είναι ότι το BlockChain σταθεροποιείται γρήγορα, που σημαίνει ότι όλοι συμφωνούν για σειρά των block μέχρι ένα σημείο από το τέλος του BlockChain.

## 2.8 Επίθεση διπλής δαπάνης στο BlockChain

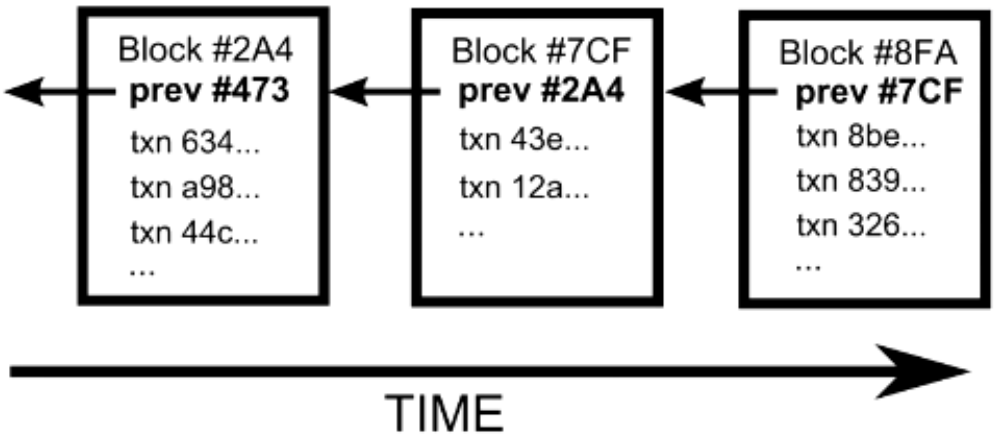
Το γεγονός ότι υπάρχει κάποια ασάφεια στο τέλος του BlockChain έχει μερικές σημαντικές επιπτώσεις στην ασφάλεια των συναλλαγών. Για παράδειγμα, αν μια συναλλαγή βρίσκεται σε έναν από τους μικρότερους κλάδους, θα χάσει τη θέση της στη γραμμή εντός του BlockChain. Τυπικά, αυτό σημαίνει ότι θα επιστρέψει απλώς στο pool των μη επιβεβαιωμένων συναλλαγών και θα συμπεριληφθεί σε ένα επόμενο block. Δυστυχώς, η δυνατότητα των συναλλαγών να χάσουν τη θέση τους ανοίγει την πιθανότητα στην επίθεση διπλής δαπάνης που ήταν και το αρχικό πρόβλημα που προσπαθεί να λύσει το BlockChain.

Η Alice, στέλνει χρήματα στον Bob. Ο Bob στη συνέχεια περιμένει τη συναλλαγή να "επιβεβαιωθεί" στο BlockChain και στη συνέχεια να στείλει ένα προϊόν. Επειδή οι κόμβοι πάνε πάντα σε ένα μεγαλύτερο κλάδο, αν η Alice μπορεί να δημιουργήσει ένα μεγαλύτερο κλάδο που αντικαθιστά τη συναλλαγή με τον Bob με μια συναλλαγή προς κάποιον άλλο, τα χρήματά του Bob θα διαγραφούν. Η συναλλαγή του Bob θα επιστραφεί πίσω στο pool με τις μη επιβεβαιωμένες συναλλαγές. Αλλά επειδή η Alice την αντικατέστησε με μια άλλη συναλλαγή που χρησιμοποιεί την ίδια εισροή της, οι κόμβοι θα θεωρούν τώρα τη συναλλαγή του Bob άκυρη, επειδή αναφέρεται σε μια ήδη δαπανημένη εισροή.

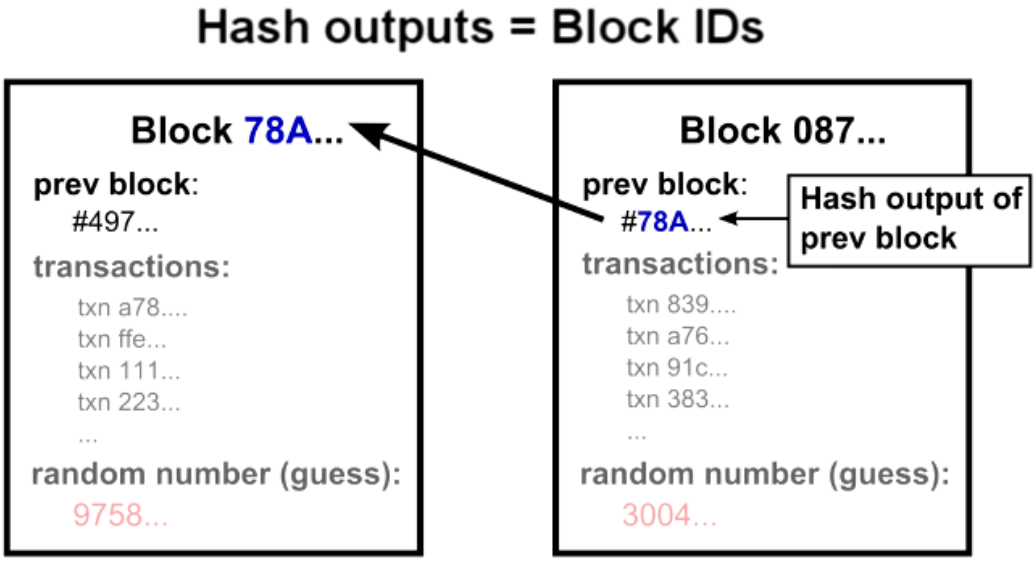
## 2.9 Πρόληψη επίθεσης διπλής δαπάνης

Πώς λοιπόν το σύστημα ταξινόμησης εμποδίζει την Alice να εξαπατήσει τον Bob; Όπως αναφέρθηκε προηγουμένως, η επίλυση ενός block προϋποθέτει την προσπάθεια να βρεθεί ο κρυπτογραφικός κατακερματισμός του block ώστε να είναι κάτω από μια ορισμένη τιμή και αυτό γίνεται προσθέτοντας διαφορετικούς τυχαίους αριθμούς στο τέλος του block. Μόλις επιλυθεί, η έξοδος κατακερματισμού είναι σαν ένα μοναδικό αποτύπωμα που αναγνωρίζει αυτό το block. Αν αλλάξει ακόμη και ένας χαρακτήρας στο block, ο κατακερματισμός του block θα είναι εντελώς διαφορετικός. Η έξοδος κατακερματισμού ή το δακτυλικό αποτύπωμα

είναι στην πραγματικότητα αυτό που χρησιμοποιείται ως αναφορά ”προηγούμενου block”. Ένα αποτέλεσμα αυτού είναι ότι δεν υπάρχει τρόπος να μπει ένα block στη μέση του BlockChain , επειδή η τιμή κατακερματισμού για το νέο block θα είναι διαφορετική και η αναφορά block του επόμενου δεν θα το δείχνει. Αλλά ακόμα πιο σημαντικό, ένα block δεν μπορεί να λυθεί πριν λύσει το προηγούμενο block. Η προηγούμενη αναφορά block είναι μέρος του κειμένου που περνάει από τη συνάρτηση κατακερματισμού, οπότε τυχόν αλλαγές σε αυτήν θα απαιτούν ξανά επίλυση.



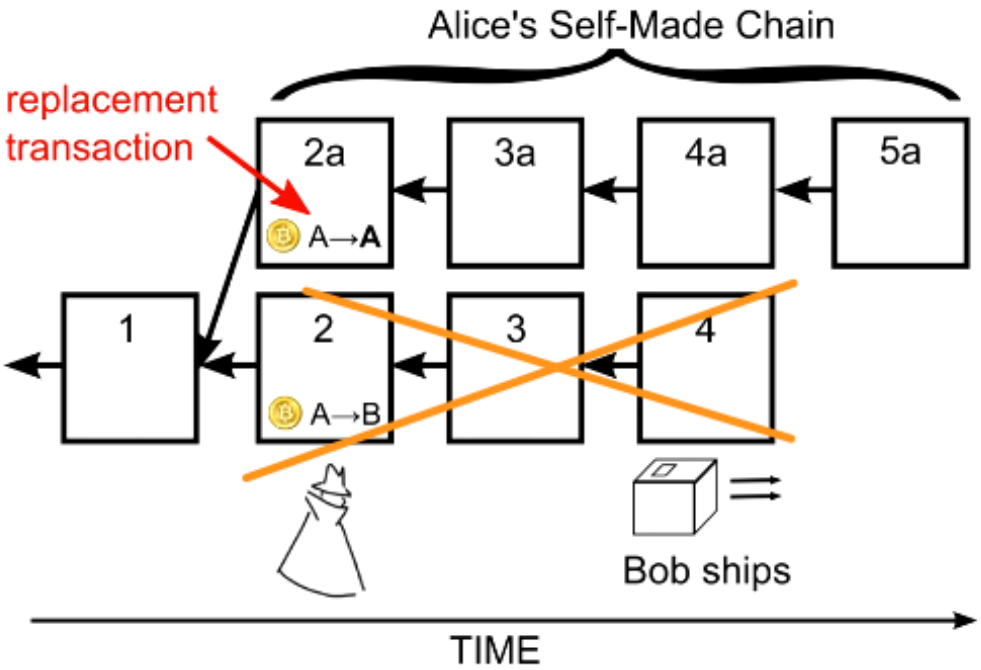

Σχήμα 2.5: Η αναφορά του προηγούμενου block στο BlockChain




Σχήμα 2.6: Κατακερματισμός με το προηγούμενο block για την δημιουργία του επόμενου

Επιστρέφοντας στην Alice, αυτός είναι ο λόγος για τον οποίο δεν μπορεί να προ υπολογίσει ένα block. Μπορεί να ξεκινήσει την επίλυση μόνο όταν το μπλοκ στο οποίο θα προσθέσει έχει επιλυθεί, και η τιμή hash του είναι γνωστή. Είναι λοιπόν σε έναν αγώνα με το υπόλοιπο δίκτυο έως ότου ο Bob αποστείλει ένα προϊόν, αυτή είναι η χρονική στιγμή που θέλει να παρουσιάσει έναν μεγαλύτερο κλάδο. Πρέπει να δουλέψει μυστικά, γιατί αν ο Bob είχε ακούσει για το διπλό block της, δεν θα έστελνε προφανώς το προϊόν.

Μια τελευταία ερώτηση είναι αν η Alice θα μπορούσε να ξεπεράσει όλους τους άλλους στο δίκτυο αν είχε έναν εξαιρετικά γρήγορο υπολογιστή ή ίσως ένα δωμάτιο γεμάτο υπολογιστές. Αλλά ακόμη και με χιλιάδες υπολογιστές, είναι απίθανο να κερδίσει τον αγώνα για να λύσει ένα block, επειδή δεν αγωνίζεται κανένα υπολογιστή, αλλά ολόκληρο το δίκτυο. Η περίπτωση μοιάζει σαν λοταρία. Μπορεί να χειρίζεται χιλιάδες υπολογιστές, ή ισοδύναμα, να αγοράζει χιλιάδες λαχεία, αλλά ακόμα και τότε είναι πολύ πιθανότερο να κερδίσει κάποιος άλλος. Θα χρειαζόταν να ελέγξει το ήμισυ της συνολικής υπολογιστικής ισχύος σε ολόκληρο το δίκτυο για να έχει 50% πιθανότητα να λύσει ένα μπλοκ πριν από κάποιον άλλο. Και πολύ περισσότερο για να έχει την πιθανότητα να λύσει πιο γρήγορα πολλά block στη σειρά.




Σχήμα 2.7: Επίθεσης διπλής δαπάνης με διπλό κλάδο

Συνεπώς, οι συναλλαγές στο BlockChain προστατεύονται από μια μαθηματική σχέση που τοποθετεί τον εισβολέα εναντίον ολόκληρου του υπολοίπου δικτύου.

Αποτέλεσμα των block που χτίζουν το ένα πάνω στο άλλο είναι ότι οι συναλλαγές που βρίσκονται πίσω στην αλυσίδα είναι πιο ασφαλείς. Ένας επιτιθέμενος θα πρέπει να ξεπεράσει το δίκτυο για μεγαλύτερο χρονικό διάστημα για να πραγματοποιήσει μια επίθεση διπλής δαπάνης και να αντικαταστήσει ένα block. Επομένως, το σύστημα είναι μόνο ευάλωτο σε μια τέτοια επίθεση στο τέλος του BlockChain, γι ᾽αυτό συνιστάται η αναμονή για μερικά block πριν γίνει θεωρηθεί η συναλλαγή ως ολοκληρωμένη.

Τίποτα που περιγράφεται μέχρι τώρα δεν απαιτεί εμπιστοσύνη. Όταν λαμβάνονται πληροφορίες από αγνώστους στο δίκτυο Bitcoin, ο χρήστης μπορεί να ελέγξει από μόνος του ότι οι λύσεις του block είναι σωστές. Και επειδή τα μαθηματικά προβλήματα είναι τόσο πολύπλοκα, γνωρίζει ότι δεν υπάρχει κανένας τρόπος με τον οποίο κάποιος εισβολέας θα μπορούσε να τα δημιούργησε μόνος του. Οι λύσεις αποδεικνύουν ότι η υπολογιστική ισχύς ολόκληρου του δικτύου επετεύχθη.

## 2.10 Εξόρυξη και Ομάδες υπολογιστών

Τώρα που περιγράφηκε πως μεταφέρονται χρήματα μέσω ψηφιακών υπογραφών και αλυσίδων συναλλαγών και πώς η σειρά των συναλλαγών αυτών προστατεύεται στο BlockChain, θα αναλυθεί από όπου προέρχονται τα κρυπτονομίσματα. Για να στείλει ένας χρήστης χρήματα, θα πρέπει να αναφερθεί σε μια προηγούμενη συναλλαγή, όπου ήταν ο παραλήπτης, αλλά δεν είναι ακόμα ξεκάθαρο πως μπαίνουν τα κρυπτονομίσματα σε αυτήν την αλυσίδα ιδιοκτησίας.

Ως ένας τρόπος για την αργή και τυχαία παραγωγή και διανομή κρυπτονομισμάτων, μια ”ανταμοιβή” δίνεται σε όποιον λύνει ένα block. Αυτός είναι ο λόγος για τον οποίο η επίλυση block ονομάζεται εξόρυξη, αν και ο πραγματικός σκοπός του είναι να επαληθεύει τις συναλλαγές και να διασφαλίζει την αλυσίδα block. Κάθε 4 χρόνια, η ανταμοιβή του block μειώνεται κατά το ήμισυ, έτσι τελικά δεν θα κυκλοφορήσουν περισσότερα νομίσματα. Θα δημιουργηθούν περίπου 21 εκατομμύρια. Λαμβάνοντας υπόψη ότι ένας χρήστης μπορεί να στείλετε το 1/100 εκατομμυριοστό του κρυπτονομίσματος (.00000001), συμπεραίνεται οτι ο διαθέσιμος συνολικός αριθμός δεν θα περιορίσει πιθανώς τη χρησιμότητα του κρυπτονομίσματος.

Μόλις σταματήσουν οι ανταμοιβές του block, ποιο κίνητρο θα έχουν οι μηχανές εξόρυξης να επεξεργάζονται συναλλαγές; Εκτός από την ανταμοιβή του block, οι μηχανές εξόρυξης λαμβάνουν επίσης οποιαδήποτε τέλη συναλλαγών που μπορούν προαιρετικά να συμπεριληφθούν στις συναλλαγές. Αυτή τη στιγμή, οι μηχανές εξόρυξης θα συμπεριλάβουν συναλλαγές χωρίς αμοιβές σε block επειδή το κύριο

κίνητρο τους είναι η ανταμοιβή, αλλά στο μέλλον οι συναλλαγές θα επεξεργάζονται σύμφωνα με τις συνημμένες αμοιβές και οι συναλλαγές χωρίς τέλη πιθανόν να αγνοηθούν. Έτσι, η αποστολή χρημάτων στο δίκτυο πιθανόν να μην είναι δωρεάν, αλλά μπορεί να είναι πιο φθηνή από τα τρέχοντα τέλη των πιστωτικών καρτών.

Όπως αναφέρθηκε προηγουμένως, θα χρειαστούν αρκετά χρόνια για έναν τυπικό υπολογιστή να λύσει ένα block, οπότε η πιθανότητα ενός ατόμου να λύσει ποτέ κάποιον πριν το υπόλοιπο δίκτυο, το οποίο συνήθως διαρκεί 10 λεπτά, είναι πολύ χαμηλή. Για να εξασφαλιστεί μια πιο σταθερή ροή εισοδήματος, πολλοί άνθρωποι συμμετέχουν σε ομάδες που ονομάζονται εξορυκτικές ομάδες, mining pools, που συλλογικά εργάζονται για την επίλυση block και διανέμουν ανταμοιβές με βάση τις εργασίες που συνέβαλαν. Αυτά λειτουργούν κάπως σαν ομάδες λαχειοφόρων αγορών μεταξύ των συναδέλφων, εκτός από το ότι μερικές από αυτές τις ομάδες είναι αρκετά μεγάλες και περιλαμβάνουν περισσότερο από το 20

Το γεγονός ότι μερικές από αυτές τις ομάδες είναι τόσο μεγάλες έχει κάποιες σημαντικές επιπτώσεις στην ασφάλεια του συστήματος. Όπως αναφέρθηκε προηγουμένως, είναι πολύ απίθανο ένας επιτιθέμενος να λύσει αρκετά block στη σειρά ταχύτερα από το υπόλοιπο δίκτυο, αλλά είναι πιθανό και η πιθανότητα αυξάνεται καθώς η ισχύς επεξεργασίας του εισβολέα αυξάνεται σε σχέση με το υπόλοιπο δίκτυο. Στην πραγματικότητα, μία από τις ομάδες εξόρυξης, η BTC Guild, έχει λύσει από μόνη της 6 μπλοκ στη σειρά και έχει περιορίσει οικειοθελώς τα μέλη της για να αποτρέψει την αναξιοπιστία σε ολόκληρο το δίκτυο του bitcoin.

Ακόμη και με σημαντική υπολογιστική ισχύ, όσο πιο πίσω γίνεται στο BlockChain βρίσκεται μια συναλλαγή, τόσο πιο δύσκολο θα ήταν για έναν εισβολέα να την αλλάξει, καθώς πρέπει να ξεπεράσει το υπόλοιπο δίκτυο για το χρονικό διάστημα μεταξύ της αποστολής μιας συναλλαγής της αποστολής ενός προϊόν. Η τρέχουσα σύσταση είναι να περιμένει ο χρήστης να πραγματοποιηθεί μια συναλλαγή σε τουλάχιστον ένα block, ή να λάβει μια επιβεβαίωση, πριν την θεωρήσει οριστική. Και για μεγαλύτερες συναλλαγές, περιμένει τουλάχιστον 6 block. Υπό το πρίσμα της ικανότητας της BTC Guild να λύσει 6 block στη σειρά, μπορεί να θέλει να περιμένει ακόμα περισσότερο.

## 2.11 Χρόνος επιβεβαίωσης

Με βάση το σχεδιασμό του συστήματος, κάθε block χρειάζεται περίπου 10 λεπτά για να λυθεί, οπότε η αναμονή για 6 block θα διαρκέσει περίπου μία ώρα. Σε σύγκριση με τα μερικά δευτερόλεπτα που λαμβάνει μια συναλλαγή με πιστωτική

κάρτα, μια αναμονή για τόσο πολύ μπορεί να φαίνεται περίπλοκη, αλλά να τονιστεί ότι οι πελάτες πιστωτικών καρτών μπορούν να διεκδικήσουν μια κλεμμένη κάρτα μήνες αργότερα για να μην χρεωθούν τις συναλλαγές προς εμπόρους. Στην πραγματικότητα είναι πολύ πιο γρήγορο το σύστημα επιβεβαίωσης από την πλευρά του εμπόρου.

Γιατί 10 λεπτά ανά block; Η συγκεκριμένη επιλογή των 10 λεπτών ήταν κάπως αυθαίρετη, αλλά εξαιρετικά μικροί χρόνοι θα μπορούσαν να οδηγήσουν σε αστάθεια, και μεγαλύτεροι χρόνοι θα καθυστερήσουν επιπλέον τις επιβεβαιώσεις. Καθώς περισσότεροι υπολογιστές εντάσσονται στο δίκτυο και εξειδικευμένο υλικό έχει σχεδιαστεί ειδικά για εξόρυξη, ο χρόνος λύσης του μπλοκ θα γίνει πολύ μικρότερος. Για να αντισταθμιστεί, κάθε δύο εβδομάδες, όλο το λογισμικό του Bitcoin επαναπροσδιορίζει τη δυσκολία του μαθηματικού προβλήματος ώστε να στοχεύσει επίλυση σε 10 λεπτά. Για σύγκριση, ένα παρόμοιο ψηφιακό νόμισμα που ονομάζεται Litecoin μπόρεσε να λειτουργήσει με χρόνο επίλυσης 2,5 λεπτών.

## 2.12 Συμπεράσματα και σύνοψη λειτουργίας της Blockchain

Συνοπτικά, τα κρυπτονομίσματα είναι ένα μαθηματικά προστατευμένο ψηφιακό νόμισμα που διατηρείται από ένα δίκτυο χρηστών. Οι ψηφιακές υπογραφές εξουσιοδοτούν μεμονωμένες συναλλαγές, η κυριότητα μεταβιβάζεται μέσω αλυσίδων συναλλαγών και η σειρά των συναλλαγών αυτών προστατεύεται στο BlockChain. Απαιτώντας την επίλυση πολύπλοκων μαθηματικών προβλημάτων με κάθε block, οι επιτιθέμενοι βρίσκονται ενάντια σε ολόκληρο το υπόλοιπο δίκτυο σε έναν υπολογιστική αγώνα που είναι απίθανο να κερδίσουν. [?]

# Κεφάλαιο 3

# Παρουσίαση του περιβάλλοντος Ethereum

## 3.1 Βασικά Τεχνικά Χαρακτηριστικά

Αν και η εστίαση της παρούσης εργασίας δεν είναι η επεξήγηση των λεπτομερειών της τεχνολογίας blockchain, καθώς υπάρχουν ήδη δεκάδες πηγές που τις εξηγούν με λεπτομέρεια, θα πρέπει να αναφερθούμε σε κάποια τεχνικά χαρακτηριστικά της πλατφόρμας Ethereum διότι καθορίζουν σε μεγάλο βαθμό τις τεχνικές απαιτήσεις που εισάγονται στα smart contracts. Όλες οι λεπτομέρειες που θα αναφερθούν στη συνέχεια επεξηγούνται πλήρως τεχνικά στο Ethereum yellowpaper [**?**].

Ποιος είναι λοιπόν ο σκοπός της πλατφόρμας Ethereum ? Σε πολύ βασικό επίπεδο, είναι η επίτευξη της συμφωνίας μεταξύ ατόμων στο διαδίκτυο για το περιεχόμενο μιας καταγραφής ανά πάσα στιγμή. Σε αντίθεση με το bitcoin, η καταγραφή αυτή δεν είναι μόνο μια λίστα με εγγραφές που αντιπροσωπεύουν διευθύνσεις πορτοφολιών και τα ποσά που αντιστοιχούν στο καθένα, αλλά επίσης, η κατάσταση προγραμμάτων. Με την έννοια κατάσταση προγράμματος εννοούμε τις τιμές των μεταβλητών κάθε είδους του προγράμματος.

Η επίτευξη της συμφωνίας επιτυγχάνεται μέσω της λύσης που προσφέρει η τεχνολογία blockchain. Οι προτεινόμενες αλλαγές από τους χρήστες της εφαρμογής εκπέμπονται στο διαδίκτυο μέσω τεχνολογιών peer-to-peer. Στη συνέχεια, κάποιοι χρήστες συλλέγουν αυτές τις προτεινόμενες αλλαγές, τις εκτελούν και βρίσκουν την επόμενη κατάσταση του δικτύου. Αυτοί οι χρήστες ονομάζονται εξορύχοι ( miners ). Από τους miners όμως μόνο ένας θα είναι αυτός του οποίου η λύση θα

συμπεριληφθεί στην κοινή λίστα από αλλαγές(blockchain) και η επιλογή γίνεται μέσω της μεθόδου proof-of-work. Οι miners δηλαδή, καλούνται να λύσουν ένα δύσκολο αλγοριθμικό πρόβλημα, το οποίο θυμίζει επίθεση ωμής βίας ( brute-force ) καθώς υποχρεούνται να δοκιμάζουν τυφλά αριθμητικούς συνδυασμούς σε έναν αλγόριθμο απαιτήσεων μνήμης. Μόνο αν τα αποτελέσματα τους πληρούν κάποιες προϋποθέσεις που εξασφαλίζουν την τυχαιότητα μπορεί η λύση τους να συμπεριληφθεί στη blockchain.

Αξίζει να αναφερθεί ότι σύντομα το Ethereum θα αλλάξει την μέθοδο επιλογής από proof-of-work σε proof-of-stake [**?**]. Το πρωτόκολλο proof-of-stake στην ουσία επιτρέπει σε έναν χρήστη να κλειδώσει για κάποιο χρονικό διάστημα ένα ποσό σε ether, ένα μερίδιο δηλαδή (stake). Ανάλογα με το ποσό που έχουν κλειδώσει οι χρήστες, θα αυξάνεται η πιθανότητα να επιλεχθούν από το σύστημα να συμπεριλάβουν την λύση τους στη blockchain. Αυτή η λύση θα δώσει τέλος στην κατασπατάληση ενέργειας από τις επαναληπτικές μεθόδους του πρωτοκόλλου proof-of-work. Και στις δυο περιπτώσεις, το κίνητρο που έχουν οι miners να συμπεριλάβουν λύσεις στο δίκτυο είναι ότι κατά την υποβολή μιας αποδεκτής λύσης, δημιουργούνται ether και πιστώνονται στον λογαριασμό τους, εξορύσσονται δηλαδή. Η ορθότητα των λύσεων ελέγχεται με ευκολία καθώς οι αλγόριθμοι επιλογής είναι τύπου hashing.

Οι λογαριασμοί στο Ethereum είναι σειρές δεκαεξαδικών αριθμών. Υπάρχουν δύο τύποι λογαριασμών - λογαριασμοί χρηστών και λογαριασμοί smart contracts. Οι πρώτοι είναι και οι πιο απλοί - περιέχουν κάποιο ποσό σε ether και η λειτουργία τους είναι είτε να λάβουν είτε να στείλουν ether σε άλλους λογαριασμούς. Οι λογαριασμοί των smart contracts είναι πιο πολύπλοκοι. Περιέχουν μέσα τους κώδικα ο οποίος ενεργοποιείται όταν σταλεί μια συναλλαγή προς τον λογαριασμό του smart contract. Η συναλλαγή αυτή λοιπόν, περιέχει τα ορίσματα τα οποία θα επεξεργαστεί ο κώδικας του smart contract για προκύψει η νέα του κατάσταση. Ο κώδικας του smart contract μπορεί ο ίδιος να στείλει συναλλαγές σε άλλες διευθύνσεις χρηστών ή ακόμα και σε άλλα smart contracts. Αυτό επιδεικνύει ότι η διεπαφή μέσω εφαρμογών σε ethereum είναι εφικτή.

Όπως αναφέραμε, η γλώσσα προγραμματισμού του Ethereum, η οποία ονομάζεται Solidity, είναι turing-complete, το οποίο σημαίνει ότι μπορεί να εκτελεί ατέρμονες βρόγχους. Τι συμβαίνει λοιπόν όταν κάποιο smart contract “κολλήσει” ή έχει δομηθεί λάθος?

Εδώ το ethereum ορίζει ένα σύστημα χρέωσης της κάθε λειτουργίας που το σύστημα εκτελεί για λογαριασμό κάποιου χρήστη ή smart contract. Το σύστημα αποφασίζει ανά τακτά χρονικά διαστήματα το κόστος κάθε υπολογιστικού βήματος μέσω της τιμής gasprice, και οι συναλλαγές πρέπει να περιέχουν έναν αριθμό

gas, ο οποίος δηλώνει το μέγιστο αριθμό υπολογιστικών βημάτων τον οποίο είναι διατεθειμένες να πληρώσουν. Το γινόμενο gas * gasprice είναι ένα ποσό σε ether το οποίο επιβαρύνει τον αποστολέα. Εάν μια συναλλαγή προς smart contract ξεμείνει από gas χωρίς το smart contract να εκτελέσει τη λειτουργία του, επιστρέφεται ένα μήνυμα λάθους και το smart contract επιστρέφει στην αρχική του κατάσταση με σκοπό να αποφευχθούν σφάλματα από την μη ολοκληρωμένη εκτέλεση. Το κόστος του gas το επωφελούνται οι miners ώστε να έχουν κίνητρο να συμπεριλαμβάνουν τα πιο υπολογιστικά πολύπλοκα προγράμματα στις λύσεις τους.

Ήδη είναι εμφανής μια νέα πρόκληση στη δημιουργία smart contracts. Όχι μόνο η λειτουργία τους, αλλά και το μέγεθος του κώδικα τους χρεώνεται σε ether για την υποβολή τους. Ένα καλογραμμένο smart contract θα έχει μικρό κόστος λειτουργίας ενώ ένα κακογραμμένο smart contract όχι μόνο θα κοστίζει παραπάνω, αλλά έχει μεγαλύτερες πιθανότητες να αποτύχει, κοστίζοντας στον αποστολέα / χρήστη κάποιο ποσό. Είναι επίσης προφανές ότι ένα σφάλμα στον κώδικα θα μπορούσε να αδειάσει τον λογαριασμό του smart contract, δημιουργώντας μεγάλες ζημιές. Τέτοιου είδους θέματα θα αναλυθούν στο κεφάλαιο 3.

Ποια είναι λοιπόν η ωφέλεια της δημιουργίας εφαρμογών σε Ethereum ? Τα πλεονεκτήματα που προσφέρει η πλατφόρμα είναι αυτά της αποκέντρωσης και της δυνατότητας διεπαφής. Η εφαρμογή μας δεν χρειάζεται να διαθέτει δομές αποθήκευσης δεδομένων, καθώς τα δεδομένα βρίσκονται στους υπολογιστές των χρηστών του δικτύου. Φυσικά, αυτό δεν είναι εφικτό για μεγάλους όγκους δεδομένων λόγω του μεγάλου κόστους αποθήκευσης, αλλά είναι πολύ χρήσιμο για μικρές εφαρμογές, καθώς επίσης εφαρμογές δημιουργίας κρυπτονομισμάτων, εφαρμογές ηλεκτρονικής ψηφοφορίας και εφαρμογές crowdfunding.

## 3.2 Επισκόπηση λειτουργίας smart contracts από υποδομή υποστήριξης έως υποδομή διεπαφής

Σε αυτό το κεφάλαιο θα δείξουμε πως ένα smart contract εκτελεί εφαρμογές από επίπεδο υποδομής υποστήριξης ( back-end ) ως το επίπεδο υποδομής διεπαφής ( front-end ). Θα αναφερθούμε σε καθαρά blockchain εφαρμογές αντί για υβριδικές για λόγους απλότητας. Σε μία καθαρά blockchain εφαρμογή, το back-end είναι το smart contract, ενώ σε μια υβριδική υλοποίηση κάποια δεδομένα θα αποθηκεύονται στη blockchain και κάποια σε μια κεντρική δομή αποθήκευσης. Αναφερόμενοι στο back-end, το smart contract μας δηλαδή, θα μπορούσαμε να αναλύσουμε ποιά είναι τα βασικά στοιχεία του. Αυτά διαφέρουν σε κάθε εκτέλεση, υπάρχουν όμως μερικές βασικές κοινές δομές. Οι δομές αυτές είναι συναρτήσεις, γραμμένες

κυρίως στη γλώσσα Solidity, η οποία είναι μια υψηλού επιπέδου γλώσσα αντικειμενοστραφούς προγραμματισμού η οποία θυμίζει την JavaScript. Μακροσκοπικά λοιπόν, ανιχνεύονται οι εξής δομές :

- Μεταφορά κρυπτονομισμάτων : Αυτές οι συναρτήσεις επιτρέπουν την ασφαλή μεταφορά κρυπτονομισμάτων είτε εντός του smart contract από έναν χρήστη σε κάποιον άλλο είτε εκτός του smart contract για κάποια πληρωμή.
- Κλήσεις σε άλλα smart contracts : Στον χώρο του Ethereum platform υπάρχουν διάφορα ασφαλή smart contracts που εκπληρώνουν χρήσιμες λειτουργίες όπως μαθηματικές πράξεις, επικοινωνία με το περιβάλλον και κρυπτογραφικές λειτουργίες. Είναι κοινή και αποτελεσματική τακτική να γίνεται κλήση σε αυτά από άλλα smart contracts αντί να ενσωματώνεται ο κώδικας τους.
- Χαρτογράφησης : Αυτές οι συναρτήσεις χρησιμοποιούνται κυρίως σε smart contracts που εκδίδουν ένα δικό τους νόμισμα, και αντιστοιχίζουν με ασφάλεια διευθύνσεις σε ποσά, στις οποίες έχουν πρόσβαση μόνο οι νόμιμοι κάτοχοι.
- Αποσύνθεσης : Καθώς ο μόνος τρόπος να διαγραφεί ένα smart contract είναι να το επιτρέπει ο ίδιος του ο κώδικας, οι συναρτήσεις αυτο-διαγραφής είναι ένα χρήσιμο εργαλείο για την διαχείριση των πόρων μιας παλαιάς έκδοσης ενός συμβολαίου ώστε να μεταφερθούν σε μια νέα έκδοση του. Είναι επίσης μια καλή πρακτική για την διατήρηση του όγκου ολόκληρου του οικοσυστήματος σε χαμηλά επίπεδα.

