Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Εργαστήριο Υπολογιστικών Συστημάτων

Αποκεντρωμένη εφαρμογή για φοιτητικές ψηφοφορίες στο Midnight blockchain

Διπλωματική Εργασία

Δημήτριος Γ. Βασιλείου

Επιβλέπων: Νεκτάριος Κοζύρης, Καθηγητής ΕΜΠ

Συνεπιβλέπων: Αναστάσιος Κατσιγιάννης, Υποψήφιος Διδάκτορας ΕΜΠ

Αθήνα, Φεβρουάριος 2026

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Εργαστήριο Υπολογιστικών Συστημάτων

Αποκεντρωμένη εφαρμογή για φοιτητικές ψηφοφορίες στο Midnight blockchain

Διπλωματική Εργασία

Δημήτριος Γ. Βασιλείου

Επιβλέπων: Νεκτάριος Κοζύρης, Καθηγητής ΕΜΠ

Συνεπιβλέπων: Αναστάσιος Κατσιγιάννης, Υποψήφιος Διδάκτορας ΕΜΠ

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 3η Φεβρουαρίου 2026

.....	.....	.....
Νεκτάριος Κοζύρης