Όσον αφορά την διεπαφή του χρήστη με την εφαρμογή, κανονικά γίνεται κατεβάζοντας και εγκαθιστώντας μέσω διάφορων προγραμμάτων ολόκληρη την Ethereum blockchain , κάνοντας τον υπολογιστή μας έναν πλήρη κόμβο ( full node ). Γινόμαστε έτσι ένα μέρος του peer-to-peer δικτύου του Ethereum. Ωστόσο, υπάρχουν εφαρμογές και βιβλιοθήκες που επιτρέπουν την αλληλεπίδραση με την blockchain χωρίς το πλήρες κατέβασμα της, οι οποίες ονομάζονται ελαφρού κόμβου ( light node ). Όσον αφορά εφαρμογές ιστοσελίδων, αρκεί ο χρήστης να εμπλουτίσει τον περιηγητή του με ένα πρόσθετο πρόγραμμα ( plugin) το οποίο θα περιέχει την βιβλιοθήκη web3.js . Αυτή η βιβλιοθήκη προσθέτει εντολές javascript η οποίες υλοποιούν την επικοινωνία με την blockchain. Στην συνέχεια, η υπόλοιπη σελίδα μπορεί να στηθεί με οποιοδήποτε framework έχει άνεση ο προγραμματιστής και τα αποτελέσματα να εμφανίζονται μέσω αυτού.

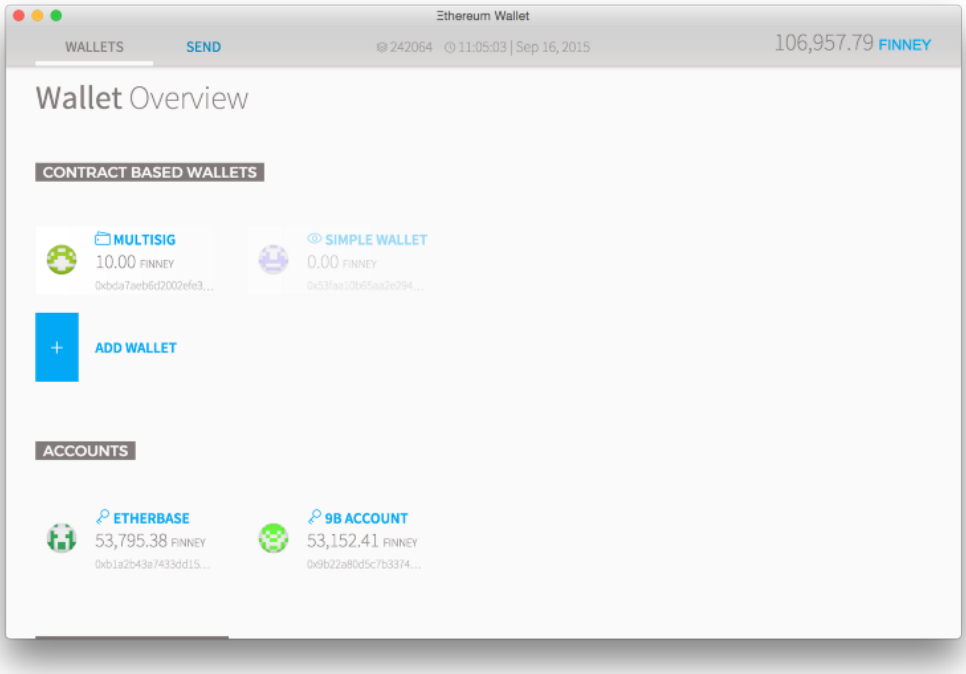
Η ροή ανάπτυξης dapps είναι η εξής. Πρώτα γίνεται η ανάλυση του προβλήματος, οι σχεδιαστικές απαιτήσεις και λαμβάνονται οι αποφάσεις για το ποια δεδομένα της εφαρμογής θα αποθηκεύονται εντός της blockchain και ποιά εκτός. Στη συνέχεια, γίνεται η ανάπτυξη του smart contract και ο έλεγχος ορθότητας. Για

τον έλεγχο χρησιμοποιούνται δίκτυα ελέγχου, πρώτα τοπικά στον υπολογιστή του προγραμματιστή και στη συνέχεια ειδικά διανεμημένα δίκτυα τα οποία προσομοιώνουν τη λειτουργία στο πραγματικό blockchain και δεν έχουν κόστος υποβολής προγραμμάτων. Ταυτόχρονα μπορεί να αναπτύσσεται το front end όσο υπάρχει γνώση του API του υπό ανάπτυξη smart-contract. Τέλος γίνεται η υποβολή ( deployment ) στο κύριο blockchain και το dapp είναι πλέον σε λειτουργία.

## 3.3 Παρουσίαση συμπληρωματικού λογισμικού ανάπτυξης

Ακολουθούν μερικά από τα βασικά εργαλεία και frameworks που χρειάζονται για την κατασκευή και τον έλεγχο smart contracts στο Ethereum Blockchain.

- **Mist**




Σχήμα 3.1: Mist Wallet View

Το Mist [?] είναι ένα πρόγραμμα πορτοφολιού, κρατάει δηλαδή τα ζεύγη ιδιωτικών-δημόσιων κλειδιών τα οποία είναι απαραίτητα για την επιβεβαίωση

μιας συναλλαγής.Μέσω αυτού είναι δυνατή η χρήση των ether του προγραμματιστή. Ταυτόχρονα υποστηρίζει την ενσωμάτωση κρυπτονομισμάτων που αντιστοιχούν στη διεύθυνση από άλλα Ethereum Smart Contracts, είναι δηλαδή ένα πορτοφόλι για πολλά διαφορετικά κρυπτονομίσματα.

- **Geth**


Σχήμα 3.2: Geth Library

Το Geth [?] είναι μια απαραίτητη βιβλιοθήκη για την ανάπτυξη smart contracts καθώς επεκτείνει τις εντολές αλληλεπίδρασης με τη blockchain τις οποίες προσφέρουν τα απλά wallets. Είναι μια από τις πρώτες βιβλιοθήκες που κατεβάζει όποιος θέλει να κάνει ανάπτυξη στο Ethereum.

- **web3.js**


Σχήμα 3.3: Web3 Library

Η βιβλιοθήκη web3.js [?] περιέχει παραπάνω λειτουργίες οι οποίες είναι απαραίτητες για την επικοινωνία του front-end των εφαρμογών με τη blockchain και είναι επίσης από τις πρώτες που χρειάζεται να κατέχουμε.

- **Parity**


Σχήμα 3.4: Parity

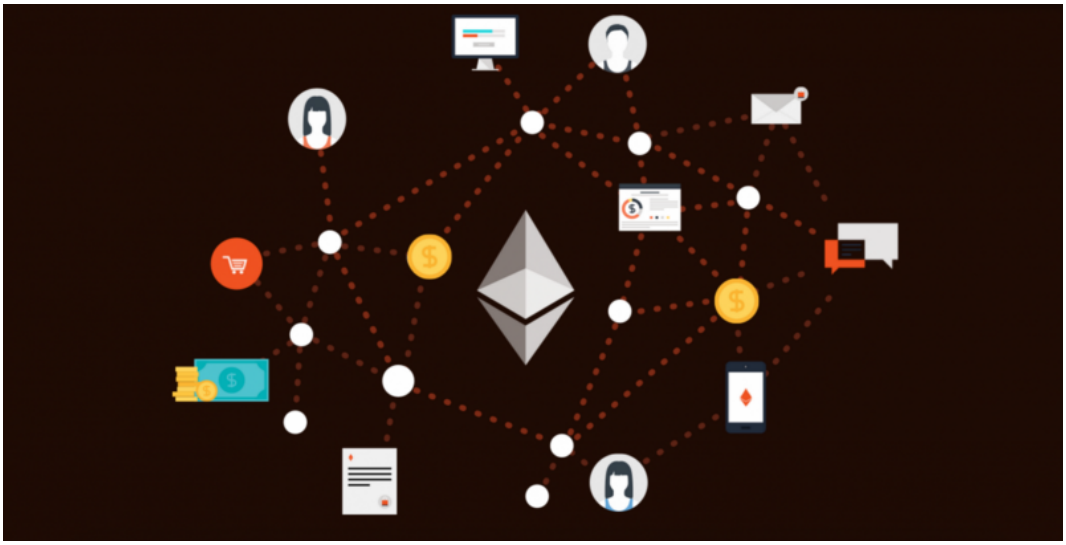
Το Parity [?] μας επιτρέπει να κατεβάσουμε γρήγορα και χωρίς μεγάλο όγκο την Ethereum Blockchain , κάνοντας τον υπολογιστή μας αυτό που ονομάζεται node ( κόμβος ) του δικτύου, κάτι που είναι απαραίτητο για το ανέβασμα του smart contract μας.

- **Truffle**


Σχήμα 3.5: Truffle

Το Truffle [?] είναι αυτή τη στιγμή το πιο διαδεδομένο framework ανάπτυξης smart contracts. Προσφέρει δυνατότητες compilation, ελέγχου και διαχείρισης του κώδικα μας, με βιβλιοθήκες που δίνουν τη δυνατότητα να γράψουμε smart contracts τα οποία θα αλλάξουν εύκολα όταν χρειαστεί και πολλά άλλα. Συνεχώς ανανεώνεται με νέες λειτουργίες.

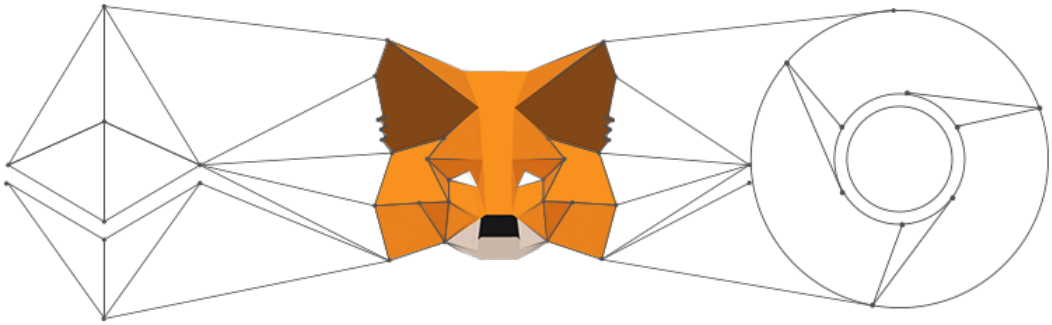
- **Testrpc**



Σχήμα 3.6: Testrpc

Το testrpc [?] είναι από τα πιο διαδεδομένα και εύκολα στη χρήση εργαλεία για testing των smart contracts. Μας επιτρέπει να δημιουργήσουμε μια τοπική blockchain με έναν αριθμό πορτοφολιών και χρησιμοποιώντας τα να εκτελέσουμε συναλλαγές για να επιβεβαιώσουμε τη λειτουργία του smart contract μας πριν το deployment σε κάποια δημόσια blockchain.

- **Metamask**



Σχήμα 3.7: Metamask

Το Metamask [?] είναι ένα ευρέως διαδεδομένο plug-in το οποίο δίνει στον περιηγητή μας την δυνατότητα επικοινωνίας με web3.js πρωτόκολλα. Συνεπώς, είναι κάτι που θα πρέπει να ενσωματώσουν οι χρήστες μιας dapp για να μπορούν να έχουν την πλήρη λειτουργικότητα της. Ταυτόχρονα, περιέχει ένα δικό της ενσωματωμένο ethereum wallet, το οποίο έχει μάλιστα πρόσβαση και στα διάφορα testing networks, οπότε χρησιμεύει και για την διαδικασία ελέγχου κατα την ανάπτυξη.

- **Etherscan**


Σχήμα 3.8: Etherscan

Το Etherscan [?] είναι μια ιστοσελίδα που επιτρέπει την εύκολη επίβλεψη των συναλλαγών που συμβαίνουν ανα πάσα στιγμή στο δίκτυο του Ethereum. Πέρα από την αναζήτηση συγκεκριμένων συναλλαγών για την επιβεβαίωση της λειτουργίας των προγραμμάτων μας, δίνει και διάφορες χρήσιμες μετρικές για την κατάσταση του δικτύου.

- **Zeppelin**


Σχήμα 3.9: Zeppelin

Το Zeppelin [?] αντίστοιχα φροντίζει να ελέγξει θέματα ασφάλειας του κώδικα μας και να αποτρέψει γνωστές αδυναμίες. Είναι πολύ σημαντικό να ελέγχουμε το κομμάτι της ασφάλειας των smart contract μας με όσο περισσότερα εργαλεία και μεθόδους είναι δυνατό.

## 3.4 Παρουσίαση παραδείγματος Smart Contracts σε Solidity

Στη συνέχεια θα γίνει η παρουσίαση ενός παραδείγματος smart contract γραμμένο με τη γλώσσα Solidity ώστε να γίνει μια καλύτερη κατανόηση των εννοιών που αναφέρθηκαν στα προηγούμενα κεφάλαια [?].

Το παράδειγμα είναι ένα smart contract ηλεκτρονικής ψηφοφορίας. Είναι σχετικά πολύπλοκο, ωστόσο επιδεικνύει αρκετές από τις ιδιότητες της Solidity. Ένα από τα κύρια προβλήματα μιας ηλεκτρονικής ψηφοφορίας είναι το πώς θα διαμοιραστούν οι ψήφοι στα κατάλληλα άτομα και πως να εμποδιστεί η χειραγώγηση. Το πρόγραμμα αυτό δεν θα λύσει όλα αυτά τα προβλήματα, θα δώσει όμως μια λύση ψηφοφορίας κατ'εξουσιοδότηση ώστε η καταμέτρηση των ψήφων θα είναι αυτόματη και εντελώς διαφανής. Ο δημιουργός του προγράμματος θα μπορεί να δώσει το δικαίωμα ψήφου σε διευθύνσεις και οι ψηφοφόροι θα μπορούν να ψηφίσουν μια από τις προτάσεις ή να αναθέσουν την ψήφο τους σε κάποιον άλλο. Στο τέλος του διαθέσιμου χρόνου, η συνάρτηση winningProposal() θα επιστρέψει την πρόταση με τις περισσότερες ψήφους.

```
1 pragma solidity ^0.4.11;
```

Εδώ καθορίζουμε την ελάχιστη έκδοση του solidity compiler που μπορεί να χρησιμοποιηθεί.

```
/// @title Voting with delegation.
contract Ballot {
    // This declares a new complex type which will
    // be used for variables later.
    // It will represent a single voter.
    struct Voter {
        uint weight; // weight is accumulated by delegation
        bool voted;  // if true, that person already voted
        address delegate; // person delegated to
        uint vote;   // index of the voted proposal
    }

    // This is a type for a single proposal.
    struct Proposal {
        bytes32 name;   // short name (up to 32 bytes)
        uint voteCount; // number of accumulated votes
    }
```

Τα παραπάνω είναι τύποι δομών στην solidity που χρησιμοποιούνται για την ομαδοποίηση μεταβλητών σε λογικές ενότητες

```
    address public chairperson;
```

Η μεταβλητή chairperson είναι τύπου address, ο οποίος διατηρεί διευθύνσεις ethereum μεγέθους 40 δεκαεξαδικών χαρακτήρων ή 20 bytes. Ως public, τα δεδομένα της θα είναι διαθέσιμα, και θα δημιουργηθεί αυτόματα μια συνάρτηση επιστροφής της τιμής της.

```
    // This declares a state variable that
    // stores a `Voter` struct for each possible address.
    mapping(address => Voter) public voters;
```

Το mapping είναι μια διαδικασία αντιστοίχησης. Σε αυτή την περίπτωση δημιουργούμε μια λίστα από δομές τύπου Voter που αντιστοιχίζονται σε μια μεταβλητή address η κάθε μια.

```
    // A dynamically-sized array of `Proposal` structs.
    Proposal[] public proposals;

    /// Create a new ballot to choose one of `proposalNames`.
    function Ballot(bytes32[] proposalNames) {
        chairperson = msg.sender;
        voters[chairperson].weight = 1;

        // For each of the provided proposal names,
        // create a new proposal object and add it
```

```
        // to the end of the array.
        for (uint i = 0; i < proposalNames.length; i++) {
            // `Proposal({...})` creates a temporary
            // Proposal object and `proposals.push(...)`
            // appends it to the end of `proposals`.
            proposals.push(Proposal({
                name: proposalNames[i],
                voteCount: 0
            }));
        }
    }
```

Η μεταβλητή msg.sender ορίζεται ως η διεύθυνση αυτού που καλεί τη συνάρτηση. Το άτομο αυτό ορίζεται ως chairperson. Προσθέτει επίσης μια ψήφο στις προτάσεις που υπέβαλε. Στη συνέχεια προστίθενται οι προτάσεις που υποβλήθηκαν στον πίνακα proposals.

```
    // Give `voter` the right to vote on this ballot.
    // May only be called by `chairperson`.
    function giveRightToVote(address voter) {
        // If the argument of `require` evaluates to `false`,
        // it terminates and reverts all changes to
        // the state and to Ether balances. It is often
        // a good idea to use this if functions are
        // called incorrectly. But watch out, this
        // will currently also consume all provided gas
        // (this is planned to change in the future).
        require((msg.sender == chairperson) && !voters[voter].
            voted && (voters[voter].weight == 0));
        voters[voter].weight = 1;
    }
```

Η συνάρτηση require θα επιτρέψει να γίνουν αλλαγές με επιτυχία μονο εαν ο αποστολέας είναι ο chairperson και ο voter του οποίου η διεύθυνση υποβάλλεται δεν έχει ψηφίσει και δεν έχει προστεθεί ήδη. Αλλιώς η συναλλαγή θα αποτύχει.

```
    /// Delegate your vote to the voter `to`.
    function delegate(address to) {
        // assigns reference
        Voter storage sender = voters[msg.sender];
        require(!sender.voted);

        // Self-delegation is not allowed.
        require(to != msg.sender);

        // Forward the delegation as long as
        // `to` also delegated.
        // In general, such loops are very dangerous,
        // because if they run too long, they might
```

```
14          // need more gas than is available in a block.
15          // In this case, the delegation will not be executed,
16          // but in other situations, such loops might
17          // cause a contract to get "stuck" completely.
18          while (voters[to].delegate != address(0)) {
19              to = voters[to].delegate;
20
21              // We found a loop in the delegation, not allowed.
22              require(to != msg.sender);
23          }
24
25          // Since `sender` is a reference, this
26          // modifies `voters[msg.sender].voted`
27          sender.voted = true;
28          sender.delegate = to;
29          Voter delegate = voters[to];
30          if (delegate.voted) {
31              // If the delegate already voted,
32              // directly add to the number of votes
33              proposals[delegate.vote].voteCount += sender.weight;
34          } else {
35              // If the delegate did not vote yet,
36              // add to her weight.
37              delegate.weight += sender.weight;
38          }
39      }
```

Ο ορισμός της μεταβλητής sender ώς storage λειτουργεί ως δείκτης, δηλαδή οι αλλαγές που θα γίνουν στη μεταβλητή sender θα εφαρμοστούν αμέσως στο αντικείμενο τύπου Voter στο οποίο αναφέρεται. Η συνάρτηση αυτή επιτρέπει τη μεταφορά δικαιώματος ψήφου σε κάποιον ψηφοφόρο/αντιπρόσωπο.

```
1       /// Give your vote (including votes delegated to you)
2       /// to proposal `proposals[proposal].name`.
3       function vote(uint proposal) {
4           Voter storage sender = voters[msg.sender];
5           require(!sender.voted);
6           sender.voted = true;
7           sender.vote = proposal;
8
9           // If `proposal` is out of the range of the array,
10          // this will throw automatically and revert all
11          // changes.
12          proposals[proposal].voteCount += sender.weight;
13      }
```

Η συνάρτηση αυτή υλοποιεί τη διαδικασία ψήφου. Οι αλλαγές και πάλι γίνονται στα δεδομένα του χρήστη μέσω αναφοράς. Σε περίπτωση που ο αριθμός της πρότασης που υποβάλλεται δεν υπάρχει στο σύστημα, η συνάρτηση θα αποτύχει

λόγω απόπειρας πρόσβασης σε στοιχείο του πίνακα που δεν υπάρχει.

```
    /// @dev Computes the winning proposal taking all
    /// previous votes into account.
    function winningProposal() constant
            returns (uint winningProposal)
    {
        uint winningVoteCount = 0;
        for (uint p = 0; p < proposals.length; p++) {
            if (proposals[p].voteCount > winningVoteCount) {
                winningVoteCount = proposals[p].voteCount;
                winningProposal = p;
            }
        }
    }

    // Calls winningProposal() function to get the index
    // of the winner contained in the proposals array and then
    // returns the name of the winner
    function winnerName() constant
            returns (bytes32 winnerName)
    {
        winnerName = proposals[winningProposal()].name;
    }
}
```

Οι τελευταίες δυο συναρτήσεις ορίζονται ως view , που σημαίνει ότι δεν μπορούν να μεταβάλλουν δεδομένα του προγράμματος και χρησιμοποιούνται κυρίως για την ανάγνωση των μεταβλητών του smart contract. Το πλεονέκτημα χρήσης τέτοιων συναρτήσεων είναι ότι η κλήση τους μπορεί να γίνει χωρίς κόστος σε gas σε αντίθεση με συναρτήσεις που μεταβάλλουν δεδομένα και άρα την κατάσταση του smart contract. Η πρώτη επιστρέφει την νικητήρια πρόταση αφού εκτελέσει τη καταμέτρηση των ψήφων και η δεύτερη επιστρέφει το όνομα της νικήτηριας πρότασης.

# Κεφάλαιο 4

# Ασφάλεια

Σε αυτό το κεφάλαιο θα ασχοληθούμε με βασικά θέματα ασφαλείας των smart contracts. Πρώτα θα αναλύσουμε τις αδυναμίες οι οποίες βρίσκονται είτε στο επίπεδο της Ethereum Virtual Machine, είτε στο επίπεδο της βασικής γλώσσας προγραμματισμού smart contracts, Solidity, είτε στο επίπεδο της γενικής λειτουργίας των blockchains, οι οποίες κατ'επέκταση οδηγούν σε κενά ασφαλείας στα smart contracts. Στην συνέχεια, θα εξετάσουμε πραγματικές περιπτώσεις στις οποίες τα κενά αυτά οδήγησαν σε παύση ή λανθασμένη λειτουργία των smart contracts και , σε ορισμένες περιπτώσεις απώλεια ether.

## 4.1 Αδυναμίες Επιπέδου Blockchain

### 4.1.1 Αδυναμία Μέτρησης Χρόνου

Στο επίπεδο της blockchain, δεν υπάρχει κάποιο καθολικό ρολόι για τη μέτρηση χρόνου ή για τον καθορισμό της χρονικής στιγμής που συμβαίνει κάποιο γεγονός. Αντί αυτού, χρησιμοποιούνται τα χαρακτηριστικά ονόματα των blocks, τα οποία είναι δεκαεξαδικοί αριθμοί, καθώς τα blocks εξορύσσονται ανά σχετικά σταθερά τακτικά χρονικά διαστήματα. Στα blocks αυτά, οι εξορύχοι θέτουν μια τιμή χρονικής στιγμής ( timestamp) η οποία όμως είναι βοηθητικού χαρακτήρα και μπορούν να την επιλέξουν αυθαίρετα. Επίσης, μεγάλες είσοδοι ή έξοδοι επεξεργαστικής ισχύος στο δίκτυο μπορούν να συντμήσουν ή να επεκτείνουν τον χρόνο εξόρυξης blocks όπως και διάφορες επιθέσεις denial-of-service (DoS ). Smart Contracts τα οποία βασίζονται σε χρονικές περιόδους για να εκτελέσουν εργασίες θα πρέπει

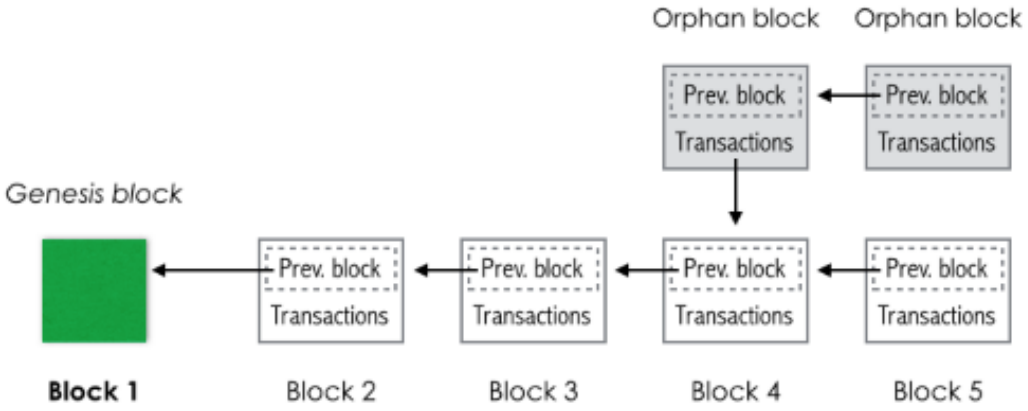
να λαμβάνουν υπ'όψιν αυτά τα περιθώρια λάθους και να μην βασίζουν κρίσιμες διαδικασίες σε ακριβείς χρονικές στιγμές ή στα block timestamps [?].

### 4.1.2 Μειωμένη Ταχύτητα

Τα δεδομένα στα οποία έχει πρόσβαση ένα smart contract είναι όλα όσα βρίσκονται στο δίκτυο του Ethereum. Επίσης, ο κώδικας ενός smart contract είναι ανοιχτός και εμφανής προς όλους και η εκτέλεση των προγραμμάτων από την EVM είναι ντετερμινιστική, δηλαδή δυο διαδικασίες με τις ίδιες εισόδους θα παράξουν τα ίδια αποτελέσματα. Συνεπώς, αλγόριθμοι οι οποίοι χρησιμοποιούν τέτοια δεδομένα σαν seeds για γεννήτριες τυχαίων αριθμών έχουν ένα σημαντικό κενό ασφαλείας, καθώς και ο αλγόριθμος και οι είσοδοι είναι γνωστές, άρα τα ίδια αποτελέσματα μπορούν να παραχθούν από τρίτους. Ακόμα και όταν χρησιμοποιούνται σαν είσοδοι τα μελλοντικά ονόματα ή timestamps από blocks, έρευνες έχουν δείξει ότι οι miners μπορούν να επηρεάσουν αυτά τα στοιχεία προς όφελος τους. ( links για τα papers από το 1007 - 49 κ 50). Συνεπώς τα smart contracts δεν πρέπει να βασίζονται σε τέτοιες τεχνικές για τη δημιουργία τυχαίων αποτελεσμάτων, αλλά σε άλλες λύσεις οι οποίες υπάρχουν διαθέσιμες.

### 4.1.3 Απρόβλεπτη κατάσταση προγράμματος

Όταν μια συναλλαγή εκπέμπεται στο δίκτυο από κάποιον χρήστη, μαζεύεται σε μια ομάδα από εκρεμμείς συναλλαγές τις οποίες οι miners συλλέγουν και εκτελούν με τρόπο αυθαίρετο, βασισμένο κυρίως σε δικούς τους αλγορίθμους μεγιστοποίησης κέρδους. Δίνουν προτεραιότητα δηλαδή σε συναλλαγές με υψηλή αξία gas. Αυτό σημαίνει ότι μια συναλλαγή που αποστάληκε πρώτη, δεν είναι απαραίτητο ότι θα εκτελεστεί και πρώτη. Επίσης, υπάρχει η περίπτωση να συμβεί μια διακλάδωση της blockchain, δηλαδή να βρουν δυο ή και παραπάνω άτομα ταυτόχρονα το επόμενο block και το ένα μερος του δικτύου να δουλεύει πάνω στο ένα block και το άλλο μέρος πάνω στο άλλο. Κάποια στιγμή το ένα από τα blocks θα αποκτήσει μεγαλύτερη αλυσίδα από τα υπόλοιπα και θα εδραιωθεί ως το κυρίαρχο, αλλά στο ενδιάμεσο υπάρχει μια περίοδος αβεβαιότητας, καθώς αν μια αλλαγή εκτελεστεί σε κάποια από τις εναλλακτικές αλυσίδες δεν θα έχει απαραίτητα εκτελεστεί και στην κεντρική, όποτε μπορεί το smart contract να επιστρέψει σε μια προηγούμενη κατάσταση [?].




Σχήμα 4.1: Οι αλλαγές που θα έχουν γίνει στα orphan blocks δεν θα ισχύουν απαραίτητα στην κεντρική αλυσίδα

Αυτό σημαίνει ότι όταν αλληλεπιδρούμε με ένα smart contract μια συγκεκριμένη στιγμή δεν είναι απαραίτητο πως όταν η αλληλεπίδραση αυτή εκτελεστεί το smart contract δεν θα έχει δεχτεί άλλες αλλαγές και αλληλεπιδράσεις στο ενδιάμεσο, ακόμα και από εντολές που στάλθηκαν μετέπειτα από την δικιά μας. Αυτό το πρόβλημα της μη τήρησης χρονικής προτεραιότητας είναι εγγενές στα peer-to-peer δίκτυα όπου η αίτηση μας για εξυπηρέτηση εκπέμπεται σε μια πληθώρα χρηστών που θα τη λάβουν ετεροχρονισμένα. Για την σωστή δημιουργία smart contract αυτό σημαίνει ότι τα προγράμματα μας δεν πρέπει να βασίζονται στην εκτέλεση γεγονότων με μια συγκεκριμένη σειρά για να παραμένουν λειτουργικά.

## 4.2 Αδυναμίες Επιπέδου EVM

### 4.2.1 Αμεταβλητότητα του πηγαίου κώδικα

Όταν δημιουργούμε ένα smart contract, αυτό καταγράφεται και δεν υπάρχει δυνατότητα αλλαγής του. Έτσι, οι χρήστες μπορούν να εμπιστευτούν ότι το πρόγραμμα εκτελεί τον σκοπό ύπαρξης τους και δεν πρόκειται να αλλάξει στο μέλλον. Το πρόβλημα είναι ότι οι αδυναμίες που μπορεί να υπάρχουν στον κώδικα δεν γίνεται να διορθωθούν με ενημερώσεις. Το πρόβλημα αυτό αντιμετωπίζεται πρώτα με άρτιο testing από την πλευρά των προγραμματιστών στα δοκιμαστικά chains, είτε με την χρήση ειδικών smart contracts τα οποία προωθούν τις κλήσεις στην τελευταία έκδοση του προγράμματος ώστε να δίνεται η δυνατότητα να υπάρχουν πολλές διαφορετικές εκδόσεις παράλληλα. Επίσης, είναι σημαντικό να υπάρχει μια συγκεκριμένη εντολή αυτοκαταστροφής του smart contract, η

οποία να διαχειρίζεται το τι γίνεται με τυχόν κεφάλαια που είναι αποθηκευμένα σε αυτό ώστε σε περίπτωση που βρεθεί κάποιο σφάλμα να μπορεί να διορθωθεί πριν γίνει μεγάλη ζημιά ή απώλεια κεφαλαίων.

### 4.2.2 Μη ελεγχόμενη αποστολή κεφαλαίων

Όταν εκτελούμε συναλλαγές μέσω ether, ή εντολές που έχουν να κάνουν με αποστολή σε άλλες διευθύνσεις, αν γίνει κάποιο ορθογραφικό λάθος στο όνομα της διεύθυνσης ( το οποίο υπενθυμίζουμε ότι είναι δεκαεξαδικός αριθμός) η αποστολή γίνεται στην λάθος διεύθυνση και τα ether "κλειδώνονται" εκεί, καθώς κανείς δεν έχει στην κατοχή του το ιδιωτικό κλειδί για τη νέα διεύθυνση, άρα είναι σαν να έχουν χαθεί. Επίσης, μπορεί η αποστολή ether να μην πετύχει για διάφορους λόγους, όπως από έλλειψη gas. Αν δεν ελέγξουμε ότι η συναλλαγή ολοκληρώθηκε, μπορεί το πρόγραμμα μας να περιέλθει σε λάθος κατάσταση. Συνεπώς πρέπει να ελέγχεται πάντα η ορθότητα και το αποτέλεσμα των πράξεων αποστολής και λήψης, κάτι στο οποίο βοηθάνε πλέον οι περισσότεροι compilers, αλλά καλό θα ήταν να υπάρχει πάντα στο μυαλό του προγραμματιστή.

### 4.2.3 Όριο gas ανα block

Η EVM έχει ένα όριο επιτρεπτού gas ανά block, το οποίο όταν ξεπεραστεί, το smart contract σταματάει να εκτελείται και επιστρέφει στην προηγούμενη του κατάσταση. Αυτό εκ των πραγμάτων θέτει ένα όριο στο μέγιστο αριθμό υπολογισμών που μπορούν να εκτελεστούν από ένα smart contract. Ο περιορισμός αυτός έχει πολλά αντίκτυπα στο πως πρέπει να γράφονται και να ελέγχονται τα smart contracts, με ιδιαίτερη προσοχή στους ατέρμονες βρόγχους, αναδρομικές κλήσεις και στην επεξεργασία ολοένα αυξανόμενων δεδομένων.

### 4.2.4 Bugs του compiler

Η EVM είναι μια νέα μηχανή/ compiler και ως εκ τούτου ενίοτε ανακαλύπτονται bugs τα οποία μπορούν να είναι τρωτά κενά ασφαλείας. Πρόσφατα για παράδειγμα, το Νοέμβριο του 2016, εμφανίστηκε ένα bug όπου μέσω κάποιας υπερχείλισης μπορούσαν να επηρεαστούν μεταβλητές αποθήκευσης. Το bug διορθώθηκε άμεσα, ωστόσο για ένα χρονικό διάστημα πολλά smart contracts ήταν ευάλωτα. Η λύση σε τέτοιου είδους προβλήματα που δεν μπορούν να προβλεφθούν είναι και πάλι

η ύπαρξη μιας συνθήκης διαφυγής στο smart contract με την οποία μπορεί να διαφυλάξει τα ether και τις πληροφορίες που διαθέτει.

## 4.3 Αδυναμίες επιπέδου γλώσσας Solidity

### 4.3.1 Επανεισαγωγή

Όταν γίνεται κλήση σε ένα smart contract η λειτουργία του οποίου δεν είναι αναδρομική, είναι δύσκολο για τον προγραμματιστή να αντιληφθεί ότι μπορεί να οδηγηθεί σε επαναληπτική εκτέλεση αυτού. Ωστόσο, η ύπαρξη μιας συνάρτησης που εκτελείται όταν υπάρξει πρόβλημα στις εξωτερικές κλήσεις μπορεί να δημιουργήσει τέτοια προβλήματα. Ο τύπος συνάρτησης αυτός ονομάζεται fallback function, και καλείται όταν το smart contract στο οποίο έγινε η κλήση επιστρέφει λανθασμένα αποτελέσματα ή δεν έχει gas για να επιστρέψει. Αν η fallback function καλεί απλώς ξανά το ίδιο smart contract, η διαδικασία αυτή θα γίνει επαναληπτική. Αν δεν υπάρχουν σωστοί έλεγχοι κατά τις αποστολές ether, μια τέτοια περίπτωση μπορεί να ζητήσει ξανά και ξανά την αποστολή μιας συναλλαγής, μέχρι το αρχικό contract να ξεμείνει από ether ή να φτάσει το block gas limit λόγω της αναδρομής.

### 4.3.2 Ατέρμονες βρόγχοι

Λόγω του μέγιστου αριθμού gas ανά block, και του γεγονότος ότι η Solidity είναι Turing-Complete γλώσσα, ένας ατέρμων βρόγχος θα εξαντλούσε το gas της συναλλαγής και χωρίς να καταφέρει τίποτα, το smart contract θα επιστρέψει στην αρχική του κατάσταση μετά την εκτέλεση. Αν το smart contract συνεχίσει να μπαίνει σε αυτόν τον ατέρμων βρόγχο, κάποια στιγμή θα εξαντληθούν όλα του τα κεφάλαια και θα είναι αχρησιμοποίητο. Αυτός είναι και ο λόγος για την ύπαρξη του gas, το να μην εκτελούνται άσκοπα υπολογιστικές διαδικασίες, ωστόσο είναι κάτι που πρέπει να έχει στο νου του ο κάθε developer, ακόμα και για πολύ μεγάλους, αν και όχι , βρόγχους.

### 4.3.3 Έλλειψη Μυστικότητας

Στην Solidity υπάρχουν μεταβλητές και δημοσίου αλλά και ιδιωτικού τύπου. Οι μεταβλητές ιδιωτικού τύπου είναι κρυφές, ωστόσο, όταν ένας χρήστης αλληλεπιδρά

με ένα smart contract η συναλλαγή, καθώς και τα ορίσματα που αποστέλλει είναι δημόσια, και έτσι, οποιοσδήποτε, με τη γνώση του δημόσια διαθέσιμου κώδικα του smart contract και τα ορίσματα αυτά, μπορεί να εξάγει τις τιμές της ιδιωτικής μεταβλητής. Πρέπει συνεπώς να χρησιμοποιούνται ειδικοί αλγόριθμοι που θα μεταβάλλουν τη τιμή με βάση χρονικά δεδομένα (λαμβάνοντας φυσικά τις επιπτώσεις του 3.1.1.) ή άλλες μεθόδους, ώστε να διατηρηθεί η εχεμύθεια.

## 4.4 Ιστορικές επιθέσεις και αδυναμίες γνωστών smart contracts

Πολλές από τις παραπάνω αδυναμίες έχουν γίνει αντικείμενο εκμετάλλευσης από επιτιθέμενους με στόχο την κλοπή ether από smart contracts ή την παύση της λειτουργίας αυτών. Σε αυτή την ενότητα θα εξετάσουμε μερικά από τα πιο διάσημα από αυτά με σκοπό να δείξουμε το ρόλο που είχαν όσα εξετάσαμε μέχρι στιγμής, και να εμφανίσουμε πιθανές λύσεις [**?**].

### 4.4.1 The DAO

Το 'The DAO' ήταν μια από τις πρώτες και μεγαλύτερες προσπάθειες smart contracts στην πλατφόρμα Ethereum. Ο στόχος του ήταν να λειτουργήσει ως επενδυτικό κεφάλαιο, χρηματοδοτώντας την λειτουργία άλλων dapps με αντάλλαγμα ένα μερίδιο από τα κέρδη τους, το οποίο θα διένημε στους μετόχους του. Κατάφερε να μαζέψει περίπου 150 εκατομμύρια δολάρια σε ether κατά την εκκίνηση του, και ο επιτιθέμενος καταταφέρει να κλέψει περίπου 60 εκατ. δολάρια μέσω της αδυναμίας που θα αναλύσουμε. Η κοινότητα του Ethereum κατάφερε να αναιρέσει την κλοπή μέσω μιας διακλάδωσης της blockchain στην οποία η ζημιά αναιρέθηκε.

Θα αναφέρουμε μια απλοποιημένη έκδοση του ευάλωτου κώδικα, στην οποία θα αναδείξουμε τα λάθη που υπήρχαν στην αρχική.

```
1
2 contract SimpleDAO {
3 mapping (address => uint) public credit;
4 function donate(address to){credit[to] += msg.value;}
5 function queryCredit(address to) returns (uint){
6 return credit[to];
7 }
8 function withdraw(uint amount) {
9 if (credit[msg.sender]>= amount) { msg.sender.call.value(amount)()
    ;  credit[msg.sender]-=amount;
```

```
}}}
```

Το παραπάνω είναι μια φυσιολογική μέθοδος με την οποία κάποιος θα έγραφε ένα smart contract στην solidity, το οποίο επιστρέφει στον χρήστη τα tokens που του αντιστοιχούν. Τώρα, κοιτάμε το επιτιθέμενο smart contract :

```
contract Attack {
SimpleDAO public dao = SimpleDAO(0x354...);
address owner;
function Attack(){owner = msg.sender; }
function() { dao.withdraw(dao.queryCredit(this)); }
function getJackpot(){ owner.send(this.balance); }
```

Η συνάρτηση χωρίς όνομα είναι η fallback function, η οποία καλείται κάθε φορά που χρησιμοποιείται μια send ή call από ένα smart contract. Ο επιτιθέμενος λοιπόν δωρίζει κάποια smart contracts και στη συνέχεια καλεί την fallback function του Attack contract. Το attack contract καλεί την withdraw, καθώς όμως η αποστολή των χρημάτων από το The DAO contract γίνεται μέσω της call, καλείται και πάλι η fallback function πρίν το σημείο όπου το ποσό του επιτιθέμενου μειώνεται. Έτσι, μπαίνουμε σε έναν ατέρμονα βρόγχο, ενώ τα κεφάλαια αποστέλλονται κανονικά, καθώς η συνάρτηση call δημιουργεί καινούργιες συναλλαγές εκτός του εκτελούμενου κώδικα. Ο επιτιθέμενος λοιπόν μπορεί να απορροφήσει όλα τα κεφάλαια του smart contract μετά από μια σειρά από blocks, εκτός και αν εκτελεστεί κάποια συνάρτηση αποφυγής από το αρχικό contract.

Η λύση στη συγκεκριμένη επίθεση είναι η εξής υλοποίηση της συνάρτησης call :

```
function withdraw(uint amount) {
 credit[msg.sender]-=amount;
  if (!msg.sender.call.value(amount)()) {
       throw;
} }
```

Δηλαδή το ποσό μειώνεται πριν την αποστολή των κεφαλαίων και άν υπάρξει πρόβλημα κατά την αποστολή, μέσω της throw η αφαίρεση του υπολοίπου του χρήστη αναιρείται. Όπως παρατηρούμε η λύση είναι αρκετά απλή, ωστόσο ήταν πολύ δύσκολη η πρόβλεψη των συνεπειών που μπορεί να έχει η κλήση μιας fallback function από τη πλευρά του επιτιθέμενου στο συγκεκριμένο σημείο. Η αδυναμία εδώ λοιπόν είναι η επανεισαγωγή. Επίσης, μόνο για το παράδειγμα, υπάρχει και η μη ελεγχόμενη μεταφορά κεφαλαίων και στο αρχικό contract, και στο contract του επιτιθέμενου κατα τις κλήσεις call, ενώ η σωστή μορφή είναι αυτή της λύσης, όπου η αποστολή στοιχίζεται από συνθήκη if.

### 4.4.2 King of the Ether Throne

Για την εξερεύνηση των δυνατοτήτων των smart contracts, έχουν στηθεί contracts που υλοποιούν διάφορα οικονομικά μοντέλα καθώς και παίγνια, ένα από τα οποία ήταν το Rubixi, το οποίο υλοποιούσε μια συνειδητή απάτη Ponzi. Οι συμμετέχοντες στο contract κέρδιζαν χρήματα ανάλογα με το πόσους καινούργιους χρήστες μπορούσαν να κάνουν να συμμετέχουν. Επίσης, ο ιδιοκτήτης του contract σύλλεγε κάποια μέρη από τις επενδύσεις των νέων μελών. Το αρχικό όνομα του smart contract ήταν DynamicPyramid, ωστόσο όταν το όνομα άλλαξε, από λάθος των προγραμματιστών δεν άλλαξε και το όνομα της συνάρτησης κατασκευής ( constructor), κάνοντας την προσιτή από τον οποιοδήποτε, αντί να είναι μια συνάρτηση που καλείται μόνο μια φορά κατά τη δημιουργία του smart contract.

```
1
2 contract Rubixi {
3 address private owner;
4 function DynamicPyramid() { owner = msg.sender; }
5 function collectAllFees() { owner.send(collectedFees); }
6 }
```

Συνεπώς, καλώντας την DynamicPyramid(), οποιοσδήποτε μπορούσε να θέσει τον εαυτό του ως owner και να συλλέξει τα κέρδη του αρχικού ιδιοκτήτη. Η αδυναμία εδώ λοιπόν είναι η αμεταβλητότητα του πηγαίου κώδικα, η οποία εμπόδισε την βασική αυτή διόρθωση να εκτελεστεί.

### 4.4.3 Governmental

Το Governmental είναι ένα ακόμα ευάλωτο σχήμα πυραμίδας. Για να συμμετέχει κάποιος, στέλνει ένα ποσό στο smart contract. Αν κανένας δεν συμμετέχει για 12 ώρες, τα χρήματα κερδίζει ο τελευταίος συμμετέχων και ο πίνακας με τους συμμετέχοντες καθαρίζεται.

```
1
2 creditorAddresses = new address[](0);
3 creditorAmounts = new uint[](0);
```

Ο παραπάνω κώδικας καθαρίζει τα κελιά του πίνακα ένα προς ένα, ωστόσο κάποια στιγμή οι συμμετέχοντες έγιναν τόσοι πολλοί που το κόστος καθαρισμού των κελιών σε gas έγινε μεγαλύτερο από το μέγιστο δυνατό για ένα block, και έτσι κανένας περαιτέρω καθαρισμός δεν ήταν δυνατός.

Θα παρουσιάσουμε τώρα ένα απλοποιημένο παράδειγμα του Governmental,

καθώς και τρεις διαφορετικές πιθανές επιθέσεις σε αυτό.

```
1
2
3
4  contract Governmental {
5  address public owner;
6  address public lastInvestor;
7  uint public jackpot = 1 ether;
8  uint public lastInvestmentTimestamp;
9  uint public ONE_MINUTE = 1 minutes;
10 function Governmental() {
11 owner = msg.sender;
12 if (msg.value<1 ether) throw;
13 }
14 function invest() {
15 if (msg.value<jackpot/2) throw;
16 lastInvestor = msg.sender;
17 jackpot += msg.value/2;
18 lastInvestmentTimestamp = block.timestamp;
19 }
20 function resetInvestment() {
21 if (block.timestamp <
22 lastInvestmentTimestamp+ONE_MINUTE)
23 throw;
24
25 lastInvestor.send(jackpot);
26 owner.send(this.balance-1 ether);
27
28 lastInvestor = 0;
29 jackpot = 1 ether;
30 lastInvestmentTimestamp = 0;
31 } }
```

Σε αυτό το παράδειγμα, νικητής του παιχνιδιού είναι αυτός που θα έχει κάνει την τελευταία προσφορά αν περάσει ένα λεπτό χωρίς καινούρια προσφορά. Υπάρχει επίσης η συνάρτηση resetInvestment, την οποία μπορεί να καλέσει οποιοσδήποτε και επιστρέφει τα μισά κεφάλαια στον νικητή και τα άλλα μισά στον δημιουργό του contract.

Μια επίθεση που εκμεταλλεύεται το μέγεθος της στοίβας των blocks είναι το εξής smart contract :

```
1
2  contract Attack {
3  function attack(address target, uint count) {
4  if (0<=count && count<1023) this.attack.gas(msg.gas-2000)(target,
       count+1);
5  else Governmental(target).resetInvestment();
6  }
```

```
7 }
```

Το contract το υποβάλλει ο δημιουργός του αρχικού Governmental ώστε να εμποδίσει τους υπόλοιπους χρήστες να πληρωθούν. Η επίθεση καλεί αναδρομικά τον εαυτό της μέχρι να φτάσει το μήκος της στοίβας μείον δύο, στη συνέχεια καλεί την reset investment, η οποία όμως δεν καταφέρνει να εκτελέσει τη μεταφορά χρημάτων στον νικητή καθώς η στοίβα εξωτερικών κλήσεων είναι γεμάτη και δεν ελέγχει την μεταφορά, και στη συνέχεια τερματίζει τον γύρο, καθαρίζοντας τα ονόματα των συμμετεχόντων αλλά διατηρώντας όλα τα κεφάλαια της, καθώς δεν πλήρωσε τον νικητή στον προηγούμενο γύρο. Για να αποκτήσει τα κεφάλαια αυτά ο δημιουργός του αρχικού contract, πρέπει απλώς να περιμένει να τερματιστεί ένας γύρος κανονικά.

Ο δεύτερος και τρίτος τύπος επίθεσης μπορεί να εκτελεστεί από έναν εξορύχο ο οποίος παρουσιάζεται και ως παίκτης. Θα μπορούσε να επιλέξει να μην συμπεριλάβει καμία συναλλαγή προς το smart contract εκτός από τη δική του, ώστε να είναι σίγουρα ο νικητής, ή να επηρεάσει το timestamp του επόμενου block ώστε να κοροϊδέψει το smart contract ότι πέρασε παραπάνω χρόνος από την τελευταία συναλλαγή. Και στις δύο περιπτώσεις θα πρέπει ο συγκεκριμένος εξορύχος να είναι αρκετά τυχερός ώστε να είναι το δικό του block αυτό που θα συμπεριληφθεί, αντί κάποιου από χιλιάδες άλλους.

Οι αδυναμίες που ήταν εμφανείς εδώ είναι αυτές του ορίου gas ανα block, του μη ελέγχου της αποστολής κεφαλαίων, της απρόβλεπτης κατάστασης, της μη ύπαρξης σταθερών έμπιστων χρονικών τιμών και της επανεισαγωγής.

## 4.5 Συμπεράσματα

Παρατηρούμε πως η ανάπτυξη των smart contracts και η χρήση κώδικα για τις συναλλαγές μέσω blockchain τεχνολογιών είναι ακόμα σε πρώιμο στάδιο και σε ορισμένες περιπτώσεις η κλοπή ακόμα και μεγάλων χρηματικών ποσών είναι πιθανή [?]. Το γεγονός ότι τα προγράμματα αλληλεπιδρούν με άλλα αυτόνομα προγράμματα τα οποία μπορεί να έχουν δικές τους απρόβλεπτες συμπεριφορές περιπλέκει ακόμα περισσότερο την πρόβλεψη λαθών και αδυναμιών. Ωστόσο, ο τομέας μαθαίνει γρήγορα από τέτοια λάθη και σύντομα δημιουργεί κοινές πρακτικές και οδηγίες για την αποφυγή τους στο μέλλον. Ήδη οι περισσότεροι compilers προειδοποιούν για της αδυναμίες που περιγράφηκαν σε αυτό το κεφάλαιο και η EVM συνεχώς αναπτύσσεται με νέες λειτουργίες για να καλύψει τις υπάρχουσες ελλείψεις [?]. Όσοι αναπτύσσουν όμως smart contracts είναι απαραίτητο να

έχουν τις παραπάνω αδυναμίες, καθώς και πολλές άλλες που μπορεί να προκύψουν υπόψιν και να εκτελούν ενδελεχείς ελέγχους πριν ανεβάσουν το πρόγραμμα τους στην κύρια αλυσίδα.

# Κεφάλαιο 5

# Σχεδίαση Εφαρμογής

## 5.1 Εισαγωγή

Το περιβάλλον του Ethereum και οι δυνατότητες των smart contracts ενδείκνυνται για την υλοποίηση εφαρμογών που διαχειρίζονται μεταφορές αξίας μέσω tokens , με ασφαλή τρόπο, χωρίς την ύπαρξη εμπιστοσύνης μεταξύ των συναλλασόμενων. Είναι ιδιαίτερα χρήσιμο επίσης για την υλοποίηση αρχικών δημόσιων προσφορών tokens ( Initial Coin Offerings, ICOs ). Αυτή είναι η διαδικασία μέσω της οποίας μια dapp προσφέρει δημόσια τα tokens που χρησιμοποιεί η εφαρμογή της με αντάλλαγμα ether, τα οποία στη συνέχεια χρησιμοποιούνται για την κάλυψη του κόστους gas της εφαρμογής και την χρηματοδότηση της περαιτέρω ανάπτυξης της. Μια ακόμα χρήση είναι η υλοποίηση εκστρατείων χρηματοδότησης μέσω κοινού (crowdfunding) , καθώς είναι εύκολη η πληρωμή προς μια διεύθυνση η οποία θα λειτουργήσει με προβλέψιμο τρόπο από τον κώδικα που περιλαβάνει το smart contract που την υλοποιεί.

Μία τέτοια crowdfunding εφαρμογή θα υλοποιηθεί στο πλάισιο του πρακτικού κομματιού αυτής της διπλωματικής εργασίας, η οποία θα στοχεύει στην διευκόλυνση της χρηματοδότησης φιλανθρωπικών έργων.

## 5.2 Λόγοι επιλογής

Ο βασικός λόγος επιλογής ενός τέτοιο θέματος είναι η πίστη του συγγραφέα ότι οι τεχνολογίες blockchain μπορούν και πρέπει να έχουν μια θετική επίδραση

στο κοινωνικό περιβάλλον μέσω της υλοποίησης εύκολων και ασφαλών μεταφορών χρημάτων. Η ανάπτυξη τέτοιων χρηστικών και βοηθητικών εφαρμογών είναι ο καλύτερος τρόπος για την κεντρική υιοθέτηση και ανάπτυξη του οικοσυστήματος που είναι τα ψηφιακά νομίσματα αυτή τη στιγμή, σε αντίθεση με το ρεύμα της υιοθέτησης τους χάρη στην άνοδο της χρηματικής τους αξίας, η οποία δίνει ενδείξεις οικονομικής φούσκας και αποθαρρύνει την ευρύτερη υιοθέτηση τους.

Ο δεύτερος λόγος για την επιλογή είναι το γεγονός ότι η διαχείριση των κεφαλαίων δωρεάς μέσω των smart contracts μπορεί να εξασφαλίσει διαφάνεια στο πως αυτά θα χρησιμοποιηθούν, αποτρέποντας φαινόμενα κατάχρησης [**?**]. Τέλος, ένας ακόμα λόγος είναι ότι αυτή τη στιγμή, τεράστια χρηματικά ποσά κινούνται μεταξύ smart contracts στο οικοσύστημα, καθώς η συνολική αξία των κρυπτονομισμάτων έχει ξεπεράσει τα 460 δισεκατομμύρια δολάρια τη στιγμή της συγγραφής αυτού του κειμένου και αυτή του Ethereum τα 84 δις [**?**]. Σύντομα, πολλές dapps θα περιλαμβάνουν ένα κομμάτι εταιρικής κοινωνικής ευθύνης στα smart contracts τους, μέσω του οποίου πιθανών να μοιράζονται ένα κομμάτι των κερδών τους για φιλανθρωπικούς σκοπούς , οπότε θα ήταν ευκαιρία για την εφαρμογή μας να καλύψει αυτή την ανάγκη, προσφέροντας μια διαυγή και έμπιστη μέθοδο χρηματοδότησης.

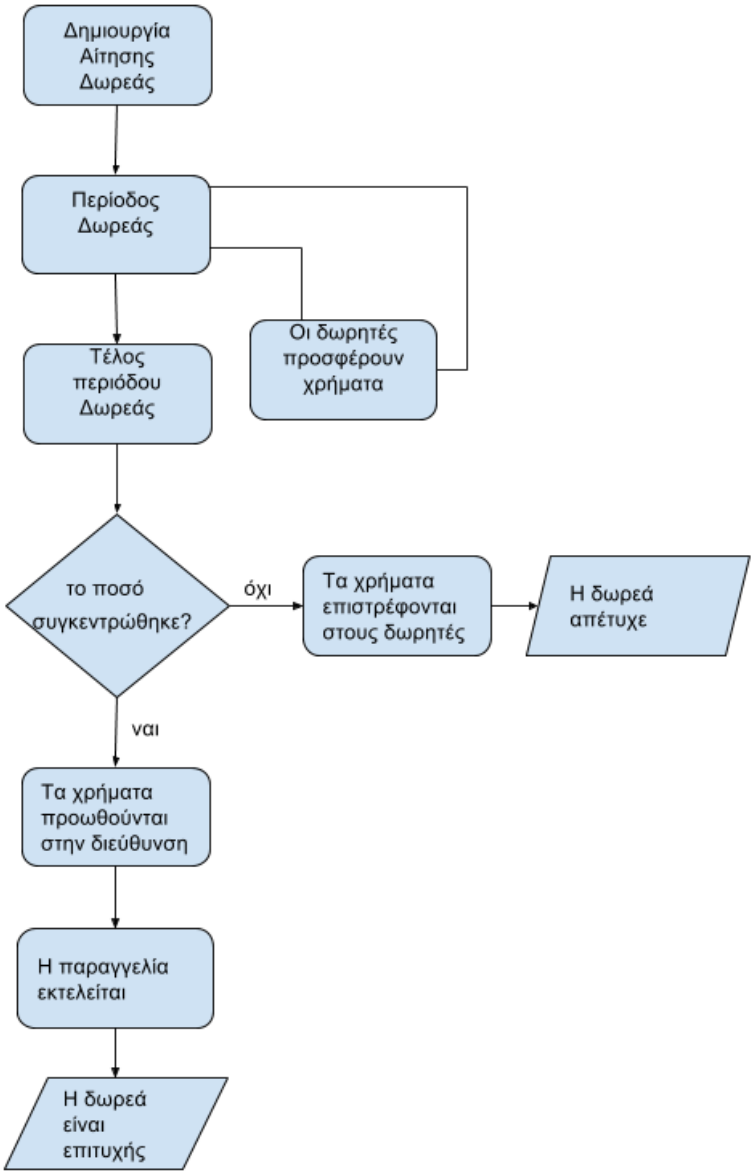
## 5.3 Περιγραφή λειτουργίας

Η αρχική ανάπτυξη της εφαρμογής μας στα πλαίσια της διπλωματικής αυτής εργασίας θα είναι μια υλοποίηση των βασικών λειτουργιών της ως απόδειξη της έννοιας ( proof of concept). Η εφαρμογή μας θα λειτουργεί ως ένας δεσμευμένος λογαριασμός ( escrow account ), ο οποίος θα εκτελεί την μεταφορά χρημάτων όταν έχει συγκεντρωθεί ολόκληρο το απαιτούμενο ποσό εντός κάποιου χρονικού ορίου, ενώ αν δεν καταφέρει να συγκεντρωθεί τα χρήματα θα επιστρέφονται στους δωρητές. Οι φιλανθρωπικές οργανώσεις θα μπορούν να εισάγουν πληροφορίες για το τι χρειάζονται καθώς και την διεύθυνση στην οποία θα πρέπει να γίνει η πληρωμή για την παραγγελία τους. Σε πρώτη φάση, η εφαρμογή θα επιτρέπει την αγορά πρώτων υλών από τις διάφορες ψηφιακές αγορές προιόντων που έχουν υλοιποιθεί, βοηθώντας παράλληλα την ανάπτυξη του ψηφιακού εμπορίου. Το front end της εφαρμογής θα εμφανίζει στους χρήστες τις διάφορες χρηματοδοτικές ανάγκες, τους δημιουργούς τους, τα ποσά που έχουν συγκεντρωθεί και τον χρόνο που απομένει μέχρι τη λήξη της χρηματοδότησης. Θα δίνει επίσης τη δυνατότητα στους ενδιαφερόμενους δωρητές να αποστείλουν χρήματα στην φιλανθρωπία της επιλογής τους.

## 5.4 Λειτουργικές Απαιτήσεις

1. Το σύστημα θα πρέπει να εμφανίζει τις διαθέσιμες αιτήσεις δωρεάς.
2. Οι χρήστες θα πρέπει να μπορούν να χρηματοδοτήσουν τις αιτήσεις δωρεάς της επιλογής τους.
3. Οι φιλανθρωπικοί οργανισμοί ( Φ.Ο.) θα πρέπει να μπορούν να υποβάλλουν αιτήσεις δωρεάς.
4. Οι αιτήσεις δωρεάς θα πρέπει να περιλαμβάνουν τον λογαριασμό πληρωμής.
5. Οι αιτήσεις δωρεάς θα πρέπει να περιλαμβάνουν το ζητούμενο ποσό.
6. Οι αιτήσεις δωρεάς θα πρέπει να περιλαμβάνουν τον διαθέσιμο χρόνο πληρωμής.
7. Οι αιτήσεις δωρεάς θα πρέπει να περιλαμβάνουν πληροφορίες για τον Φ.Ο. που τις υπέβαλλε.
8. Το σύστημα θα πρέπει να μεταφέρει τα χρήματα στο λογαριασμό όταν το ποσό πληρωμής συσσωρευτεί πρίν τη λήξη του διαθέσιμου χρόνου.
9. Το σύστημα θα πρέπει να επιστρέφει τα χρήματα στους χρήστες όταν το το ποσό πληρωμής δεν συσσωρευτεί πρίν τη λήξη του διαθέσιμου χρόνου.
10. Το σύστημα θα πρέπει να μπορεί να προτείνει αιτήσεις δωρεάς σύμφωνα με διάφορες μετρικές.

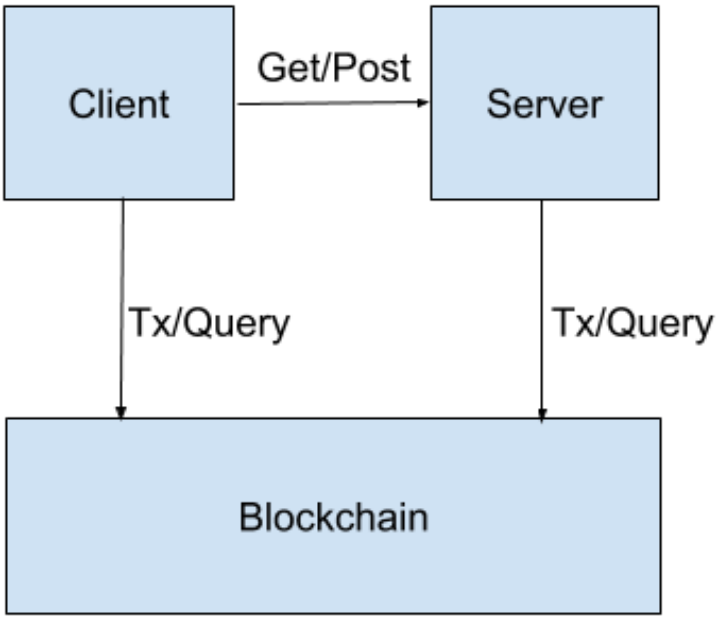
## 5.5 Διάγραμμα Λειτουργίας




Σχήμα 5.1: Διάγραμμα Ροής Εφαρμογής [?]

## 5.6 Διαφορές με την κλασσική σχεδίαση λογισμικού

Η ανάλυση των smart contracts επιφέρει νέες προκλήσεις σε σχέση με την παραδοσιακή σχεδίαση της εφαρμογής : Οι παραδοσιακές εφαρμογές πελάτη-εξυπηρετητή (client-server) πλέον έχουν ένα τρίτο στοιχείο, την blockchain.




Ανάλογα με τα ποσοστά χρήσης του καθενός από τα παραπάνω στοιχεία, προκύπτουν διαφορετικές κατηγορίες εφαρμογών, οι οποίες θα αναλυθούν.

### 5.6.1 Εφαρμογές client-blockchain χωρίς server

Η πρότυπη έκδοση μιας εφαρμογής Ethereum είναι τέτοια ώστε να μην περιέχει server, με ολόκληρη τη ροή της εφαρμογής να συμβαίνει μεταξύ client και blockchain. Το πρώτο βήμα εδώ είναι η μετάδοση του κώδικα του client στους χρήστες. Η πιο απλή λύση είναι μια στατική ιστοσελίδα με μια εφαρμογή ενεργοποιημένης web3 βιβλιοθήκης. Σε περίπτωση που μπορούμε να βασιστούμε ότι οι clients έχουν υποστήριξη για τα πρωτόκολλα bzz ή ipfs , μπορεί να γίνει διάδοση μέσω Swarm ή IPFS για πλήρη αποκέντρωση.

Στην συνέχεια, η εφαρμογή πρέπει να μπορεί να διαβάζει πληροφορίες από τη blockchain, το οποίο απαιτεί σύνδεση σε έναν ενεργό κόμβο Ethereum, το οποίο θα υλοποιηθεί από τον προμηθευτή web3, ο οποίος προστίθεται μέσω προγραμμάτων όπως Mist και Metamask που αναφέρθηκαν στο δεύτερο κεφάλαιο. Αυτές οι

εφαρμογές αναλαμβάνουν επίσης και την αποστολή συναλλαγών εκ μέρους των clients, ζητώντας από τον χρήστη να επιβεβαιώσει τη συναλλαγή. Εναλλακτικά, μπορεί να ζητηθεί από τον χρήστη να αποστείλει χειροκίνητα ποσά σε eth στην επιθημητή διεύθυνση μαζί με τα απαραίτητα δεδομένα για τη λειτουργία της εφαρμογής χρησιμοποιώντας τη μέθοδο της επιλογής του.

Τέλος ο έλεγχος για την κατάσταση των δεδομένων του χρήστη γίνεται μέσω ανάγνωσης events από την blockchain και ανανέωσης του client [?].

### 5.6.2 Εφαρμογές server-blockchain χωρίς client

Τέτοιου είδους εφαρμογές μπορεί να προσφέρουν υπηρεσίες στην blockchain όπως scripts, batch processes και διασυνδεσιμότητα με εφαρμογές internet of things που δεν απαιτούν άμεση αλληλεπίδραση με ανθρώπινους χρήστες. Η πρώτη λύση είναι η πιο απλή :

Στήνουμε έναν τοπικό κόμβο Ethereum και χρησιμοποιούμε τη διασυνδεσιμότητα JSON RPC που διαθέτει μέσω της εφαρμογής μας ωστε να εκτελέσουμε όλες τις εντολές σχετικά με τη blockchain. Χρήσιμο είναι να διαθέτουμε έναν ξεκλείδωτο λογαριασμό για την εκτέλεση συναλλαγών από την εφαρμογή, ωστόσο θα πρέπει να φροντίσουμε να μην υπάρχει εξωτερική πρόσβαση στην εφαρμογή μας, καθώς θα μπορούσε να οδηγήσει σε απώλεια των κεφαλαίων της.

Μια παρόμοια, εναλλακτική πρόταση είναι η εφαρμογή μας να είναι εκτός δικτύου, να υπογράφει συναλλαγές και να τις μεταδίδει σε ένα δημόσιο κόμβο. Αυτό είναι χρήσιμο σε περιπτώσεις που δεν υπάρχει υποδομή για τη συνεχή λειτουργία ενός ιδιωτικού κόμβου. Σε αυτή την περίπτωση όμως, υπάρχει τυφλή εμπιστοσύνη στον δημόσιο κόμβο, ο οποίος ενώ δεν μπορεί να μεταβάλλει τις συναλλαγές που υποβάλλονται, μπορεί να επιλέξει να μην τις μεταδώσει στο δίκτυο ή να παρέχει ψευδείς απαντήσεις σε διερωτήσεις. Μπορεί να υπάρξει προστασία ενάντια σε αυτά τα προβλήματα μέσω προώθησης σε πολλαπλούς δημόσιους κόμβους, ωστόσο περιπλέκει τον κώδικα της εφαρμογής.

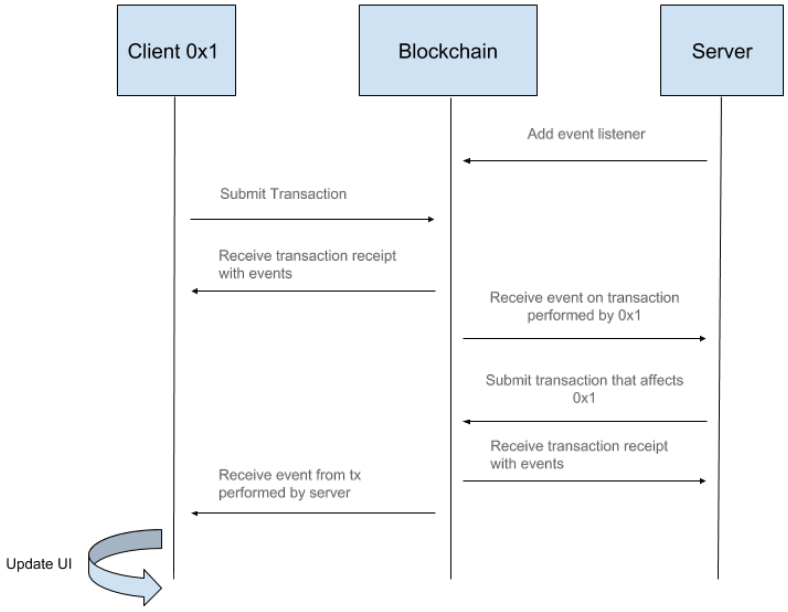
### 5.6.3 Εφαρμογές client-server-blockchain

Μια τέτοια υλοποίηση μας επιτρέπει να χρησιμοποιήσουμε όλες τις δυνατότητες της blockchain αποφεύγοντας πολλά από τα μειονεκτήματα της ανάπτυξης smart contracts. Για παράδειγμα, δεδομένα με μεγάλο όγκο μπορούν να αποθηκεύονται

στην πλευρά του server με παραδοσιακές μεθόδους και να επαληθεύεται η ακρίβεια τους μέσω αποθήκευσης του αποτυπώματος κατακερματισμού τους (hash) στη blockchain, εξοικονωμόντας ether. Επίσης, ο server προσφέρει διασύνδεση με εκτός blockchain υπηρεσίες όπως να παρακολουθεί δεδομένα, να αποστέλλει emails ή να χρησιμοποιεί λογισμικό από τρίτους.

Η κύρια πρόκληση αυτής της υλοποίησης είναι ο συγχρονισμός των αλληλεπιδράσεων client και server με τη blockchain. Για παράδειγμα, μπορεί να απαιτείται ο server να αντιδράσει σε μια δράση του client στη blockchain , ή να γίνει η οπτικοποίηση αποτελεσμάτων μιας ενέργειας του server στην blockchain από τη πλευρά του client. Αυτό τυπικά γίνεται με τη χρήση events, όπου οι clients παρακολουθούν events που αφορούν αυτούς συγκεκριμένα, ενώ οι servers παρακολουθούν κάθε σχετικό event ή ακόμα και συγκεκριμένες συναλλαγές. Εναλλακτικά, οι clients θα μπορούσαν να υποβάλλουν τον κωδικό μιας συναλλαγής τους άμεσα στον server, ωστόσο μόνο ως ειδοποιήσεις και όχι ως πηγές αλήθειας, καθώς κακοήθεις χρήστες θα μπορούσαν να παρακολουθούν τις συναλλαγές επί της αλυσίδας και να υποβάλλουν κωδικούς που δεν είναι δικοί τους. Σε κάθε περίπτωση, η αντίδραση στις συναλλαγές που εκτελούνται από τη blockchain πρέπει να γίνεται μετά από έναν λογικό αριθμό επαληθεύσεων καθώς μπορεί να υπάρξει αναδιοργάνωση της αλυσίδας. Η χρήστες ωστόσο, θα πρέπει να ενημερώνονται πως η συναλλαγή τους παρατηρήθηκε.

Όσον αφορά τη δική μας εφαρμογή, καθώς δεν υπάρχει ανάγκη αποθήκευσης μεγάλου όγκου δεδομένων ή επικοινωνία με εφαρμογές εκτός αλυσίδας, θα υλοποιηθεί ως μια εφαρμογή client-blockchain.

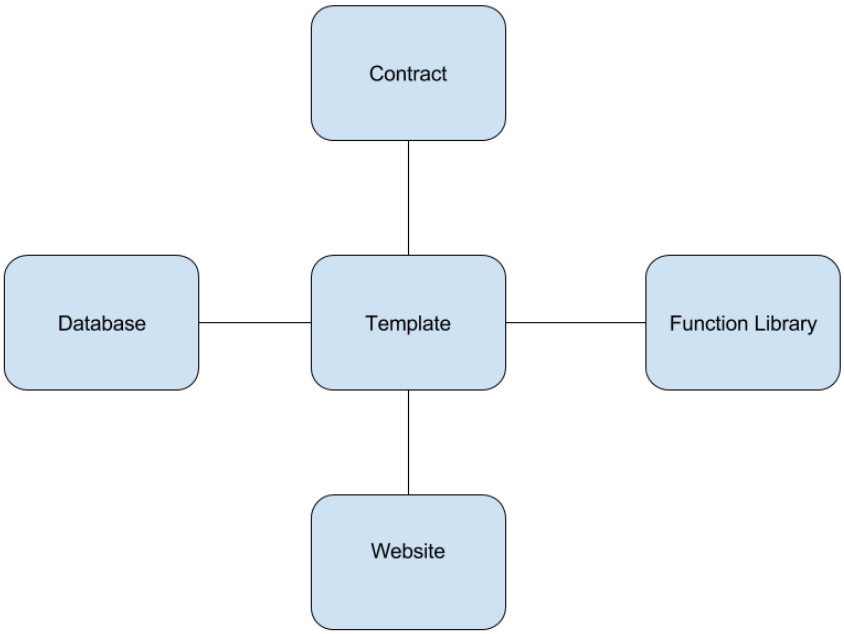



## 5.7 Δυνατότητα update

Για να μπορεί ένα smart contract εφαρμογής τύπου client-blockchain να αλλάζει εύκολα από τη μία έκδοση στην επόμενη, δεδομένου ότι μπορεί να προστεθούν πολλές νέες λειτουργίες στο μέλλον, είναι συνετό να ακολουθεί αυτή τη δομή [?] :

- Τα δεδομένα να αποθηκεύονται σε ένα ξεχωριστό smart contract, το οποίο θα επιτρέπει μόνο την λήψη και την αποθήκευση νέων δεδομένων.
- Οι συναρτήσεις που επιδρούν στα δεδομένα να αποθηκεύονται σε ένα smart contract τύπου βιβλιοθήκης, όπως αναφέραμε στο Κεφάλαιο 2.
- Ένα smart contract τύπου interface να λειτουργεί ώς ενδιάμεσο για τις επικοινωνίες των υπολοίπων.
- Το κεντρικό smart contract που θα επιτελεί τη λειτουργία

Συνεπώς, μια πλήρως ανανεώσιμη έκδοση [?] του dapp που θα αναπτυχθεί θα είχε την εξής μορφή :




**Database**: Περιέχει τα δεδομένα της εφαρμογής. Συγκεκριμένα :

- Την διεύθυνση όπου πρέπει να γίνει η κάθε πληρωμή από τις τρέχουσες αιτήσεις.
- Το ποσό που έχει μαζευτεί για την κάθε τρέχουσα αίτηση.
- Το ποσό που πρέπει να μαζευτεί για την κάθε τρέχουσα αίτηση.
- Τον χρόνο που απομένει για την ολοκλήρωση της περιόδου δωρεάς.

**Function Library** : Περιέχει τις συναρτήσεις για την εφαρμογή αλλαγών πάνω στα δεδομένα.Δηλαδή συναρτήσεις :

- Εισαγωγής νέων αιτήσεων με όλα τα χαρακτηριστικά τους.
- Διαγραφής αιτήσεων.
- Αποστολής/ Λήψης ether.
- Ενημέρωσης δεδομένων.

**Template** : Η οντότητα template είναι υπεύθυνη να παραπέμπει στις τελευταίες εκδόσεις των smart contracts που υλοποιούν τις οντότητες database, function library, contract. Περιέχει :

- Τις μεταβλητές που αποθηκεύουν τις τρέχουσες διευθύνσεις των smart contracts.
- Τις συναρτήσεις που μεταβάλλουν τις τρέχουσες διευθύνσεις των smart contracts
- Συναρτήσεις που επιστρέφουν τις αποθηκευμένες μεταβλητές.

**Contract**: Η οντότητα contract υλοποιεί το διάγραμμα ροής της εφαρμογής, χρησιμοποιώντας τα δεδομένα της database και τις συναρτήσεις της library function. Επίσης, διατηρεί τα ether της εφαρμογής έως ότου να κάνει τις πληρωμές ή τις επιστροφές. Περιλαμβάνει λοιπόν και μια συνάρτηση εξόδου ασφαλείας ( escape hatch ).

**Website**: Η οντότητα website περιλαμβάνει το front end της εφαρμογής το οποίο επικοινωνεί με το smart contract μέσω web3.js ώστε να λάβει τις απαραίτητες πληροφορίες και να τις εμφανίσει με κατάλληλο τρόπο. Επίσης, βοηθάει στην σύνταξη υποβολών αιτήσεων και στις πληρωμές προς το smart contract. Αποτελεί το front end της εφαρμογής.

Ωστόσο, καθώς οι κλήσεις εξωτερικών smart contracts κοστίζουν περισσότερο gas από την κλήση εσωτερικών συναρτήσεων, υπάρχει μια αντίστροφη σχέση μεταξύ υλοποίησης προσανατολισμένης στην ανανεωσιμότητα και υλοποίησης προσανατολισμένης στην ελαχιστοποίηση κόστους λειτουργίας. Για αυτό το λόγο, οι παραπάνω ενότητες στην εφαρμογή μας, θα ενσωματωθούν σε ένα μοναδιαίο smart contract ως επιμέρους λογικές μονάδες με τη λειτουργία τους να εκτελείται μέσω εσωτερικών συναρτήσεων.

# Κεφάλαιο 6

# Ανάπτυξη εφαρμογής

Η αρχική υλοποίηση θα κάνει τα εξής: Για κάθε εκστρατεία πληρωμής θα δέχεται μια διεύθυνση δωρεάς, χρόνο διάρκειας και το επιθυμητό ποσό μαζί με συμπληρωματικες πληροφορίες όπως το όνομα της εκστρατείας και έναν μικρό σχολιασμό. Για το χρονικό διάστημα που έχει υποβληθεί θα δέχεται καταθέσεις, οι οποίες δεν μπορούν να ξεπεράσουν το επιθυμητό ποσό και όταν αυτό συμπληρωθεί, θα τις προωθήσει στην διεύθυνση δωρεάς, χαρακτηρίζοντας την εκστρατεία ως επιτυχημένη. Εάν δεν συμπληρωθεί μέχρι τη λήξη της διάρκειας, το ποσό εισφοράς θα επιστραφεί στους καταθέτες και η εκστρατεία θα χαρακτηριστεί ως αποτυχημένη. Το περιβάλλον διεπαφής της εφαρμογής θα εμφανίζει τις διαθέσιμες εκστρατείες ανάλογα με την κατάσταση τους και με όλες τις σχετικές πληροφορίες. Επίσης θα επιτρέπει τη συνεισφορά σε κάποια εκστρατεία και τη δημιουργία νέας εκστρατείας

Το υψηλό επίπεδο της γλώσσας solidity επιτρέπει στο contract μας να είναι απλό και να επαληθεύεται εύκολα η λογική της εφαρμογής μας. Η διαφάνεια εξασφαλίζεται από την ντετερμινιστική λειτουργία του συμβολαίου. Οι υποβάλλοντες εκστρατεία είναι υπεύθυνοι να αποδείξουν ότι πρόκειται όντως για φιλανθρωπική οργάνωση και ότι η διεύθυνση ανήκει σε κάποιον αξιόπιστο πάροχο προμηθειών ή υπηρεσιών. Αυτό μπορεί εύκολα να γίνει μέσω άλλων αποκεντρωμένων εφαρμογών όπως η OpenBazaar [**?**]. Οι δωρητές είναι επίσης υπεύθυνοι να ελέγχουν την αξιοπιστία των πωλητών, ωστόσο ενέχουν μειωμένο ρίσκο καθώς η δωρεά τους θα επιστραφεί σε περίπτωση αποτυχίας της εκστρατείας. Το μόνο έξοδο τους είναι το τέλος συναλλαγών του δικτύου μέσω της χρήσης gas, το οποίο όμως είναι κοινό χαρακτηριστικό όλων των εφαρμογών του δικτύου Ethereum και με τις μελλοντικές επεκτάσεις αναμένεται να μειωθεί σε απειροελάχιστο ποσό. Ακολουθεί αναλυτική περιγραφή του smart contract της εφαρμογής

## 6.1 Κώδικας του Smart Contract

Θα εξετάσουμε την τελική έκδοση του smart contract,η οποία είναι η εξής [?]
:

```
$pragma solidity ^0.4.18;

contract CrowdfundedCharity {

    address private owner;
```

Η διεύθυνση του δημιουργού του smart contract αποθηκεύται και χρησιμοποιείται για λειτουργίες διαχείρισης του

```
    event NewFunder(
      address addr,
      uint amount,
      uint campaignID
    );

    event NewCharity(
      address beneficiary,
      uint fundingGoal,
      uint fundingDeadlineBlock
      );
```

Οι δυο αυτές συναρτήσεις τύπου event όταν καλούνται ενημερώνουν το περιβάλλον διεπαφής μας για την ύπαρξη νέου δωρητή και τη δημιουργία νέας εκστρατείας αντίστοιχα [?].

```
    struct Funder {
        address addr;
        uint amount;
        uint campaignID;
    }

    struct Campaign {
        address beneficiary; // This will be the adress to be paid
            for the order
        uint id; //This is different from campaign ID, used to
           identify each campaign from the          //frontend
        uint fundingGoal;
        uint fundingDeadlineBlock;
        uint numFunders;
        uint amount;
        bytes32 name;
        string description;
        bool isActive;
        mapping (uint => Funder) funders;
```

```
18      }
```

Οι δύο αυτές δομές αποθηκεύουν τις απαραίτητες μεταβλητές της εφαρμογής. Η μεταβλητή τύπου mapping funders αντιστοιχίζει τους δωρητές της κάθε εκστρατείας σε έναν αύξων αριθμό.

```
1       uint public numCampaigns = 0;
2       bytes32 public version;
3       mapping (uint => Campaign) public campaigns;
```

Οι παραπάνω μεταβλητές ορίζονται ως public ώστε ο compiler της Solidity να δημιουργήσει αυτόματα συναρτήσης όψης τους ώστε να είναι διαθέσιμες στο περιβάλλον διεπαφής. Η mapping campaigns μεταβλητή είναι η πιο σημαντική για την εφαρμογή μας καθώς διατηρεί όλες τις εκστρατείες.

```
1       modifier isAdmin(){
2         require(msg.sender == owner);
3         _;
4       }
5
6       bool private stopped = false;
7
8       function toggleContractActive() isAdmin public {
9         stopped = !stopped;
10      }
11
12      modifier stopInEmergency { if(!stopped) _;}
```

Τα παράπάνω είναι η δημιουργία κατάλληλων modifiers για τη δημιουργία μιας μεθόδου παύσης της λειτουργίας της εφαρμογής. Οι modifiers προστίθενται μετά τον ορισμό κάθε συνάρτησης, και εκτελούν κάποιο έλεγχο στα στοιχεία του προγράμματος και της κλήσης. Στην συγκεκριμένη περίπτωση, οι κλήσεις με το modifier isAdmin μπορούν να κληθούν μόνο από τον δημιουργό του smart contract. Μέσω αυτού, μπορεί να ενεργοποιηθεί ο modifier stopInEmergency, ο οποίος θα διακόπτει τη δυνατότητα δημιουργίας νέων εκστρατειών και της συμμετοχής. Έτσι, σε περίπτωση ανίχνευσης κάποιου σφάλματος στη λειτουργία του smart contract, ο δημιουργός θα μπορεί να προστατέψει ανυποψίαστους χρήστες από σφάλματα.

```
1       //simple constructor
2       function CrowdfundedCharity (bytes32 version_number) public {
3       version = version_number;
4       owner = msg.sender;
5       }
```

Ο constructor του προγράμματος καλείται μόνο κατα τη δημιουργία και ορίζει τις μεταβλητές version και owner.

```
1    function newCampaign(address beneficiary, uint goal, uint
         deadline, bytes32 name, string description)
         stopInEmergency public returns (uint campaignID) {
2        // deadline is given in days. Using the average number of
             blocks per day
3        // we calculate how many blocks the campaign will be valid
              for.
4        numCampaigns++;
5        campaignID = numCampaigns;
6        uint deadlineBlock = block.number + mul(deadline, 5760);
7        // campaignID is the return variable
8        // Creates new struct and saves in storage. We leave out
             the mapping type.
9        campaigns[campaignID] = Campaign(beneficiary, campaignID ,
              goal, deadlineBlock, 0, 0, name, description,true);
10       NewCharity(beneficiary, goal, deadlineBlock);
11       return campaignID;
12   }
```

Η συνάρτηση δημιουργίας εκστρατείας παίρνει τις κατάλληλες πληροφορίες διεύθυνσης, ποσού, διάρκειας και περιγραφής και συμπληρώνει τις υπόλοιπες που είναι απαραίτητες για τη λειτουργία, όπως το campaignID, και η μεταβλητή state. Όπως αναφέραμε στο κεφάλαιο 3, η μέτρηση χρόνου στην blockchain είναι προβληματική καθώς οι πληροφορίες χρόνου των blocks δεν είναι αξιόπιστες. Ωστόσο, ο ρυθμός δημιουργίας είναι αξιόπιστος και εύκολα μετρήσιμος. Για τη μέτρηση χρόνου λοιπόν, πολλαπλασιάζουμε την επιθυμητή διάρκεια με τον μέσο αριθμό blocks ανα ημέρα και την προσθέτουμε στον αριθμό του πιο σύγχρονου block. Έτσι ο έλεγχος για το πέρας της διάρκειας της εκστρατείας γίνεται μέσω σύγκρισης των αριθμών του τρέχων block και του deadline block. Η συνάρτηση είναι δημόσια, δηλαδή μπορεί να κληθεί από οποιονδήποτε, και μπορεί να απενεργοποιηθεί μέσω του modifier stopInEmergency σε περίπτωση προβλήματος.

```
1   function totalFundsFor(uint8 campaignID) view public returns (
        uint){
2       require(campaignID <= numCampaigns);
3       return campaigns[campaignID].amount;
4   }
5
6    function contribute(uint campaignID) stopInEmergency public
        payable {
7
8       Campaign storage c = campaigns[campaignID];
9       // Creates a new temporary memory struct, initialised with
            the given values
10      // and copies it over to memory.
11      require(campaignID <= numCampaigns);
12      require( c.fundingDeadlineBlock > block.number );
13      require( c.amount + msg.value <= c.fundingGoal );
14      require( c.isActive);
15      c.funders[c.numFunders++] = Funder({addr: msg.sender, amount
            : msg.value, campaignID: campaignID});
16      c.amount += msg.value;
17      NewFunder(msg.sender,msg.value,campaignID);
18   }
```

Η συνάρτηση contribute ελέγχει ότι : Η εκστρατεία που δίνεται ως όρισμα υπάρχει, ότι η προθεσμία υποβολής δωρεάς δεν έχει περάσει και ότι το ποσό που δωρίζεται δεν ξεπερνά τον στόχο. Το τελευταίο αποτρέπει την ύπαρξη κώδικα υλοποίησης επιστροφών.

```
1    function mul(uint256 a, uint256 b) internal pure returns (
        uint256) {
2      if (a == 0) {
3        return 0;
4      }
5      uint256 c = a * b;
6      assert(c / a == b);
7      return c;
8    }
```

Η παραπάνω συνάρτηση είναι ένα παράδειγμα μαθηματικής βιβλιοθήκης. Καθώς κατά τη διαδικασία υπολογισμού του deadline block δεν θέλουμε σε καμία περίπτωση να έχουμε υπολογιστικό λάθος, μέσω του ελέγχου assert επιβεβαώνουμε την ακρίβεια του υπολογισμού μας. Αν αυτός αποτύχει για οποιοδήποτε λόγο, η συναλλαγή θα αποτύχει.

```
1    function checkGoalReached(uint campaignID) stopInEmergency
        public returns (bool reached) {
2      Campaign storage c = campaigns[campaignID];
3      if ( c.fundingDeadlineBlock <= block.number) {
4        if (c.amount < c.fundingGoal) {
5          //Charity Failed, Initiate Refunds
6          for(uint i=1; i<c.numFunders; i++){
7            uint refund =c.funders[i].amount;
8            c.funders[i].amount =0;
9            c.funders[i].addr.transfer(refund);
10           c.isActive = false;
11         }
12       }
13
14       if (c.amount >= c.fundingGoal){
15         // Charity Successful, Transfer Amount
16         amount = c.amount;
17         c.amount = 0;
18         c.beneficiary.transfer(amount);
19         c.isActive = false;
20         return true;
21       }
22     }
23     //For testing
24     if (c.amount >= c.fundingGoal){
25       // Charity Successful, Transfer Amount
26       uint amount = c.amount;
27       c.amount = 0;
28       c.beneficiary.transfer(amount);
```

```
29            c.isActive = false;
30            return true;
31        }
32
33        if (c.amount < c.fundingGoal) {
34            return false;
35        }
36    }
```

Η συνάρτηση checkGoalReached ολοκληρώνει τη λειτουργία του smart contract. Εάν η διάρκεια της εκστρατείας έχει περάσει, ελέγχει κατα πόσο συγκεντρώθηκε το ποσό. Σε περίπτωση που το ποσό συγκεντρώθηκε, ολοκληρώνει τη μεταφορά του στη διεύθυνση-στόχο, ενώ εαν δεν συγκεντρώθηκε, επιστρέφεται στον κάθε χρήστη το ποσό συνεισφοράς του.

Παρατηρούμε ότι το ποσό του κάθε χρήστη μηδενίζεται πριν τη μεταφορά χρημάτων σε αυτόν μέσω της συνάρτησης transfer αντί για send, για την προστασία ενάντια σε επιθέσεις επανεισαγωγής, όπως εξετάσαμε στο τρίτο κεφάλαιο.

```
1     function destroy() isAdmin public{
2       //cleanup function to be used ONLY IN TESTING
3       selfdestruct(owner);
4     }
5 }
```

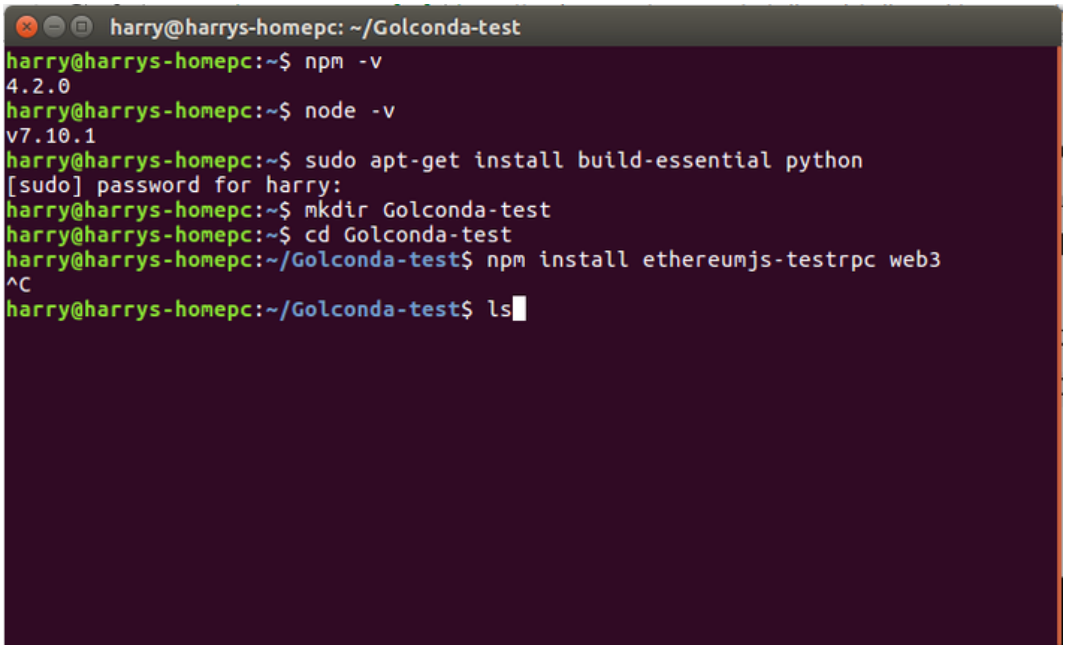
Τέλος, εισάγεται μια συνάρτηση διαγραφής του συμβολαίου,η οποία μπορεί να κληθεί μόνο από τον ιδιοκτήτη, ώστε σε περίπτωση υποβολλής καινούριας έκδοσης του συμβολαίου, η παλαιότερη να διαγράφεται ώστε να μην επιβαρρύνεται η blockchain με περιττό όγκο. Για την επιβράβευση τέτοιας συνειδητής συμπεριφοράς, η EVM επιβραβεύει τις συναρτήσεις διαγραφής συμβολαίων επιστρέφοντας το κόστος υποβολλής σε gas.

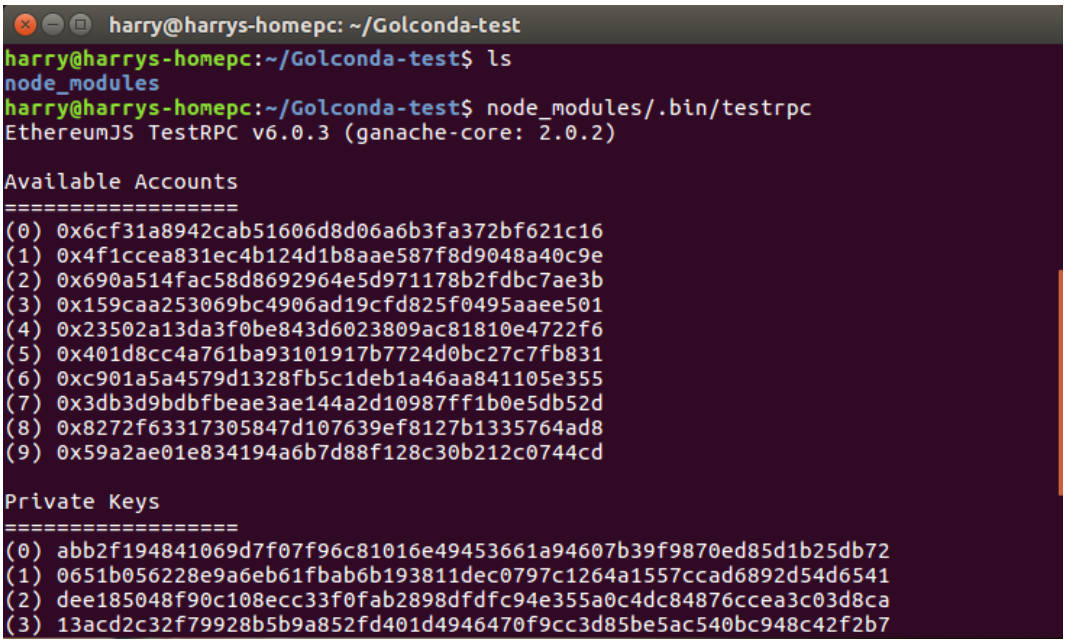
## 6.2 Έλεγχος Λειτουργίας

### 6.2.1 Έλεγχος σε τοπικό δίκτυο

Ο έλεγχος λειτουργίας του smart contract θα γίνει αρχικά σε μία τοπική προσομοίωση ενός blockchain δικτύου, ώστε να έχουμε πλήρη έλεγχο στην έκδοση νέων blocks καθώς και μια αρκετά μεγάλη ποσότητα ether για τις δοκιμές μας. Τα πακέτα που θα χρησιμοποιηθούν είναι το node.js το οποίο μέσω της βιβλιοθήκης web3.js θα μας επιτρέψει να αλληλεπιδράσουμε με την ιδιωτική μας blockchain,

το πακέτο testrpc, το οποίο θα προσωμοιώσει την τοπική blockchain και φυσικά τον compiler της γλώσσας solidity. Όλες οι εντολές θα εισαχθούν σε command line των Ubuntu 14.04, κυρίως λόγω της απλούστευσης της εγκατάστασης των προγραμμάτων λογισμικού σε αντίθεση με τα Windows [?]. Για αρχή, αφού ελέγξουμε ότι έχουμε τις κατάλληλες εκδόσεις των npm και node, θα εγκαταστήσουμε το testrpc στον φάκελο της εφαρμογής μας και στη συνέχεια θα το τρέξουμε. Οι εντολές είναι όπως φαίνεται παρακάτω :






Παρατηρούμε ότι καθώς τρέχουμε την εντολή testrpc, κάνουμε deploy ένα τοπικό δίκτυο με 10 διευθύνσεις για τις οποίες τα ιδιωτικά κλειδιά δίνονται από κάτω και είναι διαθέσιμα σε εμάς με 100 (ψεύτικα) ether η κάθε μια. Θα αφήσουμε το παράθυρο αυτό ανοιχτό ώστε να είναι το δίκτυο μας διαθέσιμο και θα ανοίξουμε ένα άλλο terminal ώστε να ανεβάσουμε την εφαρμογή μας στο δίκτυο. Αρχικά θα μεταφέρουμε το αρχείο του κώδικα μας, CrowdfundedCharity.sol, σε αυτόν τον φάκελο και στην συνέχεια θα τρέχουμε την εντολή node ώστε να ενεργοποιήσουμε την κονσόλα node και να αρχικοποιήσουμε τα αντικείμενα solc, που είναι ο compiler της solidity που θα χρησιμοποιηθεί και web3, που είναι η σύνδεση με το testrpc δίκτυο που τρέχει στο άλλο μας παράθυρο.

```
1 harry@harrys-homepc:~/Golconda-test$ node
2 > Web3 = require('web3')
3 > web3 = new Web3(new Web3.providers.HttpProvider("http://
    localhost:8545"));
```

Ελέγχουμε ότι η αρχικοποίηση έγινε σωστά μέσω της εντολής :

```
1 > web3.eth.accounts
```

```
> web3.eth.accounts
[ '0x6cf31a8942cab51606d8d06a6b3fa372bf621c16',
  '0x4f1ccea831ec4b124d1b8aae587f8d9048a40c9e',
  '0x690a514fac58d8692964e5d971178b2fdbc7ae3b',
  '0x159caa253069bc4906ad19cfd825f0495aaee501',
  '0x23502a13da3f0be843d6023809ac81810e4722f6',
  '0x401d8cc4a761ba93101917b7724d0bc27c7fb831',
  '0xc901a5a4579d1328fb5c1deb1a46aa841105e355',
  '0x3db3d9bdbfbeae3ae144a2d10987ff1b0e5db52d',
  '0x8272f63317305847d107639ef8127b1335764ad8',
  '0x59a2ae01e834194a6b7d88f128c30b212c0744cd' ]
```

Παρατηρούμε ότι έχουμε πρόσβαση στους λογαριασμούς που δημιουργήθηκαν μέσω του testrpc, άρα η αρχικοποίηση του αντικειμένου web3 έγινε σωστά. Στην συνέχεια, θα φορτώσουμε τον κώδικα μας σε μια μεταβλητή string και θα κάνουμε το compile.

```
1 > code = fs.readFileSync('CrowdfundedCharity.sol').toString()
2 > solc = require('solc')
3 > compiledCode = solc.compile(code)
```

Μετά το compile, υπάρχουν δύο σημεία του μεταφρασμένου κώδικα στα οποία αξίζει να δώσουμε σημασία.

compiledCode.contracts[':CrowdfundedCharity'].bytecode : Αυτός είναι ο συμβολομεταφρασμένο κώδικας σε μορφή bytecode, ο οποίος είναι αυτός που θα ανέβει στη blockchain compiledCode.contracts[':CrowdfundedCharity'].interface: Αυτή είναι μια φόρμα διεπαφών ( interface template/ application binary interface , "abi" ) η οποία λέει στον χρήστη ποιές συναρτήσεις και διεπαφές του smart contract είναι διαθέσιμες. Κάθε φορά που θα θέλουμε να αλληλεπιδράσουμε με το smart contract μας θα χρειαζόμαστε αυτόν τον ορισμό abi.

Στην συνέχεια, θα ανεβάσουμε το αρχείο. Πρώτα πρέπει να δημιουργήσουμε ένα αντικείμενο contract, το οποίο θα ονομάσουμε CharityContract και θα χρησιμοποιήσουμε για να εκδώσουμε το smart contract μας.

```
1 > abiDefinition = JSON.parse(compiledCode.contracts[':
    CrowdfundedCharity'].interface)
2 > CharityContract = web3.eth.contract(abiDefinition)
3 > byteCode = compiledCode.contracts[':CrowdfundedCharity'].
    bytecode
4 > deployedContract = CharityContract.new(['version'],{data:
    byteCode, from: web3.eth.accounts[0], gas: 4700000})
5 > deployedContract.address
6 > contractInstance = CharityContract.at(deployedContract.address)
```

Κατα την εντολή VotingContract.new, εισάγουμε τα δεδομένα της αρχικοποίησης, τα οποία στην αρχική μας έκδοση του κώδικα είναι μόνο ο αριθμός έκδοσης και

δίνεται μέσω της μεταβλητής version, τον κώδικα σε γλώσσα μηχανής, bytecode και τέλος ορίζουμε από ποιά διεύθυνση μας θα ανεβάσουμε το smart contract και το διαθέσιμο gas για την χρηματοδότηση της διαδικασίας. Ως ένας ακόμα λογαριασμός ethereum, το smart contract μας αποκτά μια διεύθυνση, την οποία φορτώνουμε στην μεταβλητή contract instance ώστε μέσω αυτής να μπορούμε να αλληλεπιδράσουμε με το smart contract μας. Καθώς εκτελούμε αυτές τις διαδικασίες, στο παράθυρο του testrpc μπορούμε να δούμε τη δημιουργία νέων blocks με κάθε εντολή που αποστέλουμε. Πλέον μπορούμε να ξεκινήσουμε το testing των συναρτήσεων μας μέχρι να είμαστε σίγουροι ότι λειτουργούν σωστά. Για παράδειγμα, η κλήση της συνάρτησης newCampaign θα υλοποιηθεί ώς εξής :

```
1 >contractInstance.newCampaign( web3.eth.accounts[4], 10, 15, {
    from: web3.eth.accounts[1], gas:1500000})
```

και για να επιβεβαιώσουμε ότι δούλεψε, θα κάνουμε κλήση της :

```
1 >contractInstance.numCampaigns()
```

, η οποία είναι μια getter function που δημιουργήθηκε αυτόματα από τον συμβολομεταφραστή της solidity, καθώς ορίσαμε την μεταβλητή numCampaigns ώς public. Η κλήση αυτή επιστρέφει :

[String: '1'] s: 1, e: 0, c: [ 1 ] όπου η τιμή της string επιβεβαιώνει ότι έχουμε μια ενεργή εκστρατεία.

Η κλήση της συνάρτησης contribute, η οποία είναι payable, δέχεται δηλαδή ένα χρηματικό ποσό και το μεταβιβάζει στο smart contract , θα είναι ως εξής :

```
1 >deployedContract.contribute(1,{from:web3.eth.accounts[2],value:
    web3.toWei('9','ether'),gas:150000})
```

η τιμή της value κάνει κλήση της συνάρτησης toWei, η οποία εκφράσει ένα ποσό σε Wei, που είναι η μικρότερη υποδιαίρεση του ether.

Στην συνέχεια, μπορούμε να δούμε το συνολικό ποσό για το πρώτο μας campaign μέσω της totalFundsFor

```
1 >contractInstance.totalFundsFor(1, { from: web3.eth.accounts[1],
    gas:1500000})
```

η οποία επιστρέφει :

[String: '9000000000000000000'] s: 1, e: 18, c: [ 90000 ] , το οποίο είναι 9 ether τα οποία δώσαμε προηγουμένως.

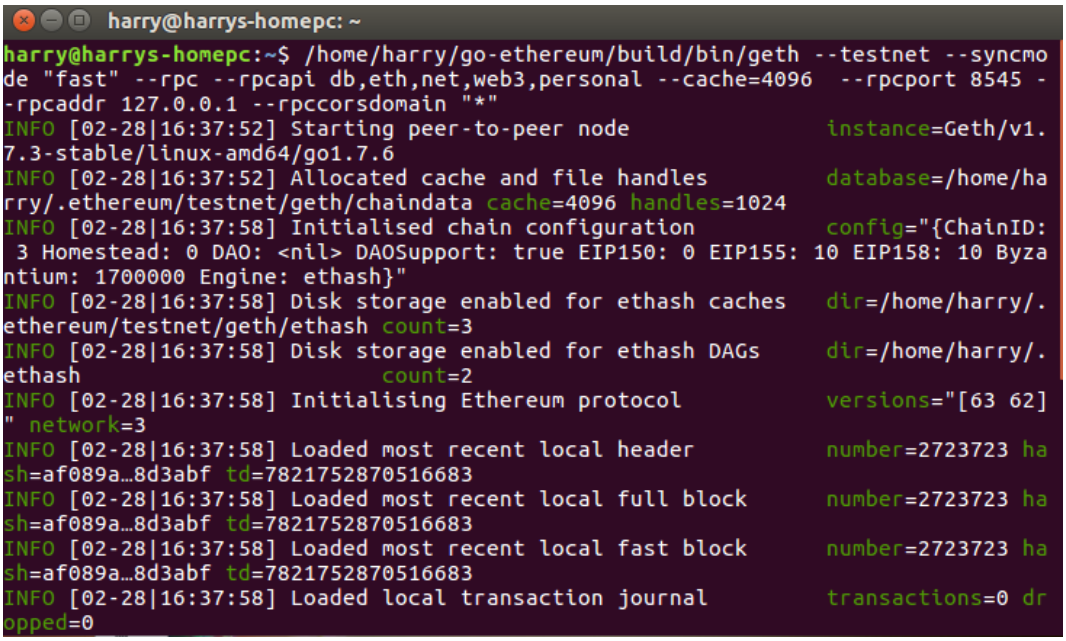
Τέλος, με την κλήση της checkGoalReached, αν τα χρήματα έχουν συγκεντρωθεί και το χρονικό περιθώριο έχει λήξει, το ποσό θα μεταφερθεί στην διεύθυνση που ορίστηκε κατα τη κλήση της newCampaign.

### 6.2.2 Έλεγχος στο δημόσιο ελεγκτικό δίκτυο Ropsten

Ο έλεγχος σε τοπικό δίκτυο είναι χρήσιμος για τον βασικό έλεγχο της λειτουργικότητας του smart contract μας, ωστόσο, νέα blocks δημιουργούνται μόνο όταν εκτελούνται εντολές από μέρους μας. Είναι χρήσιμο να γίνεται έλεγχος της συμπεριφοράς και της απόδοσης της εφαρμογής μας και σε ένα δημόσιο δίκτυο πριν την εισαγωγή της στην κύρια blockchain του Ethereum. Για αυτό υπάρχουν και χρησιμοποιούνται διάφορα δημόσια δίκτυα ελέγχου όπως τα Ropsten, Kovan, Rinkeby , τα οποία λειτουργούν ακριβώς όπως το Ethereum , στα οποία όμως το mining γίνεται από συγκεκριμένες μονάδες που υποστηρίζονται από την προγραμματιστική κοινότητα και παρέχουν δωρεάν ether για έλεγχο. Συνεπώς, μπορεί τα έξοδα gas να είναι ακριβώς τα ίδια με την κύρια αλυσίδα, ωστόσο τα ether με τα οποία εξοφλούνται δεν έχουν αξία και άρα το κόστος ανάπτυξης και ελέγχου είναι μηδενικό.

Μαζί με τη σουιτα ethereum που κατεβάσαμε όταν εγκαταστήσαμε τη solidity, εγκαταστάθηκε και το πρόγραμμα geth, το οποίο συγχρονίζει τον υπολογιστή μας με το κυρίως δίκτυο του ethereum, ενώ με τα κατάλληλα ορίσματα τον συγχρονίζει στο testnet. Μέσω της παρακάτω εντολής λοιπόν, το geth θα αρχίσει να κατεβάζει τα δεδομένα της test blockchain του Ropsten test network, διαμορφώνοντας τον υπολογιστή μας σε έναν ελαφρύ κόμβο με τη δυνατότητα να εκπέμπει συναλλαγές στο δίκτυο [?].

```
1 /home/harry/go-ethereum/build/bin/geth --testnet --syncmode "fast"
     --rpc --rpcapi db,eth,net,web3,personal --cache=4096  --
    rpcport 8545 --rpcaddr 127.0.0.1 --rpccorsdomain "*"
```



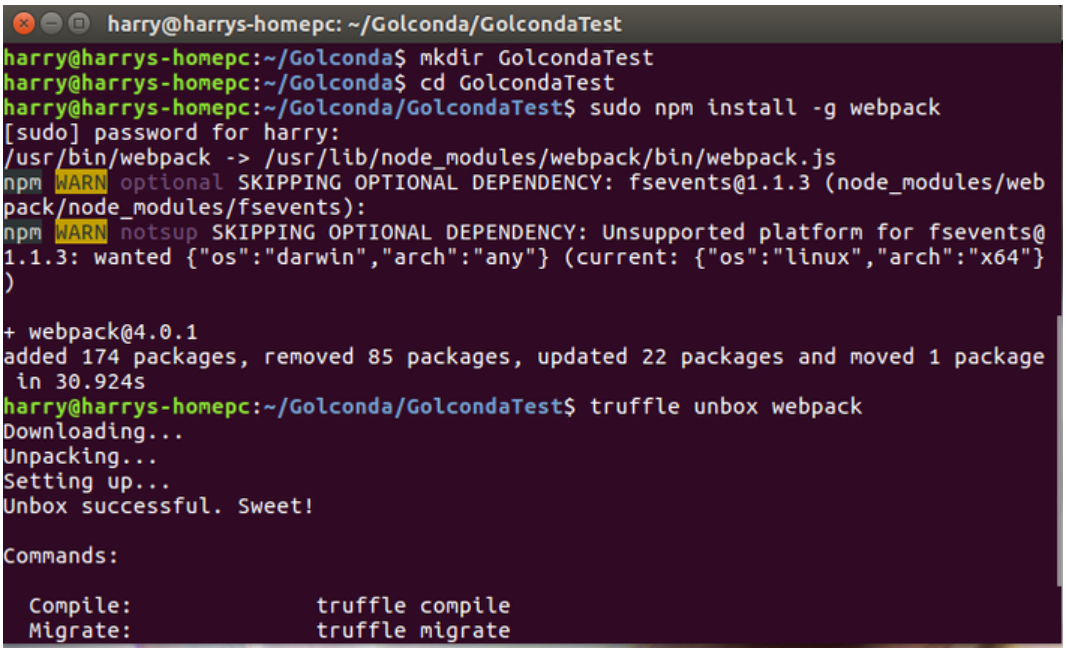

Η διαδικασία αυτή διαρκεί μερικές ώρες. Όταν ολοκληρωθεί, θα βλέπουμε blocks να προστίθενται ανα μισό δευτερόλεπτο στο παράθυρο εντολών που τρέχει το geth :

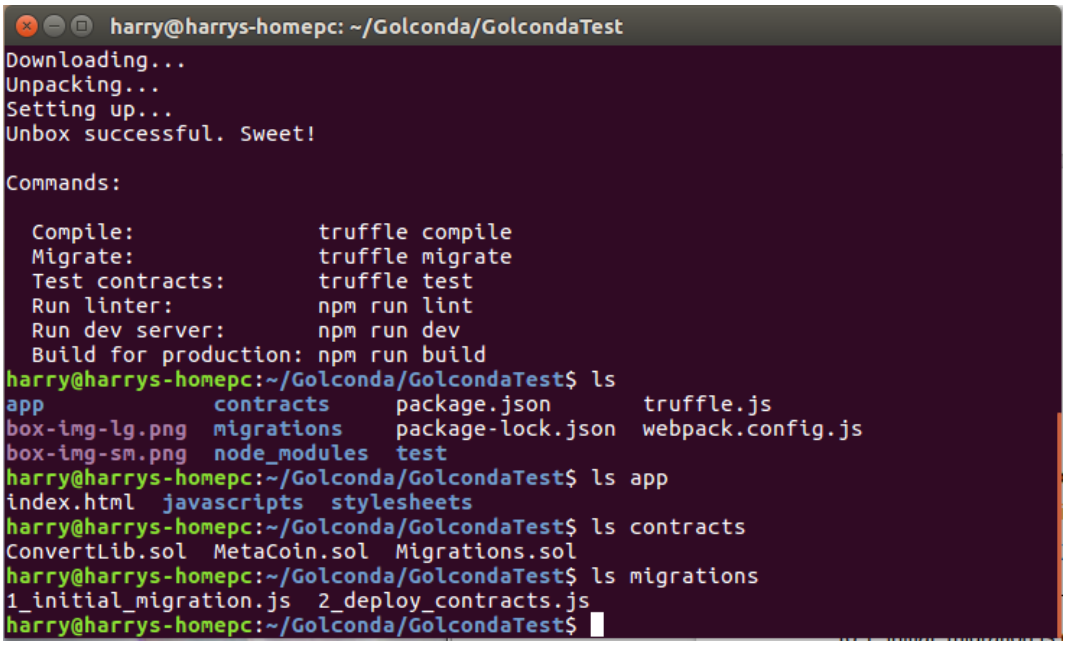
!!!

Στη συνέχεια διατηρούμε το παράθυρο εντολών που τρέχει το geth ανοιχτό και σε ένα άλλο μπορούμε να εφάρμοσουμε τις ίδιες εντολές όπως προηγουμένως. Ωστόσο, για εμβάνθυνση στις τεχνικές ανάπτυξης στο περιβάλλον ethereum, θα εξεταστεί και ένας διαφορετικός τρόπος υποβολής και ελέγχου smart contracts, ο οποίος είναι μέσω του Truffle framework. Όπως αναφέραμε στο Κεφάλαιο 2, το Truffle είναι ένα λογισμικό το οποίο διευκολύνει την δημιουργία και τον έλεγχο smart contracts αυτοματοποιώντας τις διαδικασίες compile και deployment. Η εγκατάσταση του Truffle είναι απλή. Αρχικά εκτελούμε την εντολή :

```
1 npm install -g truffle
```

Μολις ολοκληρωθεί, ορίζουμε έναν φάκελο στον οποίο θα στήσουμε το περιβάλλον του truffle για το συγκεκριμένο project.




Το truffle θα εισάγει τα απαραίτητα αρχεία για τη λειτουργία του project, καθώς και κάποια περιττά αρχεία των tutorials του, δηλαδή τα Convert.lib και Metacoin.sol, τα οποία μπορούμε να διαγράψουμε. Τα αρχεία που έχουν μεγάλη σημασία είναι τα Migrations.sol, `1_initial_migration.js`, `2_deploy_contracts.js` και το truffle.js. Το `1_initial_migration.js` όταν εκτελείται πρώτη φορά, υποβάλει το Migrations.sol, το οποίο είναι ένα smart contract που διατηρεί μια λίστα με εκδόσεις του κύριου smart contract, ώστε να είναι εμφανές

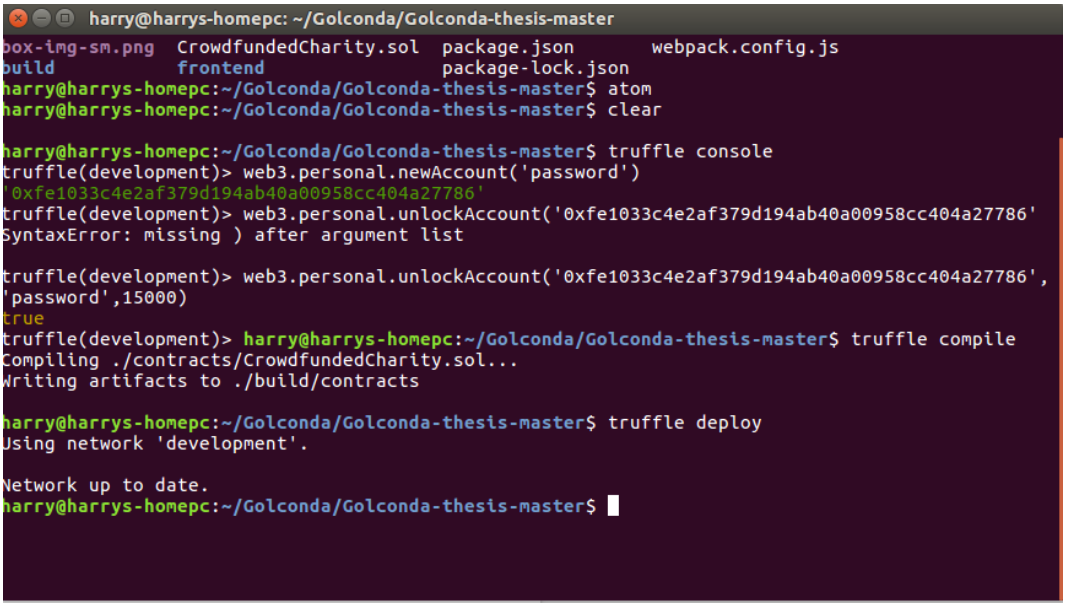
ποια είναι η πιο πρόσφατη έκδοση του. Στην συνέχεια, το `2_deploy_contracts.js` υποβάλλει το κύριο smart contract και αποθηκεύει για εμάς τα στοιχεία interface, bytecode και τη διεύθυνση του αποθηκευμένου συμβολαίου. Χρειάζεται όμως πρώτα μια μετατροπή για την αφαίρεση των δεδομένων των tutorials και την προσθήκη του ονόματους του smart contract μας. Θέτουμε λοιπόν την μορφή του ως εξής :

```
var CrowdfundedCharity = artifacts.require("./CrowdfundedCharity.
    sol");

module.exports = function(deployer) {
  deployer.deploy(CrowdfundedCharity,['1.0.4']);
};
```

Στην συνέχεια, μετακινούμε το smart contract μας, CrowdfundedChariy.sol στον φάκελο contracts. Πρέπει επίσης να δημιουργήσουμε τον λογαριασμό με τον οποίο θα αλληλεπιδρούμε με την test blockchain. Όπως και στην τοπική υλοποίηση, για την χρήση του truffle πρέπει να έχουμε σε μια κονσόλα το geth να τρέχει και να συγχρονίζει τον υπολογιστή μας με την blockchain με τον ίδιο τρόπο που αναφέρθηκε προηγουμένως. Εφ΄όσον συμβαίνει αυτό, μπορούμε να δημιουργήσουμε έναν λογαριασμό για την αποθήκευση των ether και την υποβολή συναλλαγών. Εκτελούμε τις παρακάτω εντολές :

```
truffle console
web3.personal.newAccount‘(’password)
web3.personal.unlockAccount( web3.eth.accounts’[0],’password
    ,15000)
```



όπου το τρίτο πεδίο ορίζει την διάρκεια σε δευτερόλεπτα για την οποία ο λογαριασμός παραμένει ξεκλείδωτος. Τώρα μένει να αποκτήσουμε μερικά ether για την διενέργεια ελέγχων. Η πιο εύκολη μέθοδος είναι η απευθείας λήψη τους από μια δημόσια πηγή (faucet). Για την προβολή της σελίδας είναι απαραίτητη η λήψη της προσθήκης MetaMask, η οποία θα είναι απαραίτητη και για την αλληλεπίδραση με το περιβάλλον διεπαφής, οπότε συνιστάται η εγκατάσταση της σε αυτό το σημείο.

Μέσω του MetaMask μπορούμε να λάβουμε test-ether από μια δημόσια πηγή ( faucet ) και να τα στείλουμε στην διεύθυνση που μόλις δημιουργήσαμε. Το truffle.js αποθηκεύει τις απαραίτητες πληροφορίες για τη σύνδεση στο κάθε δίκτυο και μερικές αρχές για την υποβολή των smart contracts σε αυτό όπως η προκαθορισμένη τιμή για τα gas limit, gas price και η διεύθυνση του λογαριασμού που θα χρησιμοποιεί η εφαρμογή για την υποβολή των συναλλαγών. Για την διενέργεια του testing, στην περίπτωση μας θα θέσουμε τη μορφή του ως εξής :

```
require('babel-register')

module.exports = {
  networks: {
    development: {
      host: '127.0.0.1',
      port: 8545,
      from: '0xfea62d5040c90d8733405952624fbbbdcd34fb27',
      network_id: '*', // Match any network idz
      gas: 2500000
    }
  }
}
```

όπου η τιμή from περιέχει την διεύθυνση του λογαριασμού που δημιουργήσαμε

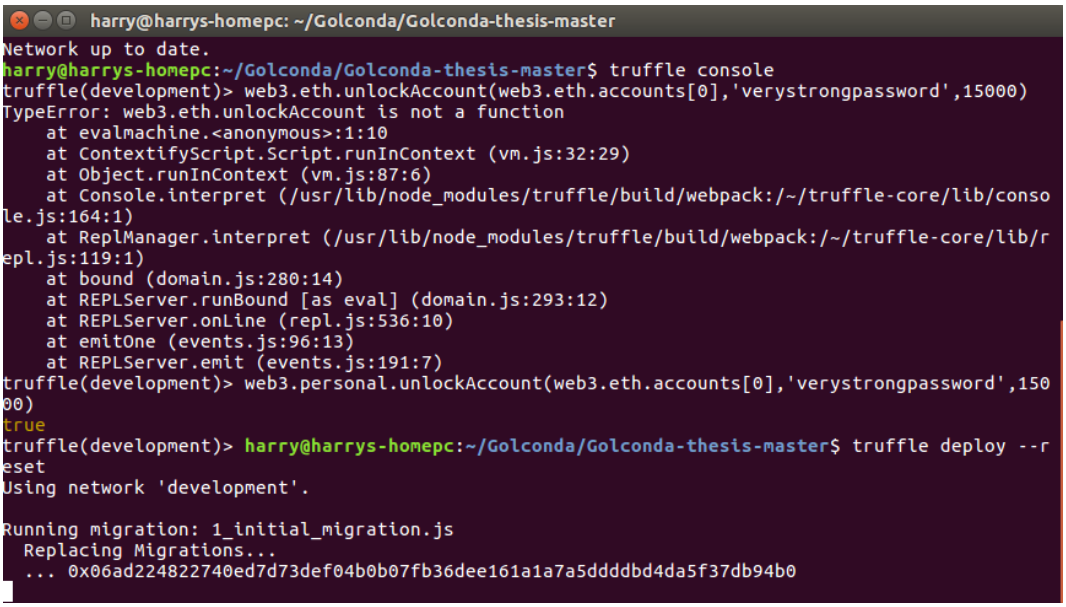
Πλέον το setup έχει ολοκληρωθεί και μπορούμε να υποβάλλουμε το smart contract μας με τις εξής εντολές( εκτός της truffle console ) :

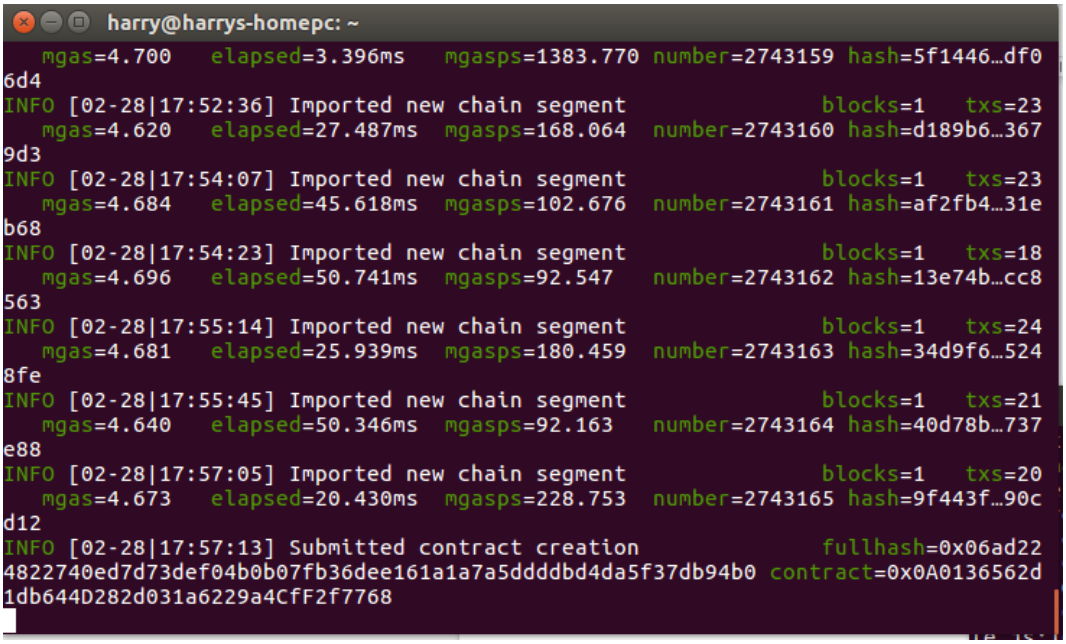
```
truffle compile
truffle deploy
```

Σε περίπτωση που δεν είναι η πρώτη φορά που υποβάλλουμε το smart contract στο δίκτυο χρησιμοποιούμε την εντολή

```
truffle migrate --reset
```

Θα δούμε από το terminal αυτό την επιβεβαίωση της υποβολής των smart contracts ενώ από την πλευρά του geth θα δούμε την υποβολή των συναλλαγών στο δίκτυο.

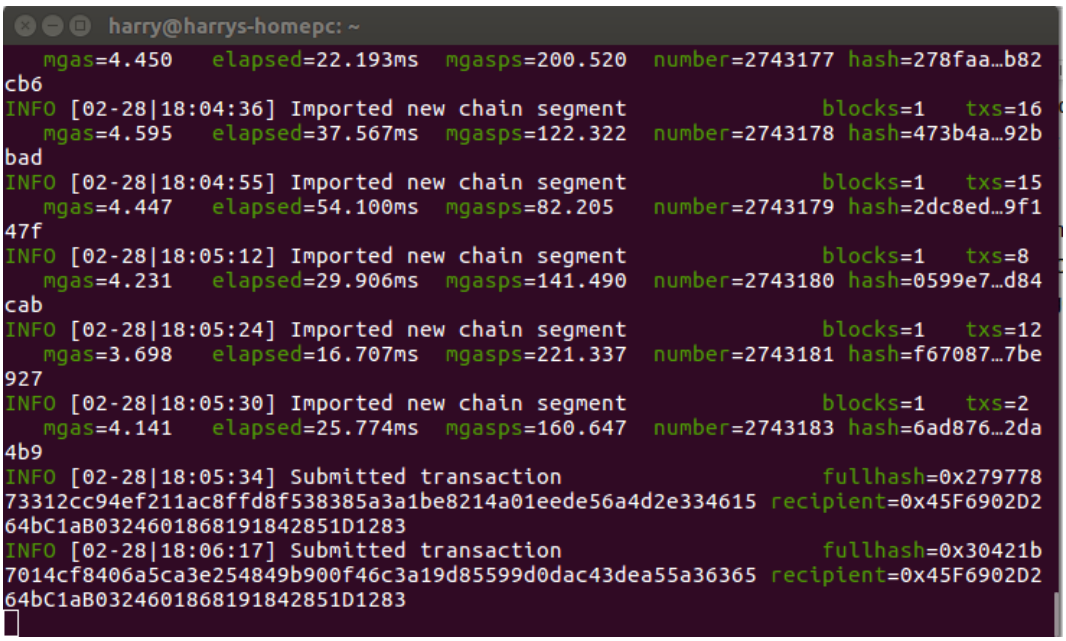



Όταν ολοκληρωθεί, τα δεδομένα της υποβολής θα αποθηκευτούν στην μεταβλητή CrowdfundedCharity.deployed() και οι εντολές για την αλληλεπίδραση με το smart contract εκτελούνται με την εξής μέθοδο εντός του truffle console :

```
1 CrowdfundedCharity.deployed().then(function(contractInstance){
    contractInstance.newCampaign('0
    x88dDe7754Ebb8000Afa52E2CBc971fD30ac424d7',web3.toWei('0.1','
    ether'),'1','testCharity','a test charity').then(function(v){
    console.log(v)})})
```


Και μπορούμε να επιβεβαιώσουμε την επιτυχή αποστολή της συναλλαγής μας μέσω του geth :



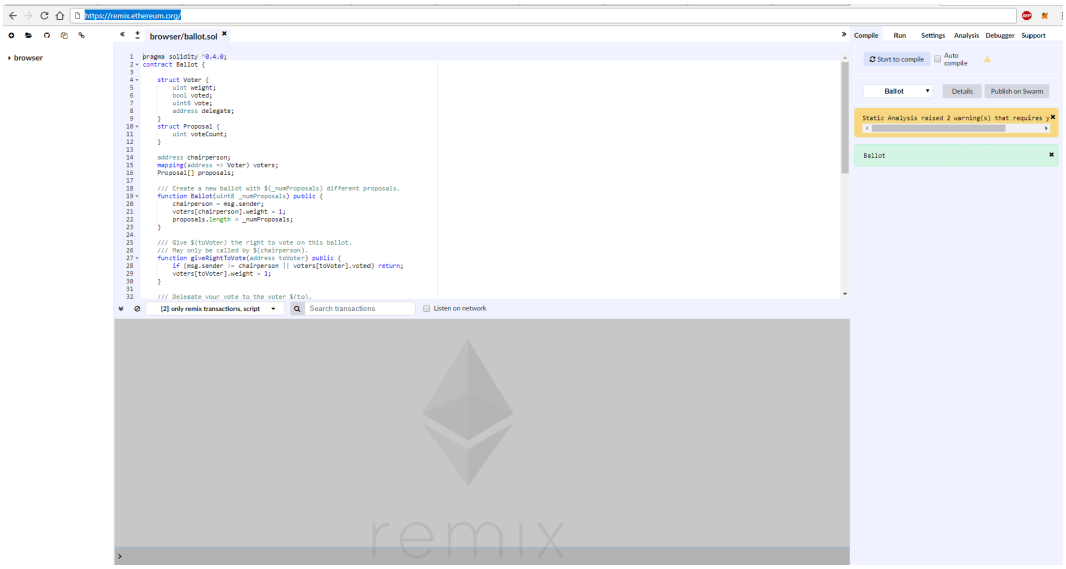
Αντίστοιχα, οι εντολές contribute και checkGoalReached υποβάλλονται με τις εξής κλήσεις :

```
1 CrowdfundedCharity.deployed().then(function(contractInstance){
    contractInstance.contribute(1,{from:web3.eth.accounts[0],value
    :web3.toWei('1','ether')}).then(function(v){console.log(v)})})
2
3 CrowdfundedCharity.deployed().then(function(contractInstance){
    contractInstance.checkGoalReached(0).then(function(v){console.
    log(v)})})
```

Τέλος, για πληρότητα, αναφέρεται και μια τρίτη μέθοδος ανάπτυξης και υποβολής smart contracts, ο οποίος υλοποιείται αποκλειστικά μέσω internet browser και δεν απαιτεί ταυτόχρονη λειτουργία του geth.

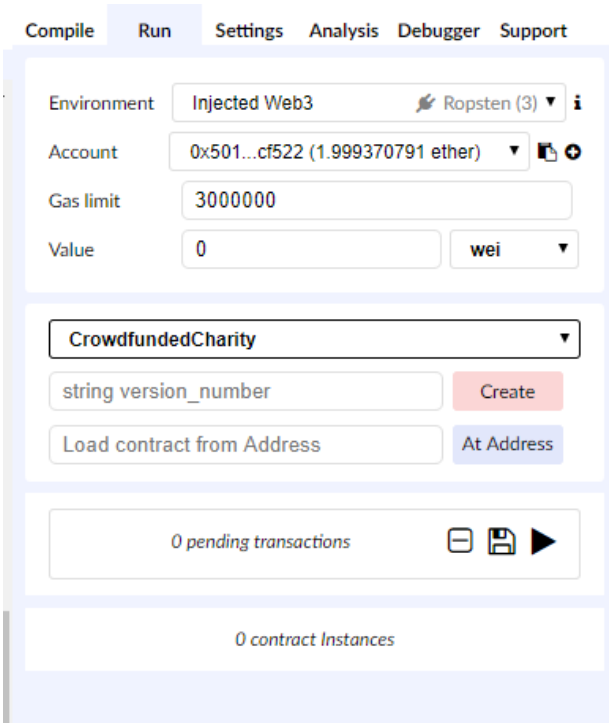
### 6.2.3 Έλεγχος μέσω REMIX IDE

Χρησιμοποιείται η ιστοσελίδα REMIX [?] https://remix.ethereum.org/ η οποία μας προσφέρει ένα γραφικό περιβάλλον διεπαφής και πολλές απαραίτητες λειτουργίες.




Διαγράφουμε τον κώδικα που βρίσκεται στο κεντρικό πλαίσιο κειμένου και επικολλούμε τον κώδικα του δικού μας smart contract. Στην συνέχεια επιλέγουμε compile. Εφ'όσον δεν υπάρχουν προβλήματα, μπορούμε να επιλέξουμε την καρτέλα run. Για την υποβολή των συναλλαγών δημιουργίας του smart contract μας είναι απαραίτητο να κατέχουμε τη προσθήκη MetaMask στον browser μας. Η εγκατάσταση και ρύθμιση του αναλύεται στο τέλος του κεφαλαίου, στις οδηγίες χρήσης της εφαρμογής.

Από την καρτέλα run, εφ'όσον έχουμε εγκαταστήσει το MetaMask σωστά, στο πεδίο environment θα είναι επιλεγμένη η τιμή "Injected Web 3 - Ropsten (3) " και στη διεύθυνση θα εμφανίζεται η διεύθυνση του λογαριασμού MetaMask μας.

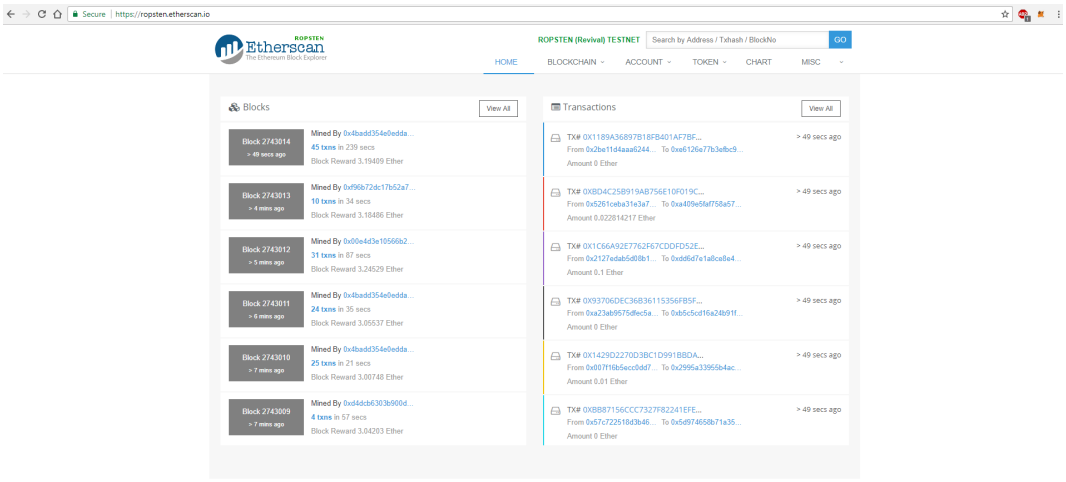



Επίσης θα εμφανίζονται τα κατάλληλα πεδία ορισμάτων για τον constructor της εφαρμογής μας. Μέσω της επιλογής Create, ο λογαριασμός του MetaMask θα χρησιμοποιηθεί για την αποστολή της συναλλαγής δημιουργίας του smart contract, και στην κονσόλα αποτελεσμάτων θα εμφανιστεί ο λογαριασμός του συμβολαίου που δημιουργήθηκε. Παρατηρούμε ότι η συγκεκριμένη μέθοδος ανάπτυξης, σε συνδυασμό με την μέθοδο επαλήθευσης που αναλύεται στην επόμενη ενότητα, είναι αρκετά πιο φιλική ως προς τον χρήστη καθώς δεν απαιτεί εγκατάσταση προγραμμάτων, κατέβασμα της blockchain ή εξειδικευμένου λογισμικού.

### 6.2.4 Χρήση της σελίδας Etherscan για την επαλήθευση αποτελεσμάτων

Χρήση της σελίδας Etherscan [?] για την επαλήθευση αποτελεσμάτων Για την επαλήθευση των αποτελεσμάτων, σε οποιαδήποτε από τις τρείς μεθόδους ανάπτυξης, ένα σημαντικό εργαλείο είναι η σελίδα Etherscan, η οποία διαθέτει μεθόδους εξερεύνησης της Ethereum Blockchain και μπορεί να εμφανίσει κάθε δημόσια διαθέσιμο δεδομένο σχετικά με αυτή στο χρήστη. Εισάγωντας στο κελί αναζήτησης τη διεύθυνση ενός λογαριασμού, μπορούμε να λάβουμε πληροφορίες

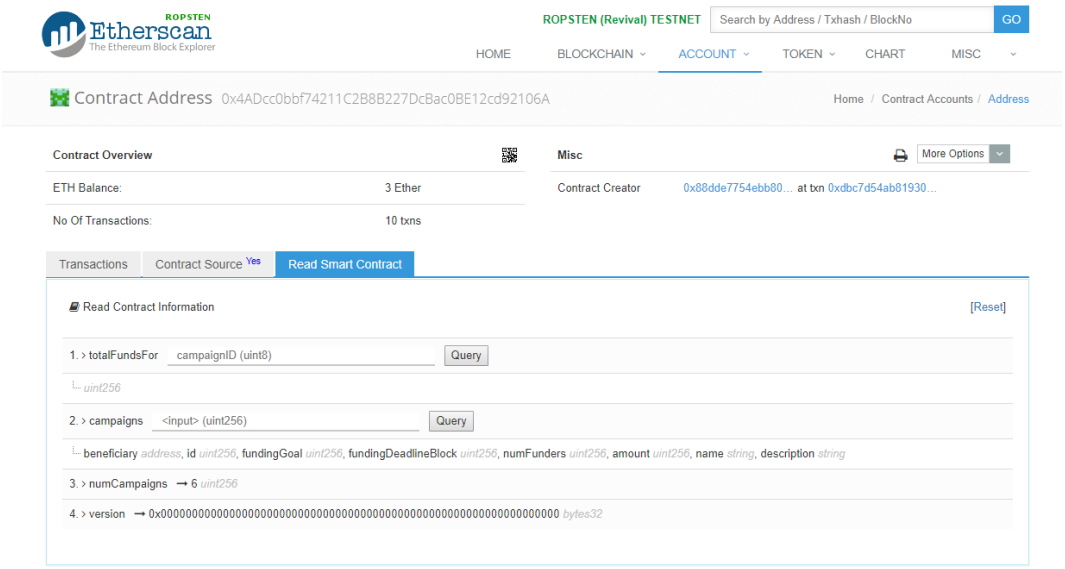
σχετικά με το υπολοιπόμενο ποσό του και το ιστορικό συναλλαγών του, ενώ αν εισάγουμε τη διεύθυνση ενός smart contract μπορούμε να δούμε τη δομή του καθώς και τα ποσά ether που διαθέτουν. Επίσης διαθέτει χρήσιμες πληροφορίες όπως ο αριθμός του τελευταίου block καθώς και διάφορες μετρικές της κατάστασης του δικτύου.




---

Εισάγουμε τη διεύθυνση του smart contract μας η οποία επιστράφηκε από την σελίδα remix. Καθώς το smart contract αποθηκεύεται στην blockchain ως bytecode, η δομή του δεν είναι άμεσα διαθέσιμη. Στην καρτέλα source code όμως, μας επιτρέπεται να ανεβάσουμε τον κώδικα του smart contract, το όνομα, τα ορίσματα και την έκδοση του compiler. Αν το bytecode που προκύπτει από τις εισόδους μας επαληθεύει το bytecode της συναλλαγής, το σύστημα εισάγει τον κώδικα αυτόν στη βάση δεδομένων του και πλέον ο κώδικας της εφαρμογής μας είναι δημόσιος.

Επίσης μας δίνεται η δυνατότητα απευθείας εξερεύνησης των μεταβλητών του smart contract μέσω της καρτέλας Read Smart Contract




## 6.3 Περιβάλλον Διεπαφής

Για τη σχεδίαση του περιβάλλοντος διεπαφής, επιλέχθηκε να γίνει σχεδίαση μέσω του framework angular.js [?] για λόγους ευκολίας και απόδοσης. Χρησιμοποιήθηκε ως βάση ένα έτοιμο template, το οποίο ονομάζεται Flatlogic [?] που διατίθεται δωρεάν στην ιστοσελίδα της Angular. Για την αρκούσα παρουσίαση όλων των σχετικών αποτελεσμάτων και λειτουργιών, αποφασίστηκε ότι αρκούν τρεις υπό-σελίδες / καρτέλες. Η αρχική σελίδα, που υποδέχεται τους χρήστες στην εφαρμογή και εμφανίζει πληροφορίες σχετικά με τη λειτουργία της. Η σελίδα view, που εμφανίζει τις διαθέσιμες εκστρατείες και επιτρέπει τη δωρεά σε κάποια από αυτές Η σελίδα create, που επιτρέπει τη δημιουργία νέων εκστρατειών Στόχος της σχεδίασης ήταν η απλοποίηση της περιήγησης και αλληλεπίδρασης του χρήστη με την εφαρμογή. Σε κάθε σελίδα είναι ξεκάθαρο ποιές είναι οι ενέργειες που μπορεί να λάβει ό χρήστης. Επίσης είναι δυνατή η μετάβαση σε κάθε σελίδα από οποιαδήποτε άλλη μέσω dashboard. Αφού αφαιρέθηκαν οι αχρείαστες λειτουργίες, το template εμπλουτίστηκε με τις απαραίτητες συναρτήσεις που επιτρέπουν την επικοινωνία με την ethereum blockchain οι οποίες και θα αναλυθούν :

### 6.3.1 Φόρτωση βιβλιοθήκης web3.js στον περιηγητή του χρήστη μέσω του προσθέτου MetaMask

Όπως αναφέραμε κατά τη διαδικασία της ανάπτυξης του smart contract, είναι απαραίτητη η χρήση κάποιου προγράμματος που να συγχρονίζει τον υπολογιστή μας με την Ethereum Blockchain και να επιτρέπει την αποστολή και λήψη δεδομένων σε αυτή. Για τους υπολογιστές των χρηστών, αυτό θα γίνεται μέσω της προσθήκης του Metamask, το οποίο μετατρέπει τον υπολογιστή σε έναν ελαφρύ κόμβο, χωρίς να υπάρχει ανάγκη κατεβάσματος όλων των δεδομένων της Blockchain. Ο ορισμός του MetaMask στον κώδικα του front-end γίνεται μέσω ενός έτοιμου snippet που διατίθεται από την ιστοσελίδα MetaMask και επικολλείται στο αρχείο app.js στη διεύθυνση frontent/app/app.js. Στο ίδιο αρχείο επικολλείται και η διεύθυνση του smart contract στην Ethereum Blockchain και το ABI ώστε να επιτρέπεται η πλήρης επικοινωνία με το smart contract μέσω της βιβλιοθήκης web3.js .

```
app.run(['$rootScope', '$state', '$cookieStore', 'appData',
   function($rootScope, $state, $cookieStore, appData) {
       if (typeof web3 !== 'undefined') {
      console.warn("Using web3 detected from external source like
         Metamask");
      // Use Mist/MetaMask's provider
      web3 = new Web3(web3.currentProvider);
      web3.version.getNetwork((err, netId) => {
          switch (netId) {
            case "1":
              console.log('This is mainnet');
              break
            case "2":
              console.log('This is the deprecated Morden test
                 network.');
              break
            case "3":
              console.log('This is the ropsten test network.');
              break
            default:
              console.log('This is an unknown network.');
          }
        })
```

Ο παραπάνω κώδικας δημιουργεί ένα αντικείμενο web3 το οποίο παρέχεται μέσω του MetaMask. Σε περίπτωση που δεν είναι εγκατεστημένος, η συνάρτηση αυτή θα αποτύχει και θα εμφανιστεί προειδοποιητικό μήνυμα στο χρήστη.

```
      var inter=
      '[{"constant":true, ... "event"}]';

      abi = JSON.parse(inter);
```

```
5       var CharityContract = web3.eth.contract(abi);
6       var contractInstance = CharityContract.at('0
            x4ADcc0bbf74211C2B8B227DcBac0BE12cd92106A');
7
8       console.log(contractInstance);
9
10      $rootScope.contract = contractInstance;
11
12      $rootScope.args = {from: web3.eth.accounts[0], gas:450000};
13    }
14
15
16 }]);
```

Οι παραπάνω εντολές δημιουργούν ένα αντικείμενο contract το οποίο περιέχει τη διεύθυνση και τον τρόπο επικοινωνίας με το smart contract μας και χρησιμοποιείται από τις υπόλοιπες συναρτήσεις αλληλεπίδρασης μέσω της μεταβλητής `$rootScope.contract` . Στην συνέχεια θα εξετάσουμε την λειτουργία των controllers των σελίδων view και create. Η διαδικασία σχεδιασμού της φόρμας υποβολής και της προσθήκης buttons ακολουθεί τις κλασσικές πρακτικές της angular.js και ορίζεται στυλιστικά από το template και δεν θα αναλυθεί.

### 6.3.2 View Controller

Ο κώδικας που καθορίζει την εύρεση και την εμφάνιση των διαθέσιμων εκστρατειών στην σελίδα view βρίσκεται στο αρχείο frontend/app/controllers/campaignsController.js και είναι ο εξής :

```
1  'use strict';
2
3  angular.module('webUI')
4      .controller('campaignsController', ['$scope', '$state', '
           $rootScope',
5          function($scope, $state, $rootScope) {
6
7              $scope.campaigns = [];
8              var that = this;
9
10
11             that.$onInit = function() {
12                 $rootScope.contract.numCampaigns({}, function(
                       error, result){
13                   if(!error)
14                   {
15                       var num = result.c[0];
16                       for (var i = 0; i <= num; i++)
```

```
                    {
                        $rootScope.contract.campaigns(i,
                           function(error, result){
                          if(!error)
                          {
                              $scope.campaigns.push(result);
                              $scope.$apply();
                          }
                          else
                          {
                              console.error(error);
                          }
                      });
                    }
                }
                else
                {
                    console.error(error);
                }
            });
        }

        $scope.changeView = function(view){
          $state.transitionTo(view);
        }
    }
]);
```

Ο controller αυτός κάνει κλήση της συνάρτησης numCampaigns() του smart contract για λάβει τον αριθμό εκστρατειών που υπάρχουν διαθέσιμες να εμφανίσει. Παρατηρούμε ότι η κλήση της συνάρτησης αυτής γίνεται ασύγχρονα, δηλαδή μέσω της μορφής :

```
$rootScope.contract.numCampaigns({}, function(error, result){
                if(!error)
                { … }
else
                {
                    console.error(error);
                }
});
```

Αυτός είναι ο μοναδικός τρόπος κλήσης συναρτήσεων smart contracts από την javascript και επαναλμβάνεται σε κάθε κλήση συνάρτησης του συμβολαίου.

Στην συνέχεια, έχουμε μια επαναληπτική κλήση της συνάρτησης campaigns( uint id ) η οποία επιστρέφει τα δεδομένα κάθε εκστρατείας και τα προωθεί στον controller campaignController.js ο οποίος αναλύεται παρακάτω :

```
'use strict';

angular.module('webUI')
    .controller('campaignController', ['$scope', '$state', '
       $rootScope',
        function($scope, $state, $rootScope) {
            var that = this;

            that.$onInit = function() {
                if (that.campaign) {
                    // campain 0 -> address to send funds
                    var addressTo = that.campaign[0];
                    // campain 1 -> id of campaign
                    var id = that.campaign[1].c[0];
                    // campain 2 -> goal of campaign
                    var goal = that.campaign[2].c[0];
                    // campain 3 -> days of campaign
                    var days = that.campaign[3].c[0];
                    // campain 4 -> number of funders
                    var funders = that.campaign[4].c[0];
                    // campain 5 -> amount gathered
                    var amount = web3.fromWei(that.campaign[5].c
                       [0],'ether');
                    // campain 6 -> name
                    var name = that.campaign[6];
                    // campain 7 -> description
                    var description = that.campaign[7];

                    web3.eth.getBlockNumber(function (error,
                       result) {
                      $scope.calcDuration(result, days);
                    });
```

Η συνάρτηση onInit καλείται κατά την κλήση του controller. Τα δεδομένα που περάστηκαν αποθηκεύονται σε μεταβλητές, και επίσης γίνονται μερικές απαραίτητες διαδικασίες εμφάνισης σωστών αποτελεσμάτων. Μια από αυτές είναι η μετατροπή σε ether από wei των χρηματικών ποσών. Επίσης, η μεταβλητή days , η οποία είναι αποθηκευμένη ως ο αριθμός του deadlineBlock, μετατρέπεται σε εναπομείνουσα χρονική διάρκεια μέσω κλήσης της web3.eth.getBlockNumber() και υπολογισμού της πράξης : $\frac{deadlineblock-currentblock}{5760}$.

```
                    $scope.campaign = {
                      'title': name,
                      'description': description,
                      'goal': goal,
                      'days': days,
                      'completed': '32%',
                      'amount': amount,
                      'address': addressTo,
```

```
                        'id': id,
                        'funders': funders,
                    }
                }

            }

            $scope.calcDuration = function(result, days) {
              var deadline = Math.max(0, Math.floor((days - result
                  ) / 5760));
              $scope.campaign.deadline = deadline;
              $scope.$apply();
            }

            $scope.contribute = function() {
                var value = web3.toWei('1','ether');
               $rootScope.contract.contribute(this.campaign.id, {
                   value: value}, function(error, result){
                   if(!error)
                   {
                       console.log(result);
                   }
                   else
                   {
                       console.error(error);
                   }
              });

            }
        }
    ]);
```

Τέλος, ορίζεται η λειτουργία του button contribute, η οποία με παρόμοιο τρόπο καλεί τη συνάρτηση contribute του smart contract για ένα σταθερό ποσό του 1 ether

### 6.3.3 Create Controller

Τέλος, αναλύεται η λειτουργία του controller υπεύθυνου για τη λειτουργία της καρτέλας create, ο οποίος στηρίζεται στις ίδιες αρχές λειτουργίας :

```
'use strict';

angular.module('webUI')
    .controller('createController', ['$scope', '$state', '
       $rootScope',
        function($scope, $state, $rootScope) {
```

```
6             var that = this;
7
8             $scope.campaign = {};
9
10            $scope.submit = function() {
11                var address = $scope.campaign.address;
12                var funds = $scope.campaign.funds;
13                var days = $scope.campaign.days;
14                var description = $scope.campaign.description;
15                var name = $scope.campaign.name;
16
17                $rootScope.contract.newCampaign(address, funds,
                      days, name, description, function(error,
                      result){
18                    if(!error)
19                    {
20                        $state.transitionTo('view');
21                    }
22                    else
23                    {
24                        console.error(error);
25                    }
26                });
27            }
28        }
29    ]);
```

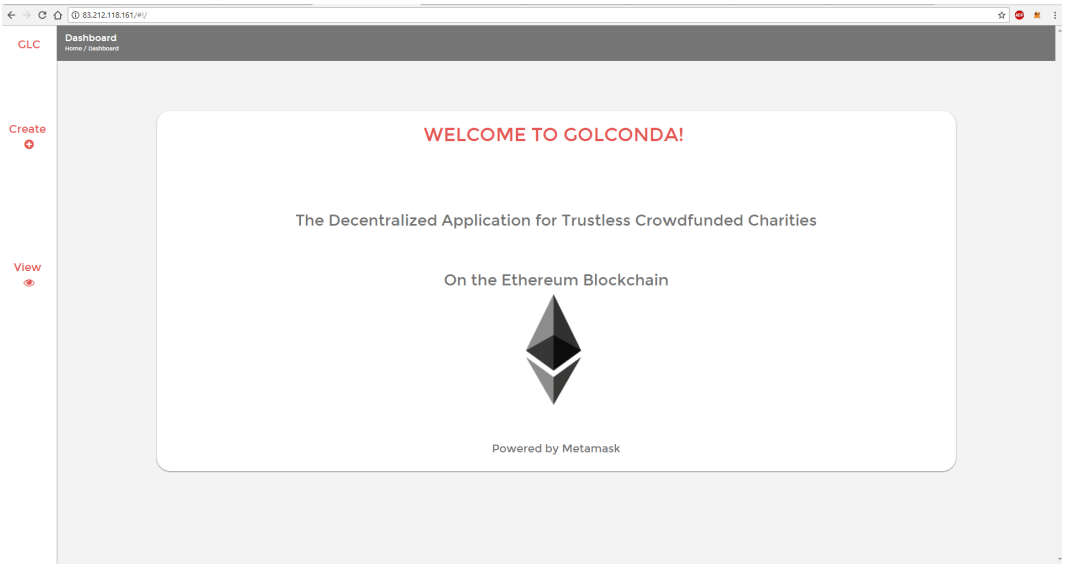
Ο controller λαμβάνει τα δεδομένα των κελιών της φόρμας υποβολής και τα περνάει ως ορίσματα στην συνάρτηση newCampaign(), η οποία καλείται επίσης ασύγχρονα.

Αυτές είναι οι βασικές μέθοδοι αλληλεπίδρασης του περιβάλλοντος διεπαφής με το smart contract. Παρατηρούμε ότι η ενσωμάτωση των εντολών αυτών είναι σχετικά απλή. Είναι επίσης αξιοσημείωτο ότι στην εφαρμογή αυτή, όλη η αποθήκευση πληροφοριών γίνεται εντός της blockchain και ότι αυτή λαμβάνει το ρόλο της υποδομής αποθήκευσης ( back-end).

## 6.4 Παρουσίαση εφαρμογής και οδηγίες χρήσης

### 6.4.1 Αρχική Σελίδα

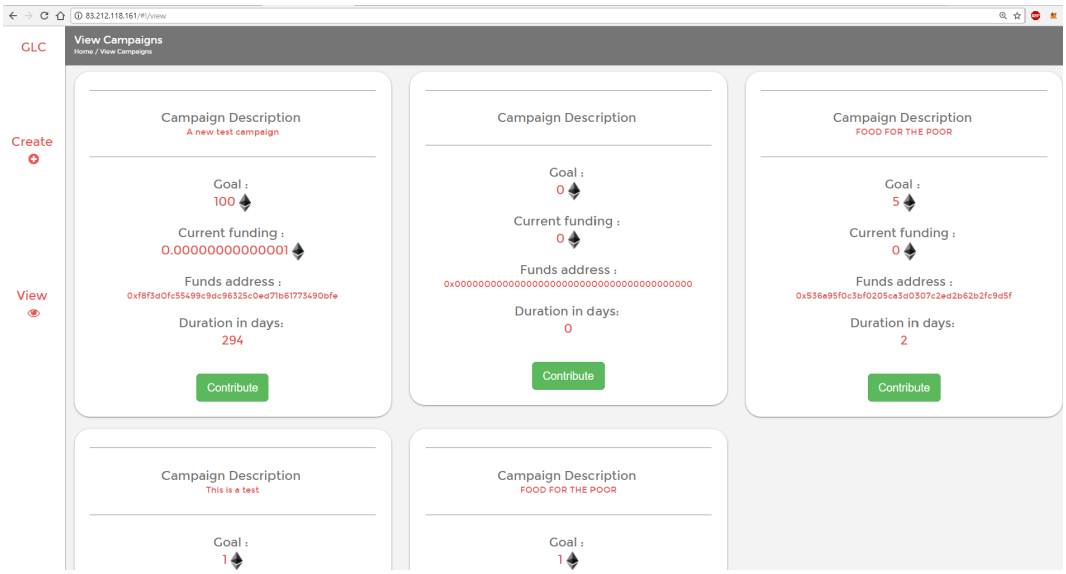
Η αρχική σελίδα [?] μας δίνει την περιγραφή της εφαρμογής



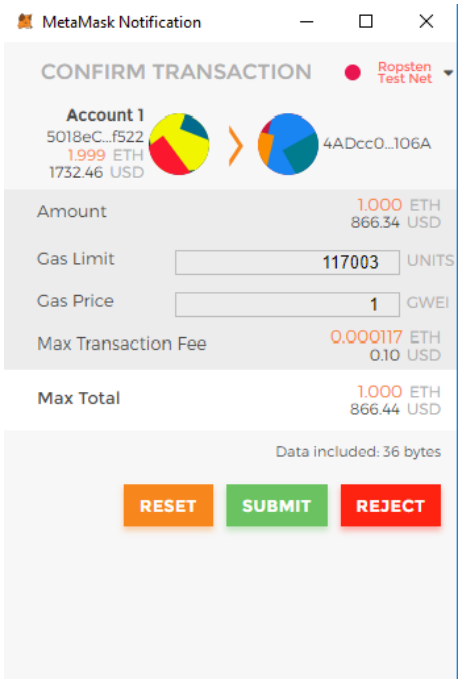

Από την στήλη αριστερά, μπορούμε να μεταβούμε στις 2 λειτουργίες της πλατφόρμας, δηλαδή επίβλεψη και δημιουργία νέων εκστρατειών χρηματοδότησης. Πρώτα όμως, είναι απαραίτητη η εγκατάσταση του πρόσθετου (plugin) Metamask στον browser μας.

### 6.4.2 Καρτέλα View

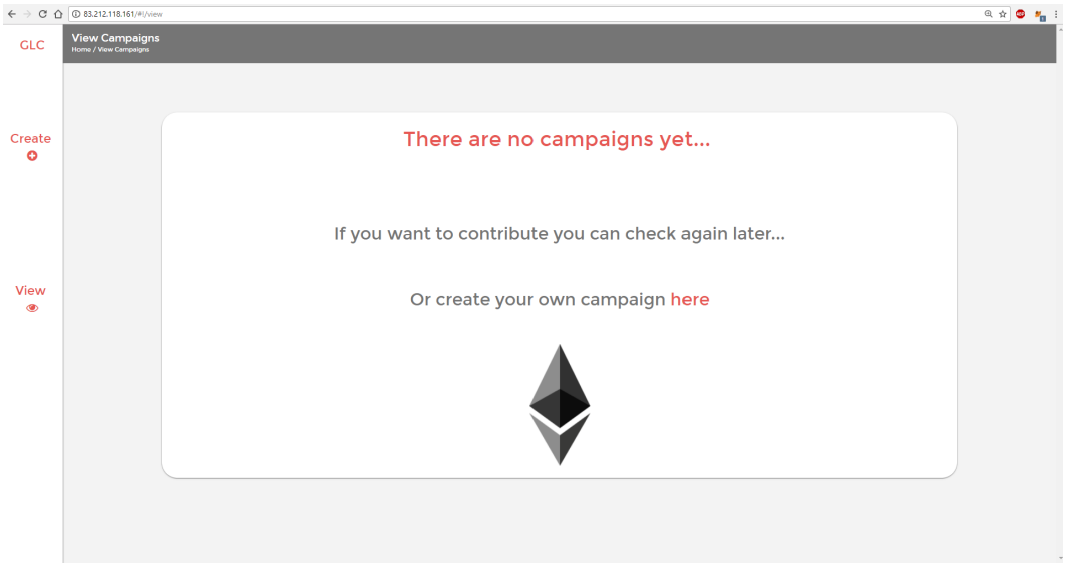
Η σελίδα αυτή εμφανίζει όλες τις υπάρχουσες εκστρατείες χρηματοδότησης, με τις σχετικές τους πληροφορίες όπως η διάρκεια και το ζητούμενο ποσό . Επίσης μας δίνει την επιλογή να συμμετέχουμε σε αυτές.




Για να έχουμε την ικανότητα συμμετοχής, θα πρέπει να διαθέτουμε μερικά ether στον λογαριασμό MetaMask μας. Επιλέγοντας Contribute για κάποια διαθέσιμη εκστρατεία, θα εμφανιστεί ένα παράθυρο επιβεβαίωσης συναλλαγής από το MetaMask. Αρκεί να επιλέξουμε submit και η συναλλαγή θα αποσταλλεί στο δίκτυο για επιβεβαίωση.

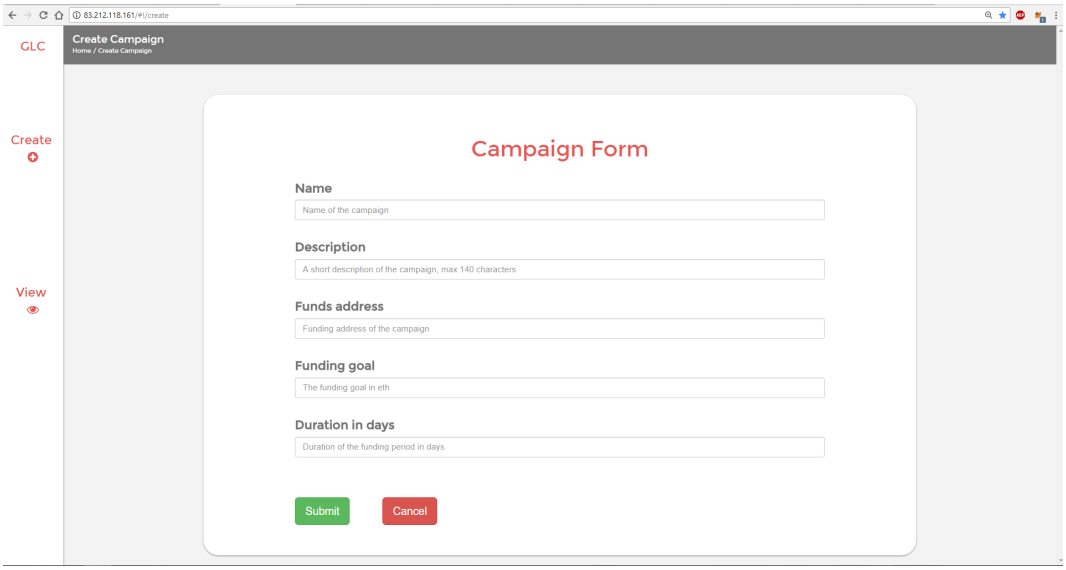


Σε λίγα λεπτά, το πεδίο current funding για την εκστρατεία που επιλέξαμε θα ανανεωθεί. Εαν η εκστρατεία είχε λήξει, η συναλλαγή θα αποτύχει και το ποσό θα μας επιστραφεί. Σε περίπτωση που δεν υπάρχουν διαθέσιμες εκστρατείες ή το Metamask κοιτάει σε λάθος δίκτυο, η σελίδα θα εμφανίζει το εξής μήνυμα :

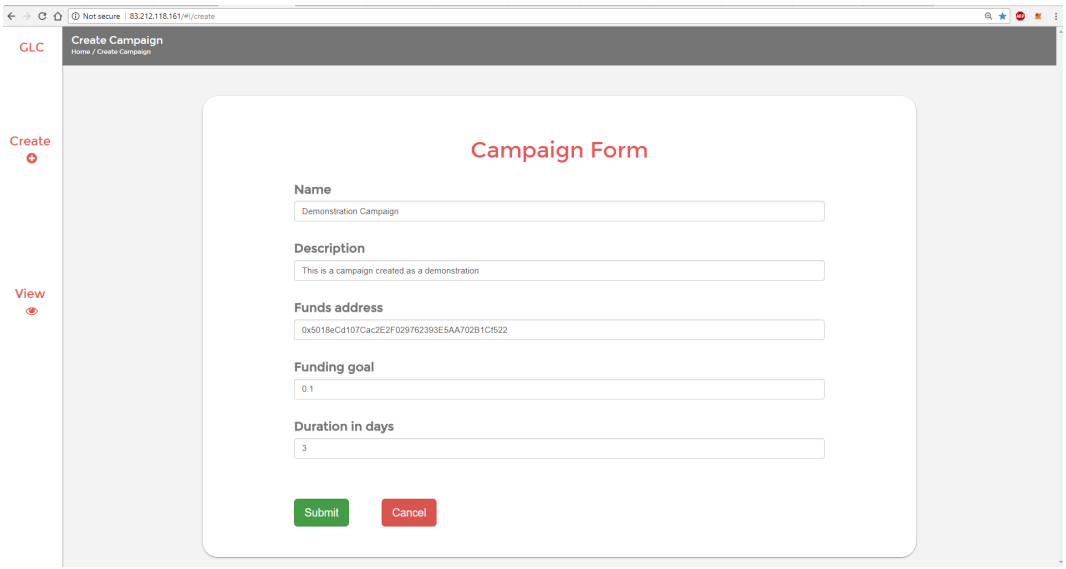



### 6.4.3 Καρτέλα Create

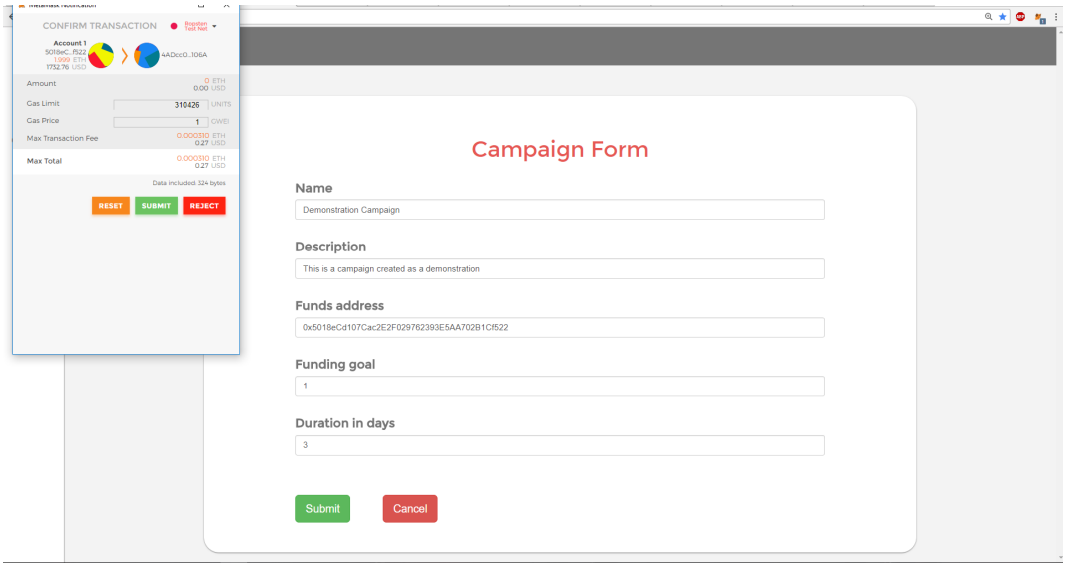
Μέσω της καρτέλας create ο χρήστης μπορεί να δημιουργήσει τη δική του εκστρατεία χρηματοδότησης



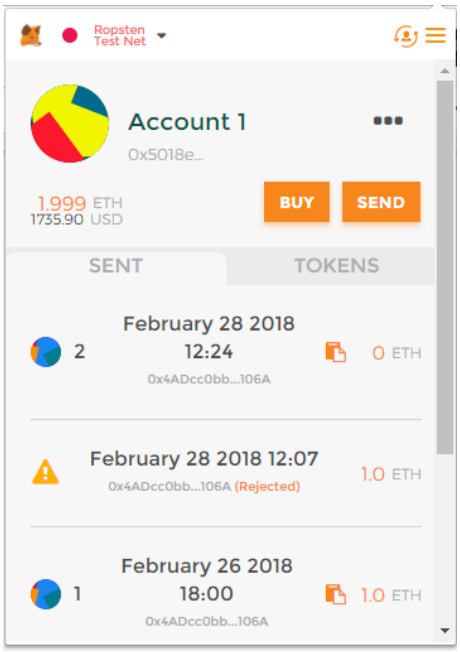
Είναι σημαντικό να εισαχθούν τα κατάλληλα στοιχεία στις σωστές θέσεις καθώς διαφορετικά η δημιουργία θα αποτύχει. Συγκεκριμένα, το funds address θα πρέπει να είναι δεκαεξαδικός αριθμός τύπου διεύθυνσης ethereum. Τα υπόλοιπα στοιχεία μπορούν να έχουν οποιαδήποτε τιμή αλλά καλό θα είναι να ακολουθούν τις υποδείξεις ώστε να είναι σωστή η λειτουργία της εκστρατείας. Εφ'όσον εισάγουμε τα στοιχεία, επιλέγουμε submit.



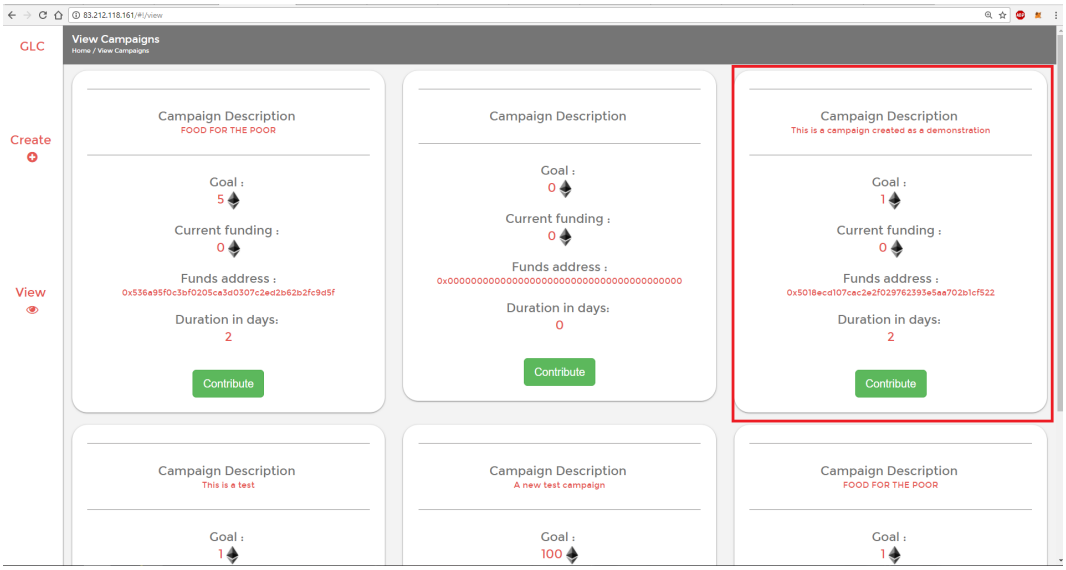

Μας ζητείται και πάλι η επιβεβαίωση συναλλαγής από το MetaMask, ωστόσο αυτή τη φορά το μεταφερόμενο ποσό είναι μηδέν. Το κόστος συναλλαγής είναι μόνο το απαιτούμενο gas για τη δημιουργία της εκστρατείας . Επιλέγοντας submit, η εκστρατεία μας θα πρέπει να εμφανιστεί στην καρτέλα view μετά από μερικά λεπτά.




Το παράθυρο του metamask μας δείχνει επίσης την κατάσταση των συναλλαγών που έχουμε υποβάλλει



Τέλος, επιβεβαιώνουμε την δημιουργία της εκστρατείας μας μέσω της καρτέλας view :

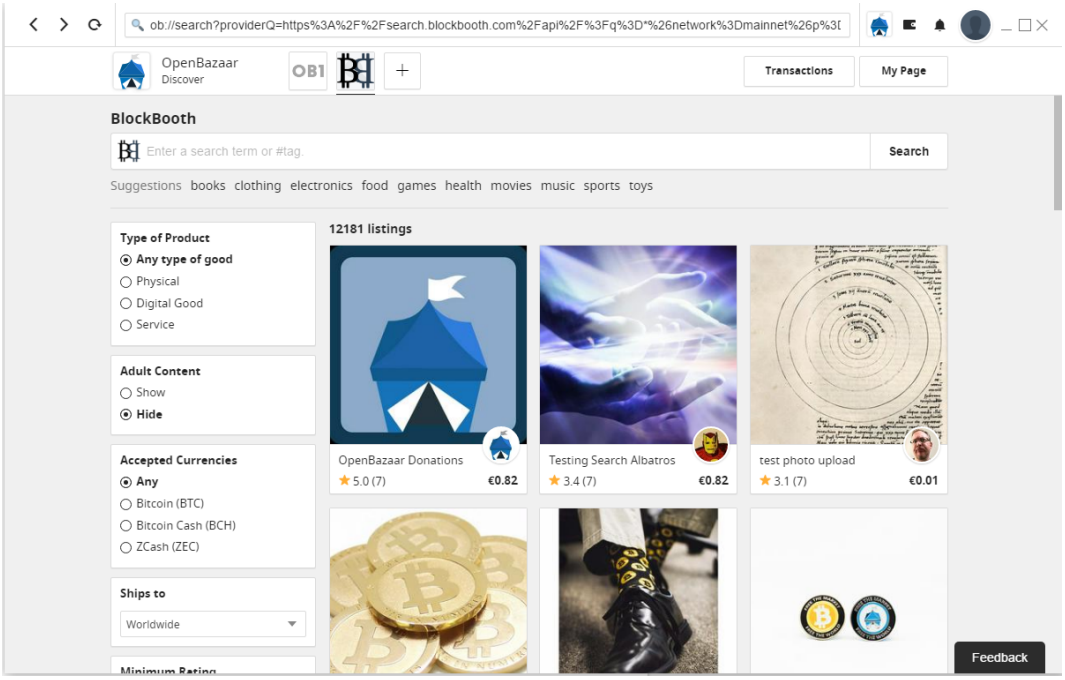



### 6.4.4 Παράδειγμα λειτουργίας

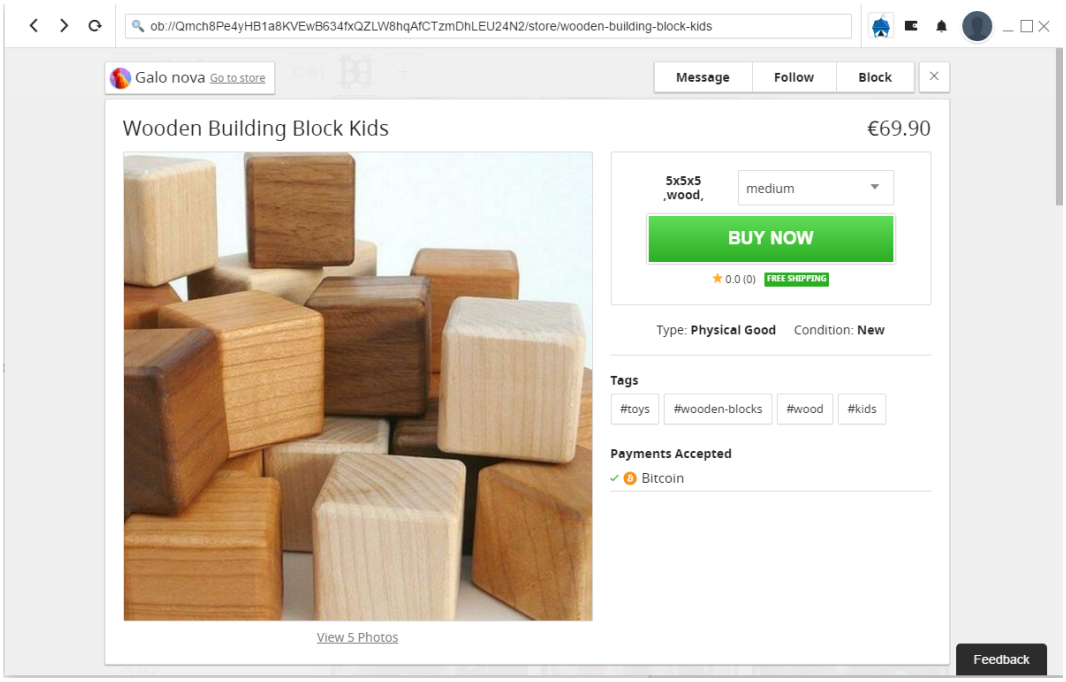
Δίνεται ένα παράδειγμα χρήσης της εφαρμογής από μια φιλανθρωπική για την εξυπηρέτηση των σκοπών της.

Αρχικά, η φιλανθρωπική εντοπίζει κάποιον πωλητή των αντικειμένων που την ενδιαφέρουν και αποδέχεται πληρωμές σε ether. Για ευκολία, προτείνεται η χρήση της ψηφιακής αποκεντρωμένης πλατφόρμας OpenBazaar. Καθώς η πλατφόρμα λειτουργεί μέσω του πρωτοκόλλου IPFS, είναι απαραίτητο να κατεβάσουμε τον ειδικό περιηγητή της εφαρμογής [?].

Στην συνέχεια εισάγει τις πληροφορίες όπως ονομασία, χώρα διαμονής και διεύθυνση αποστολής. Επιλέγοντας το κουμπί discover, εμφανίζονται τα διαθέσιμα προϊόντα :

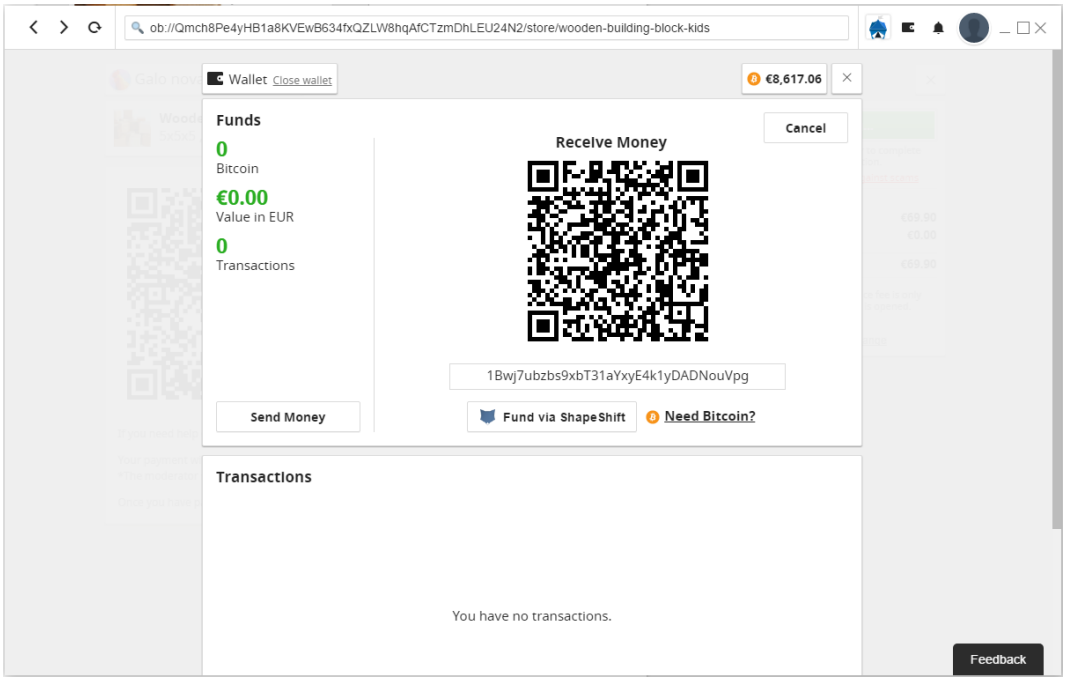



Έστω ότι η εκστρατεία άφορά την αγορά παιχνιδιών για τη διανομή τους σε διάφορα ορφανοτροφεία της χώρας. Επιλέγεται η κατηγορία toys και έστω ότι επιλέγεται η εξής αγγελία.

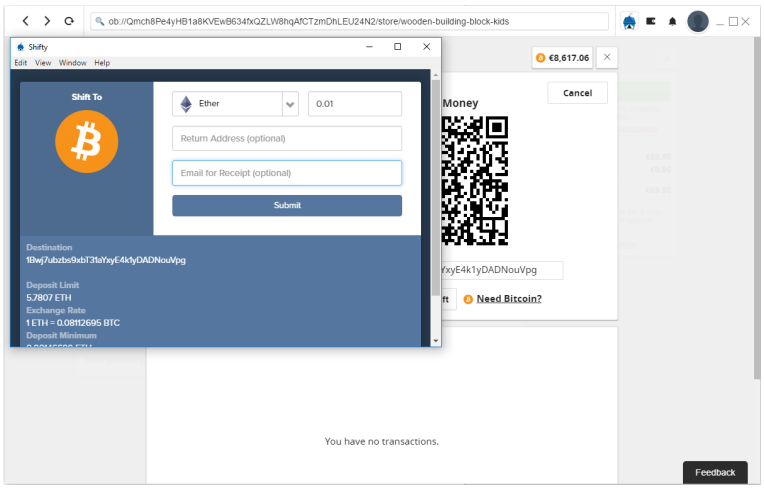


Επιλέγοντας buy now θα μεταβούμε στην οθόνη επιβεβαίωσης. Επιλέγοντας πληρωμή, θα μεταβούμε στο τελικό στάδιο της παραγγελίας.

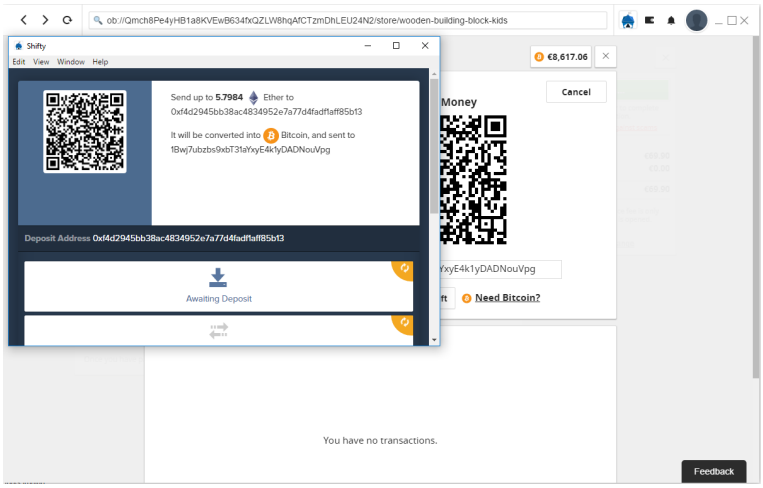
Η διεύθυνση άφορά bitcoins, ωστόσο δίνεται η δυνατότητα πληρωμής μέσω shapeshift, που είναι μια εφαρμογή που επιτρέπει τη μετατροπή κρυπτονομισμάτων από το ένα στο άλλο.



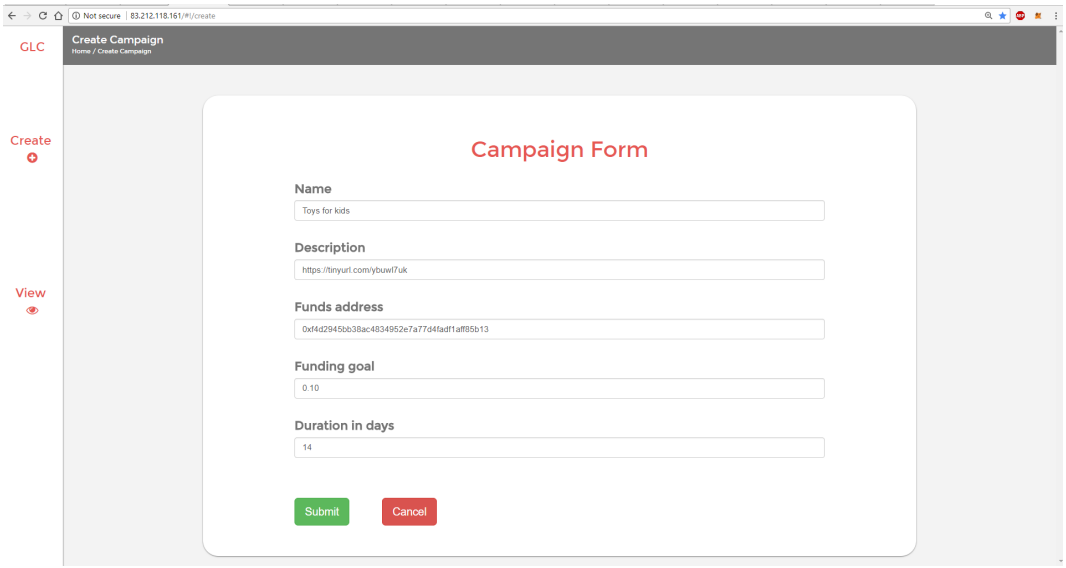

Επιλέγουμε fund via Shapeshift.



Το οποίο θα μας δώσει μια διεύθυνση ether για αποθήκευση των κεφαλαίων.




Τώρα η φιλανθρωπική εταιρία μπορεί να μεταβεί στην πλατφόρμα της Golconda και να δημιουργήσει μια εκστρατεία με στόχο την αγορά του προϊόντος χρησιμοποιώντας τη διεύθυνση που δόθηκε. Προτείνεται το όνομα που θα δοθεί να είναι το ίδιο με της εταιρίας και η περιγραφή να είναι ένας συντετμημένος σύνδεσμος στην ιστοσελίδα της ή σε κάποιο κανάλι social media όπου γίνεται η ενημέρωση.




Στη συνέχεια, επιλέγεται η υποβολή και η εκστρατεία είναι διαθέσιμη. Αν η εκστρατεία επιτύχει, τα ether θα αποσταλλούν αυτόματα στον προμηθευτή και το ποιόν θα αποσταλλλεί κατευθείαν στην εκστρατεία. Όλη η διαχείριση των χρημάτων υλοποιήθηκε εξολοκλήρου από το smart contract εξασφαλίζοντας την

χρηματοδότηση της χωρίς ο χρήστης να εμπιστεύεται τα κεφάλαια της δωρεάς στην φιλανθρωπική εταιρία.

# Κεφάλαιο 7

# Απόδοση Δικτύου

Για την αξιολόγηση της εφαρμογής που αναπτύχθηκε στα πλαίσια της εργασίας είναι απαραίτητο να εξετάσουμε πρώτα την γενικότερη απόδοση της μηχανής EVM και κατ'επέκταση αυτή του δικτύου Ethereum. Εδώ έχει νόημα να τονίσουμε ότι η λογική της σχεδίασης του, καθώς και των blockchains γενικότερα, δεν είναι η ταχύτητα των υπολογισμών ή η μείωση του απαιτούμενου αποθηκευτικού χώρου, αλλά η δυνατότητα συναλλαγών και από κοινού αλλαγών χωρίς την χρήση κεντρικών οργανισμών. Ενώ σε παραδοσιακές τεχνικές cloud storage ο επιπλέον χώρος χρησιμοποείται για υπολογιστική ταχύτητα και δυνατότητα ανάπτυξης, στο Ethereum κάθε μηχανή εκτελεί τις ίδιες γραμμές κώδικα για πολλαπλή επαλήθευση. Επίσης, η υλοποίηση μέσω δικτύων peer-to-peer θέτει φραγμούς στην μέγιστη ταχύτητα συγχρονισμού, εκτέλεσης και ανανέωσης των smart contracts , τέτοιους ώστε όλοι οι απομακρυσμένοι κόμβοι του δικτύου να έχουν την ευκαιρία να ενημερωθούν. Παρουσιάζουμε λοιπόν μια εικόνα [?] της απόδοσης του δικτύου Ethereum μέσω των παρακάτω μετρικών:

## 7.1 Αριθμός πραγματοποιούμενων συναλλαγών

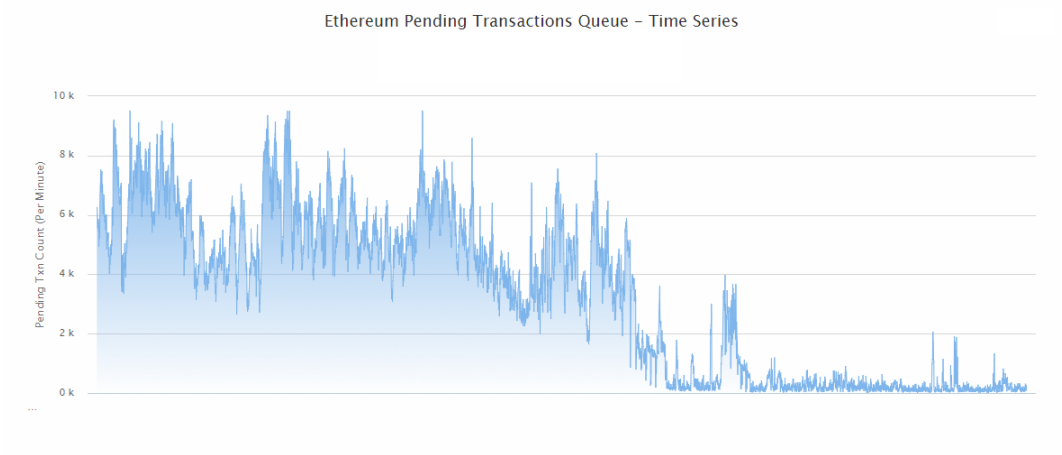
Παρακάτω παρατίθενται ιστορικά στοιχεία σχετικά με τον αριθμό υλοποιούμενων συναλλαγών την ημέρα. Θυμίζουμε ότι κάθε αλληλεπίδραση με smart contract αντιμετωπίζεται από το δίκτυο σαν συναλλαγή, συνεπώς αυτό είναι ένα άνω όριο στο πόσες φορές μπορεί κάποιος χρήστης να χρησιμοποιήσει την εφαρμογή μας την ημέρα. Παρατηρούμε την άνοδο του αριθμού των καθημερινών συναλλαγών κατά τη διάρκεια λειτουργίας του Ethereum, η οποία οφείλεται εν μέρει στην ανάπτυξη νέων χρηστικών εφαρμογών πάνω σε αυτό και εν μέρει στην πιο δημοφιλή

χρήση του ether ως νόμισμα. Το δίκτυο έχει αποδείξει την ικανότητα του να χειριστεί μέχρι και 1.350.000 συναλλαγές μέσα σε μια μέρα.

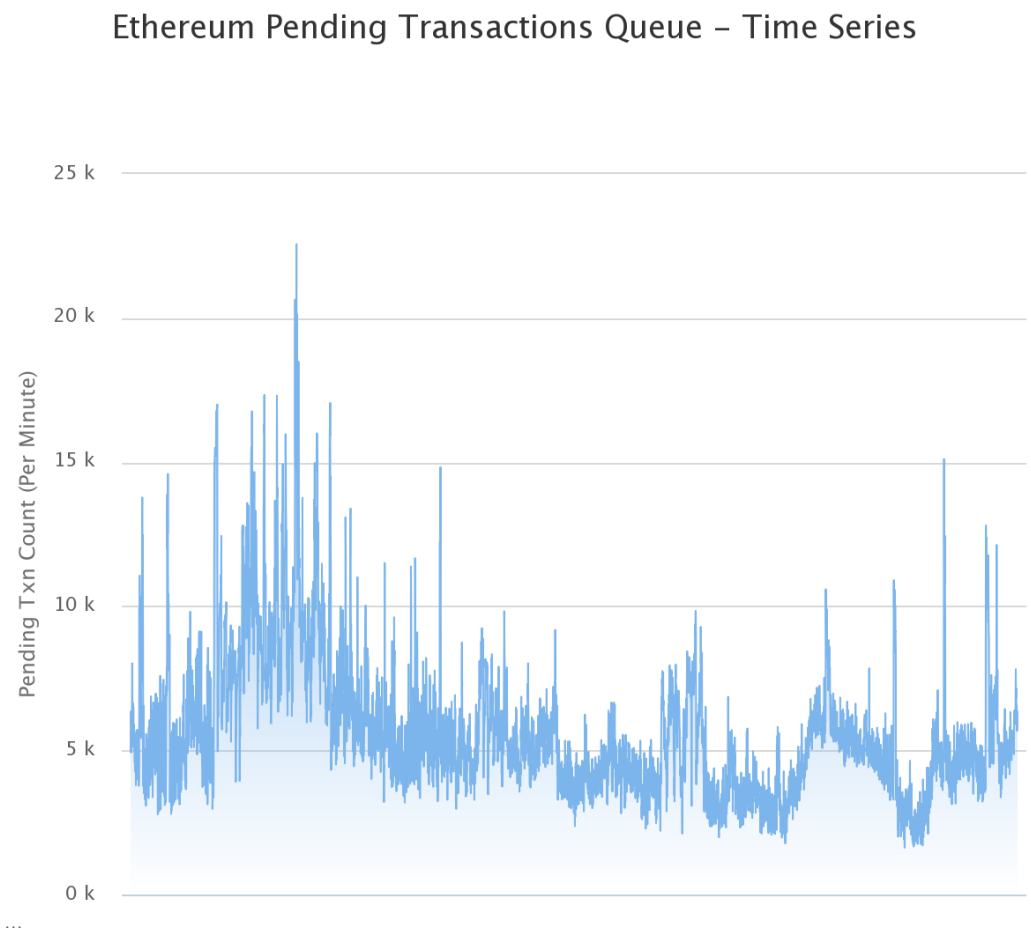

Σχήμα 7.1: Αριθμός συναλλαγών ανά ημέρα

## 7.2 Αριθμός συναλλαγών σε αναμονή

Τα παρακάτω γραφήματα, σε συνδυασμό με το προηγούμενο μας δίνει μια πιο ολοκληρωμένη εικόνα σχετικά με την απόδοση του δικτύου. Ο αριθμός συναλλαγών σε αναμονή συνδέεται άμεσα με το χρόνο αναμονής μεταξύ της αποστολής μιας εντολής στο πρόγραμμα μας και την εκτέλεση της στο δίκτυο και την εμφάνιση αποτελεσμάτων στο front-end. Σχετίζεται επίσης και με το κόστος λειτουργίας της εφαρμογής, καθώς με την αύξηση της τιμής gasprice μπορούμε να επιτύχουμε στόχους μέγιστης χρονικής περιόδου μέχρι την εκτέλεση των εντολών από το δίκτυο, το οποίο θα εξηγηθεί αργότερα. Παρατηρούμε ότι ο αριθμός αυτός κρατείται σε σταθερές σχετικά τιμές εκτός από περιόδους συνωστισμού, οι οποίο οφείλονται κυρίως στην έναρξη λειτουργίας καινούριων και δημοφιλών εφαρμογών, ή πιό συχνά, τη διένεξη ICOs, τα οποία όπως αναφέραμε είναι διαδικασίες crowdfunding νέων projects και συχνά έχουν περιορισμένη διάρκεια.




Σχήμα 7.2: Ιστορικά στοιχεία αριθμού συναλλαγών σε αναμονή

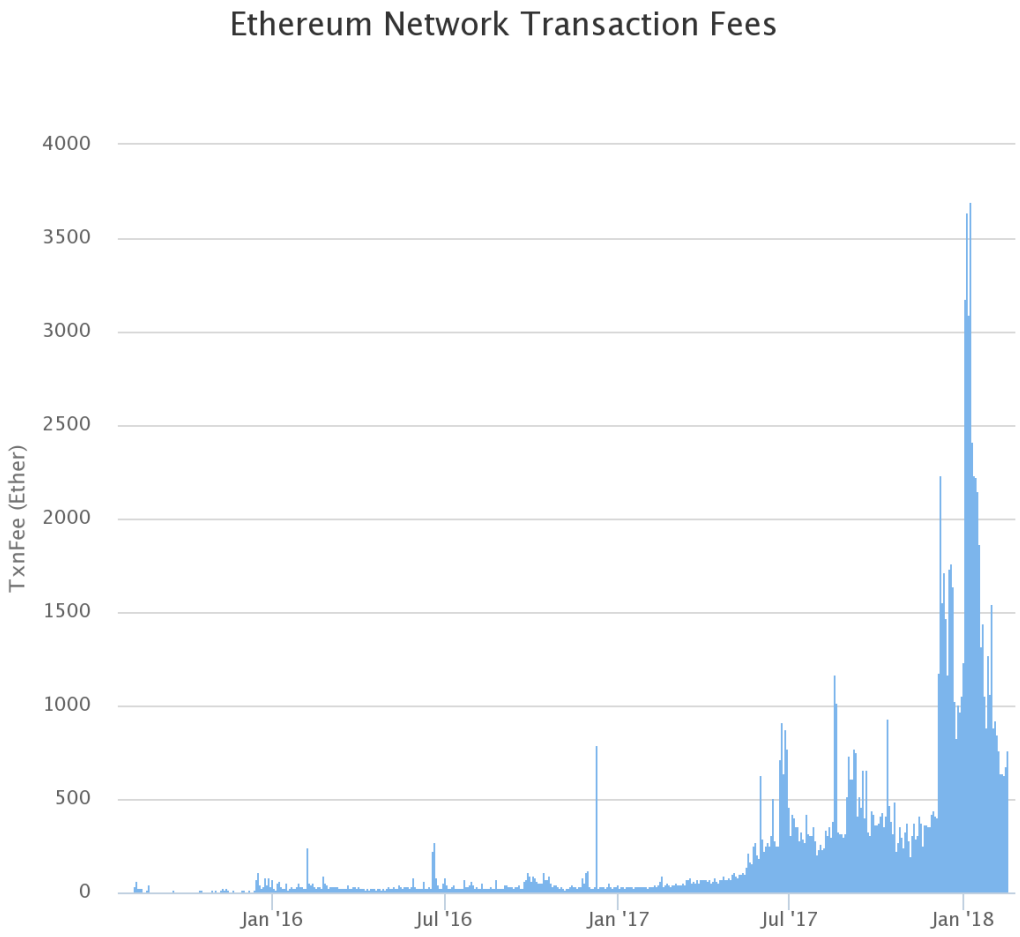



Σχήμα 7.3: Πρόσφατα στοιχεία αριθμού συναλλαγών σε αναμονή

## 7.3 Κόστος τελών συναλλαγών

Τα τέλη συναλλαγών υπολογίζονται ώς gasPrice * gasAmount. Η τιμή gasPrice τίθεται από τον χρήστη ενώ η τιμή gasAmount εξαρτάται από τις λειτουργίες ή τα δεδομένα που περιέχει η συναλλαγή. Όπως αναφέρθηκε και προηγουμένως, μέσω της τιμής gasPrice πετυχαίνουμε στόχους ταχύτητας εκτέλεσης. Αυτό συμβαίνει καθώς με υψηλό gasPrice κάνουμε τη συναλλαγή μας πιο θεμιτή προς τους miners ώστε να την συμπεριλάβουν στο επόμενο block. Έτσι, ακόμα και σε περιόδους συμφόρησης είναι βέβαιο πως με κάποια τιμή gasPrice η συναλλαγή μας μπορεί να εκτελεστεί ακόμα και άμεσα. Ωστόσο, όπως βλέπουμε από το γράφημα, τα τέλη συναλλαγών αυξάνονται καθώς το δίκτυο ethereum γίνεται πιο δημοφιλές και σε συνδυασμό με την πρόσφατη αύξηση της τιμής του ether μπορεί να οδηγήσει το κόστος λειτουργίας κάθε εφαρμογής σε δυσθεώρητα επίπεδα, ειδικά σε περιόδους

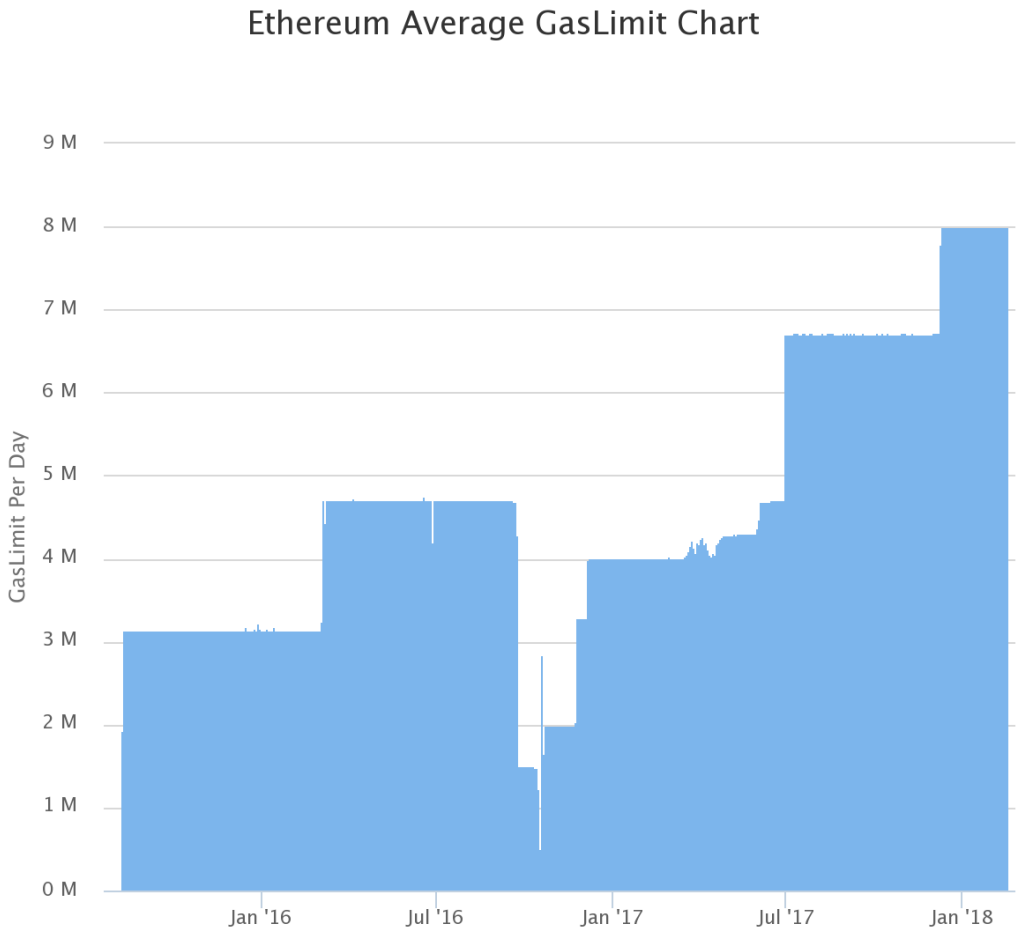
αιχμής. Σε μια προσπάθεια καταπολέμησης αυτού του φαινομένου, πολλοί developers θέτουν όριο gaslimit στις εφαρμογές τους, πάνω από το οποίο απορρίπτουν τις συναλλαγές, ώστε να διατηρείται το κόστος λειτουργίας του δικτύου χαμηλό και να μην δημιουργούν φαινόμενα συμφόρησης.



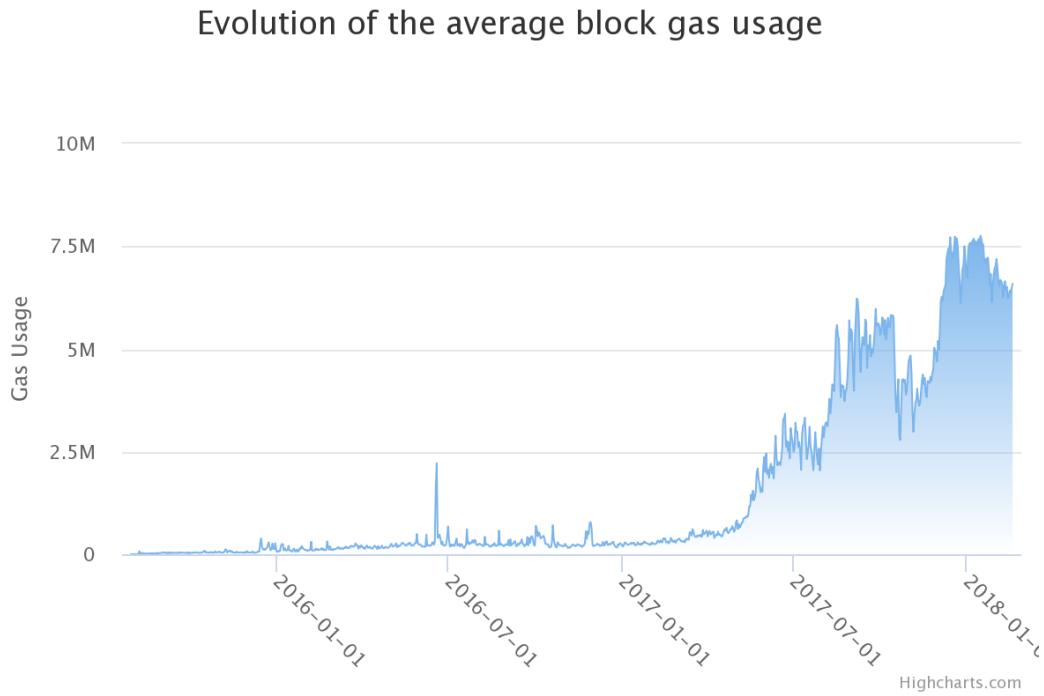

Σχήμα 7.4: Ημερήσια αξία τελών συναλλαγών σε ether

## 7.4 Μέγεθος gas/block και Όριο gas/block

Σε αντίθεση με το bitcoin, το ethereum δεν έχει μέγιστο όριο συναλλαγών σε kb. Έχει ωστόσο μέγιστο όριο gas ανά block, το οποίο επίσης προσαρμόζεται σύμφωνα με τις ανάγκες του δικτύου. Τη στιγμή αυτή βρίσκεται στα 8 εκ. wei ( $10^{-18}$) ether. Το διάγραμμα λοιπόν, μας δείχνει πόσο κοντά στην πλήρη χρήση των πόρων του δικτύου βρισκόμαστε ανά πάσα στιγμή και προκύπτει το συμπέρασμα ότι το δίκτυο τον τελευταίο καιρό βρίσκεται στα άκρα των δυνατοτήτων του. Το gasPrice, όπως αναφέραμε, μπορεί φυσικά να αυξηθεί ώστε να καλύψει περαιτέρω συναλλαγές. Από προγραμματιστικής απόψεως, με την άνοδο του gasLimit είναι δυνατόν να γράψουμε προγράμματα με περισσότερες γραμμές κώδικα και να εκτελέσουμε πιο υπολογιστικά απαιτητικές εντολές.



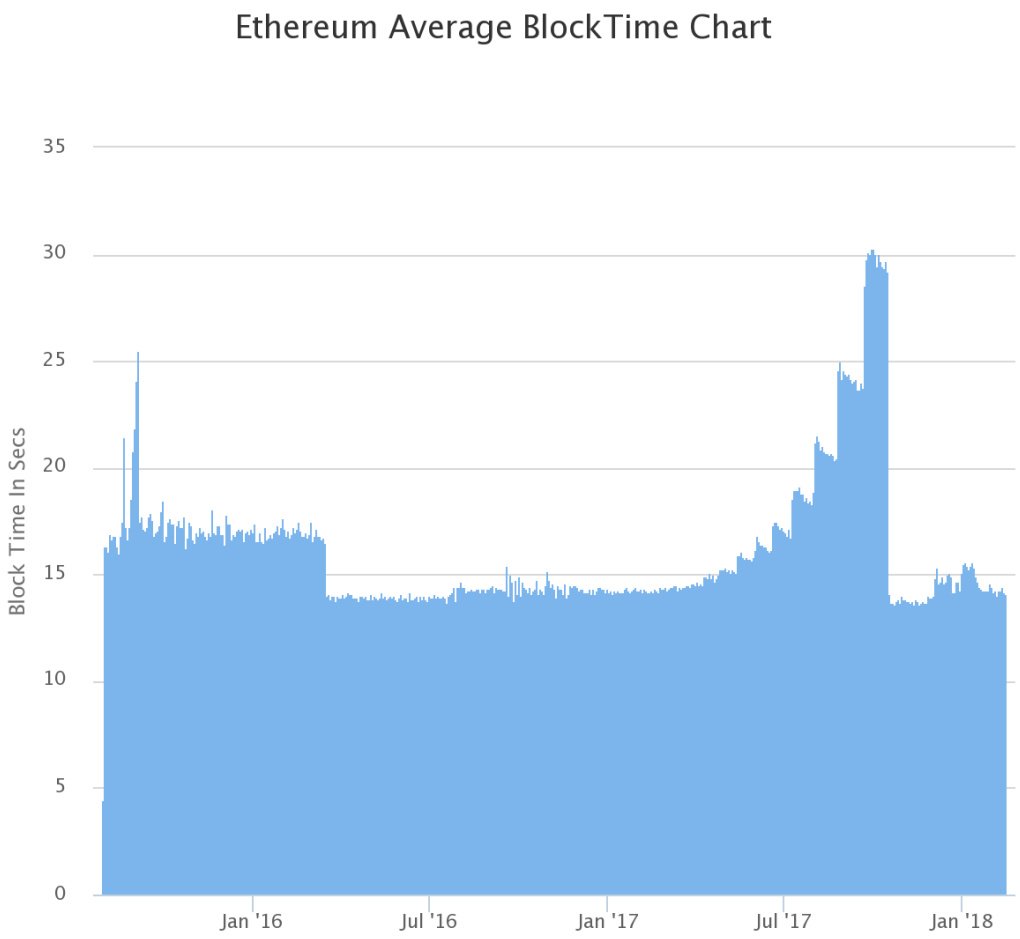

Σχήμα 7.5: Όριο gas ανά block




Σχήμα 7.6: Μέση χρήση gas ανά block

## 7.5 Περαιτέρω Μετρικές

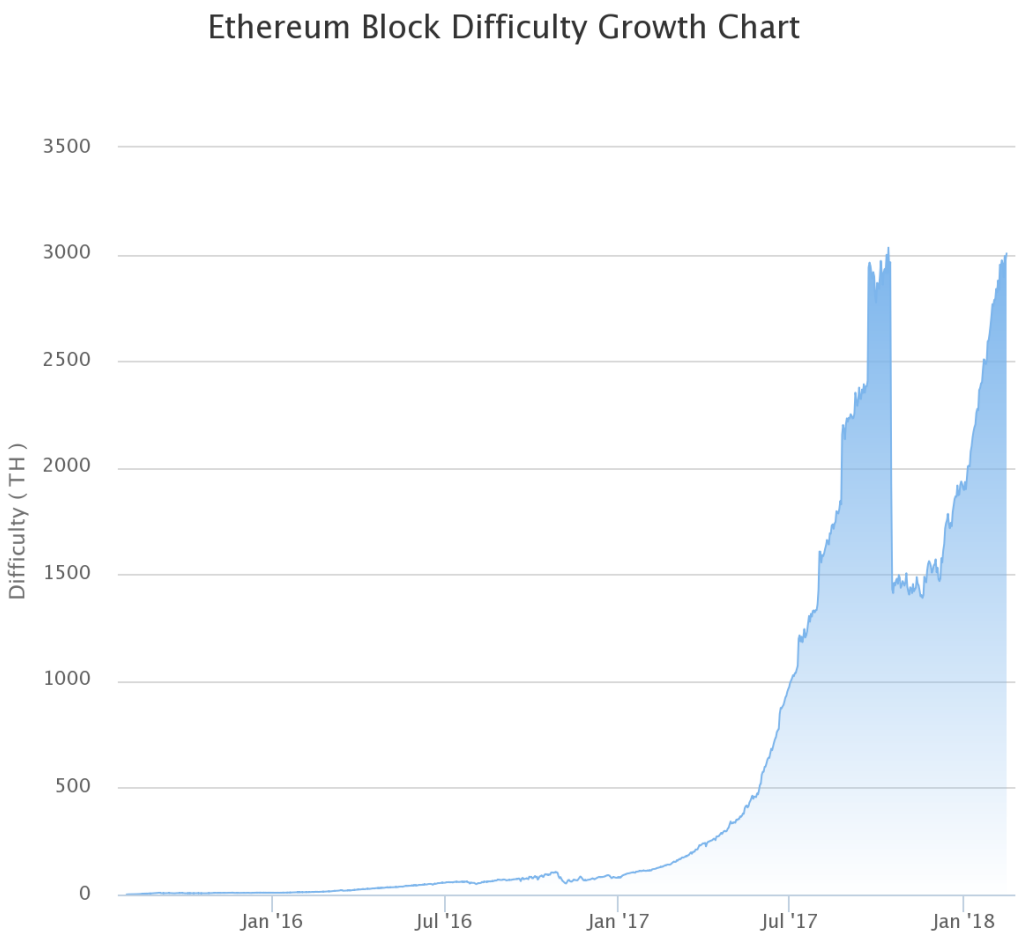
Ακολουθούν μερικές μετρικές που αφορούν περισσότερο τη λειτουργία του mining και η σημασία των οποίων θα εξηγηθεί συνοπτικά.




Σχήμα 7.7: Μέσος χρόνος εξόρυξης block

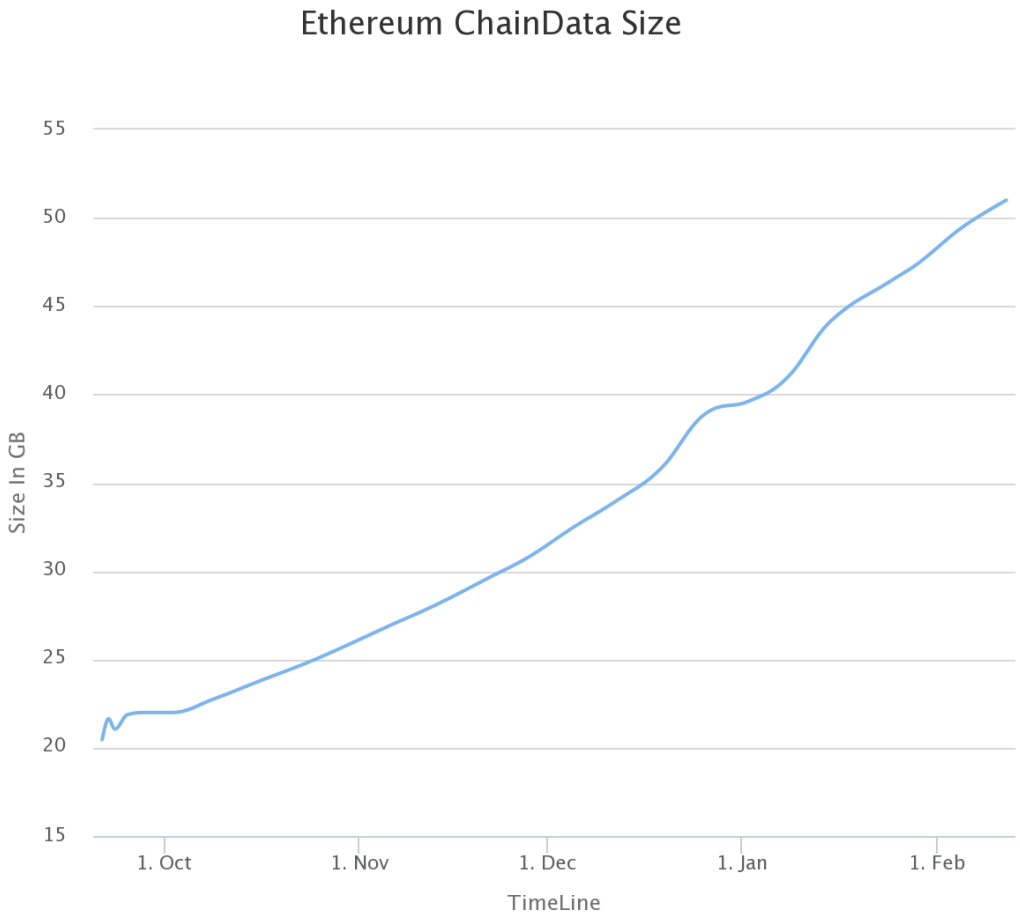
Παρατηρούμε ότι η δυσκολία και ο μέσος χρόνος εισαγωγής ανά block σε μια περίοδο αυξάνονταν σταδιακά ενώ στην συνέχεια μειώθηκαν απότομα για να συνεχίσουν να αυξάνονται με τον ίδιο ρυθμό. Αυτό συνέβη καθώς το δίκτυο προσπαθούσε να μεταβεί από το πρωτόκολλο proof-of-work, όπου η εισαγωγή νέων block γίνεται με τη χρήση υπολογιστικά απαιτητικών αλγορίθμων σε proof-of-stake, όπου η εισαγωγή νέων block γίνεται με τη δέσμευση ( staking ) κεφαλαίων του miner. Ο σκοπός ήταν η ενσωμάτωση νέων block μέσω proof-of-work να γίνεται σταδιακά πιο δύσκολή και το δίκτυο να μεταβεί οργανικά σε πλήρη χρήση proof-of-stake. Δυστυχώς όμως, τα προβλήματα της χρήσης proof-of-stake δεν κατάφεραν να λυθούν εγκαίρως και μέσω update η δυσκολία δημιουργίας νέων blocks επέστρεψε στα προηγούμενα επίπεδα ώστε να δοθεί περαιτέρω χρόνος στην ομάδα ανάπτυξης να το ενσωματώσει.

Τέλος, το Σχ. 6.9 μας δείχνει την αύξηση του μεγέθους της Ethereum Blockchain,




Σχήμα 7.8: Δυσκολία εξόρυξης block

δηλαδή του απαιτούμενου μεγέθους για τον συγχρονισμό ενός ελαφρού Ethereum Node. Ενώ η συνεχής αύξηση του blockgasLimit για την εκτέλεση όλο και περισσότερων συναλλαγών είναι επιθυμητή από την πλευρά των χρηστών, η συνέπεια είναι η με αυξανόμενο ρυθμό αύξηση του συνολικού μεγέθους της blockchain. Αυτό αποτρέπει στην ουσία τους miners με περιορισμένους πόρους από τον να διατηρούν πλήρη nodes, αφαιρώντας τους από το δίκτυο κάνοντας το λιγότερο αποκεντρωμένο καθώς μόνο λίγοι και εξοπλισμένοι με τελευταίας τεχνολογίας υποδομές θα μπορούν να επικυρώνουν τις συναλλαγές του δικτύου. Τα δεδομένα που δίνονται είναι τα απαιτούμενα για fast sync, δηλαδή για εφαρμογές wallets στον προσωπικό μας υπολογιστή. Ακόμα και τέτοιου είδους εφαρμογές θα γίνονται ολοένα και λιγότερο φιλικές προς τον χρήστη καθώς το ChainSize αυξάνεται.




Σχήμα 7.9: Συνολικό Μέγεθος Ethereum Blockchain για γρήγορο συγχρονισμό

## 7.6 Συμπεράσματα

Παρατηρούμε πως το δίκτυο Ethereum έχει επέλθει σε μια κατάσταση σχετικού κορεσμού. Καθώς η ανάπτυξη εφαρμογών πάνω σε αυτό γίνεται ολοένα και πιο προσιτή, με τη διάδοση της γλώσσας Solidity και η αύξηση της τιμής προκαλεί περισσότερες συναλλαγές, τα blocks είναι πλήρη, οι συναλλαγές σε αναμονή ανα πάσα στιγμή σταδιακά αυξάνονται και αυτό οδηγεί σε αύξηση του κόστους συναλλαγών και λειτουργίας παράλληλα με αύξηση του χρόνου αναμονής. Από πλευράς ανάπτυξης σημαίνει πως θα πρέπει να δωθεί ιδιαίτερη έμφαση στην μείωση του κώδικα που θα πρέπει να εκτελείται ανα πάσα στιγμή καθώς επιπλέον κώδικας μεταφράζεται ως κόστος στον χρήστη. Επίσης πως κάθε εφαρμογή πάνω στο Ethereum Network , όσο βελτιστοποιημένη και να είναι, περιορίζεται τελικά από την γενικότερη κατάσταση και επίδοση του δικτύου.

Τα προβλήματα αυτά είναι γνωστά στον χώρο και συνεχώς προτείνονται και αναπτύσσονται νέες λύσεις. Ήδη αναφέραμε την αλλαγή πρωτοκόλλου από proof-of-work σε proof-of-stake, το οποίο θα αφαιρέσει την ανάγκη για εξειδικευμένο εξοπλισμό και μεγάλη κατανάλωση ρεύματος παράλληλα με τη μείωση του κόστους των συναλλαγών. Μια άλλη τεχνολογία που αναπτύσσεται ονομάζεται sharding ( κατακερμάτιση ) [?] και θα παραλληλοποιήσει την εκτέλεση των smartcontracts ανάμεσα στα nodes, διατηρώντας όμως την ασφάλεια και τη συμφωνία μεταξύ πόρων του δικτύου. Το πρόβλημα αυτό της ανάπτυξης υποδομών (scaling ) είναι κοινό ανάμεσα σε όλα τα δημοφιλή blockchains και μεγάλο μέρος επιστημονικής έρευνας [?] απασχολείται με την εύρεση μιας ικανοποιητικής λύσης.

# Κεφάλαιο 8

# Μελλοντικές επεκτάσεις και συμπεράσματα

Οι τεχνολογίες blockchain έχουν γνωρίσει τεράστια ανάπτυξη από την δημιουργία τους το 2008 αποτελώντας έναν ταχύτατα μεταβαλλόμενο κλάδο με πληθώρα τεχνολογιών και προσεγγίσεων σε προβλήματα νομισμάτων, λογιστικής, ενέργειας, νομικής, φιλανθρωπίας κ.α. Η πλατφόρμα ethereum με τη δημιουργία της το 2014 επιτρέπει την υλοποίηση εφαρμογών smart contracts για κάθε είδους χρήση οι οποίες επωφελούνται από τα χαρακτηριστικά κρυπτογραφικής ασφάλειας και αποκέντρωσης.

Αναφερθήκαμε μέχρι στιγμής στα τεχνικά χαρακτηριστικά της πλατφόρμας ethereum, αναλύσαμε τις απαραίτητες τεχνικές ασφαλείας και σχεδιασμού εφαρμογών με χρήση smart contracts, αναπτύξαμε μια τέτοια εφαρμογή και τέλος εξετάσαμε τους δείκτες απόδοσης της, στα πλαίσια ολόκληρου του δικτύου.

Η εφαρμογή που αναπτύχθηκε, η οποία ονομάστηκε Golconda, είναι σε θέση να προσφέρει μια ρεαλιστική εναλλακτική μέθοδο χρηματοδότησης φιλανθρωπικών υπηρεσιών. Σε αυτή τη μορφή δεν λύνει ακόμα τα προβλήματα έλλειψης εμπιστοσύνης στις υπηρεσίες, καθώς δεν υπάρχει έλεγχος της χρήσης των κεφαλαίων που λαμβάνει. Συνεπώς η πρώτη, απαραίτητη επέκταση της είναι η διασύνδεση με τις διάφορες ηλεκτρονικές αποκεντρωμένες αγορές, όπως το OpenBazaar ώστε οι υπηρεσίες-χρήστες της Golconda να επιλέγουν ένα σύνολο από απαραίτητα αγαθά και να δημιουργούν μια αίτηση δωρεάς στην οποία θα είναι εμφανή, ώστε να είναι αποδεικνύεται στους δωρητές πως η δωρεά τους θα καταλήξει στο σωστό σκοπό.

Σε δεύτερο επίπεδο, για την αντιμετώπιση κακοηθών χρηστών που δημιουργούν

ψευδείς λογαριασμούς είτε από τη πλευρά των πωλητών, είτε από την πλευρά των υπηρεσιών είτε και από τις δύο, προτείνεται η ανάπτυξη ενός συστήματος φήμης, όπου οι χρήστες θα μπορούν να αξιολογούν υπηρεσίες και προμηθευτές ανάλογα με τις δράσεις τους, τις προηγούμενες επιτυχημένες αιτήσεις δωρεάς και την υλοποίηση αυτών. Ένα τέτοιο σύστημα θα επιτρέπει επίσης την αυτόματη επιλογή των πιο υποσχόμενων αιτήσεων για επιλογή και χρηματοδότηση τους από τρίτες εφαρμογές που μπορεί να περιέχουν μεθόδους δωρεάς σε φιλανθρωπία στην λειτουργία τους. Η υλοποίηση αυτού του συστήματος θα μπορούσε να γίνει μέσω ενός εσωτερικού token της εφαρμογής, το οποίο οι χρήστες θα μπορούν να χρησιμοποιούν κανονικά μέσω συναλλαγών για την αξιολόγηση. Αυτό το token [**?**], θα μπορούσε να διαμοιραστεί στους ενδιαφερόμενους χρήστες είτε ως διαμοιρασμό εκ' αέρος ( airdrop ) είτε ως μια αρχική προσφορά tokens, κατά την οποία οι χρήστες θα πρόσφεραν ether για την χρηματοδότηση της ανάπτυξης της εφαρμογής με αντάλλαγμα tokens.

Σε τρίτο και τελευταίο επίπεδο, η εφαρμογή θα μπορούσε να επανασχεδιαστεί ως ένας εντελώς αποκεντρωμένος και αυτόνομος οργανισμός - DAO. Η εφαρμογή σε αυτή τη μορφή θα επέτρεπε την πρόσληψη προγραμματιστών για την συντήρηση, ανανέωση και βελτίωση του κώδικα της. Η επιλογή αυτών θα γινόταν μέσω ψηφοφορίας των μελών της με την χρήση των tokens και η πληρωμή τους μέσω τον κεφαλαίων που συγκεντρώθηκαν κατά το ICO. Η λειτουργία της συνεπώς θα ήταν πλήρως στα χέρια των χρηστών ή μετόχων της. Θα μπορούσε επίσης να στηθεί και ως νόμιμη επιχείρηση σε κάποια χώρα, της οποίας το καταστικό να αντικατοπτρίζει την λειτουργία του κώδικα της.

Ο στόχος της εργασίας ήταν να παρουσιάσει όλες τις σχετικές πληροφορίες που είναι απαραίτητες για την κατανόηση των εννοιών της νέας αυτής τεχνολογίας και τη διευκόλυνση όσων σκοπεύουν να αναπτύξουν ασφαλείς, αποδοτικές και διατηρήσιμες αποκεντρωμένες εφαρμογές. Τα συμπεράσματα που προέκυψαν από την συγγραφή της εργασίας είναι τα εξής:

Η συγγραφή smart contracts επιτρέπει πλέον την διαχείριση χρηματικών ποσών με εντελώς καινούριες μεθόδους, δημιουργώντας νέα επιχειρηματικά μοντέλα και προϊόντα. Δημιουργεί όμως και νέες δυνατότητες κλοπής ή απώλειας αυτών των ποσών, συνεπώς η ασφάλεια και ο ενδελεχής έλεγχος πρέπει να έχουν πρωτεύουσα σημασία για κάθε προγραμματιστή. Χάρη στην γλώσσα υψηλού επιπέδου Solidity, καθώς και την πληθώρα έτοιμων πακέτων δημιουργίας περιβάλλοντος διεπαφής, η συγγραφή τέτοιων απλών dapps είναι χαμηλής δυσκολίας και πολυπλοκότητας κάτι που διευκολύνει το οικοσύστημα να αναπτυχθεί. Ωστόσο, αυτή η ανάπτυξη και η δημόσια προβολή των τελευταίων χρόνων έχει ωθήσει την πλατφόρμα στα όρια των τεχνικών δυνατοτήτων της. Μέχρι να δημιουργηθούν οι σωστές τεχνολογίες για την κλιμάκωση της, η αύξηση του κόστους λειτουργίας και η

καθυστέρηση επιβεβαίωσης συναλλαγών αποτελούν σημαντικό πρόβλημα. Ακόμα και αν η πλατφόρμα Ethereum δεν είναι αυτή που θα λύσει αυτά τα προβλήματα, πολλές από τις αρχές ανάπτυξης που αναλύθηκαν σε αυτή την εργασία θα συνεχίσουν να έχουν ισχύ καθώς ακόμα και ανταγωνιστικά περιβάλλοντα και οικοσυστήματα, οι βασικές αρχές της blockchain παραμένουν κοινές. Αν και η τεχνολογία blockchain δεν λύνει όλα τα σύγχρονα χρηματοπιστωτικά προβλήματα, είναι σίγουρο πως θα συνεχίσει να έχει μεγάλη χρήση στο μέλλον και οι γνώσεις που την αφορούν αποτελούν ένα πολύτιμο εργαλείο.

# Βιβλιογραφία


[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *metzdowd.com*, 2017.

[2] M. Swan, *Blockchain: Blueprint for a new economy*. O'Reilly Media Inc., 2015.

[3] V. Durham, "Namecoin.org." https://namecoin.org/. Accessed on 2017-11-1.

[4] V. Buterin, "Ethereum: A secure decentralised generalised transaction ledger," *https://ethereum.github.io*, 2015.

[5] S. Raval, *Decentralized Applications: Harnessing Bitcoin's Blockchain Technology*. O'Reilly Media Inc., 2016.

[6] https://www.stateofthedapps.com/. Accessed on 2018-2-3.

[7] http://www.instructables.com/id/Understanding-how-ECDSA-protects-your-data. Accessed on 2018-24-6.

[8] http://www.imponderablethings.com/2013/07/how-bitcoin-works-under-hood.html?m=1. Accessed on 2018-24-6.

[9] D. G. Andersen, "Ethereum: A secure decentralised generalised transaction ledger," *https://ethereum.github.io*, 2017.

[10] https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ. Accessed on 2018-2-3.

[11] https://github.com/ethereum/mist. Accessed on 2018-2-3.

[12] https://github.com/ethereum/go-ethereum/wiki/geth. Accessed on 2018-2-3.

[13] https://github.com/ethereum/web3.js/. Accessed on 2018-2-3.

[14] https://www.parity.io/. Accessed on 2018-2-3.

[15] http://truffleframework.com/. Accessed on 2018-2-3.

[16] https://github.com/trufflesuite/ganache-cli. Accessed on 2018-2-3.

[17] https://metamask.io/. Accessed on 2018-2-3.

[18] https://etherscan.io/. Accessed on 2018-2-3.

[19] https://zeppelin.solutions/. Accessed on 2018-2-3.

[20] http://solidity.readthedocs.io/en/develop/solidity-by-example.html. Accessed on 2017-11-6.

[21] E. Karapetsas, "Smart contract security: How to never break the blockchain," *DZone Guide to Application and Data Security, Volume II*, 2016.

[22] A. Pace. https://medium.com/@alpalpalp/chain-splits-and-resolutions-d3398bddf4ab. Accessed on 2018-2-3.

[23] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on ethereum smart contracts," *Universit'a degli Studi di Cagliari, Cagliari, Italy*, 2016.

[24] V. Buterin, "Thinking about smart contract security," *https://blog.ethereum.org/2016/06/19/thinking-smart-contract-security/*, 2016.

[25] A. Mavridou and A. Laszka, "Designing secure ethereum smart contracts: A finite state machine based approach," *22nd International Conference on Financial Cryptography and Data Security (FC 2018)*, 2016.

[26] A. Maslar. https://dappdaily.com/the-charitable-powers-of-ethereum-13d50e4561b1. Accessed on 2018-2-3.

[27] https://coinmarketcap.com/. Accessed on 2018-2-3.

[28] O. Tomalka. https://medium.com/tomalka-me/ethereum-dapp-for-crowdfunded-loans-case-study-7903155c780d. Accessed on 2017-10-6.

[29] S. Palladino. https://blog.zeppelin.solutions/designing-the-architecture-for-your-ethereum-application-9cec Accessed on 2018-10-1.

[30] https://consensys.github.io/smart-contract-best-practices/software_engineering/. Accessed on 2017-11-6.

[31] E. Dimitrova. https://blog.colony.io/writing-upgradeable-contracts-in-solidity-6743f0eecc88. Accessed on 2017-10-6.

[32] https://www.openbazaar.org/. Accessed on 2018-23-2.

[33] https://github.com/harrymantelak/Golconda-thesis.

[34] http://solidity.readthedocs.io/en/develop/types.html. Accessed on 2017-10-6.

[35] M. Murthy. https://medium.com/@mvmurthy/full-stack-hello-world-voting-ethereum-dapp-tutorial-part-1-4 Accessed on 2017-10-6.

[36] M. Murthy. https://medium.com/@mvmurthy/full-stack-hello-world-voting-ethereum-dapp-tutorial-part-2-3 Accessed on 2017-10-6.

[37] http://remix.ethereum.org/. Accessed on 2018-15-2.

[38] https://angularjs.org/. Accessed on 2018-15-2.

[39] shoudaos. https://startangular.com/product/flatlogic-angular-material-dashboard/. Accessed on 2018-15-2.

[40] C. Mantelakis. http://users.auth.gr/mantchar. Accessed on 2018-2-3.

[41] https://www.openbazaar.org/download/. Accessed on 2018-2-3.

[42] https://github.com/ethereum/wiki/wiki/Sharding-FAQ. Accessed on 2018-2-3.

[43] REX. https://blog.rexmls.com/sharding-raiden-plasma-the-scaling-solutions-that-will-unchai Accessed on 2018-2-3.

[44] https://www.ethereum.org/crowdsale#tokens-and-daos. Accessed on 2018-2-3.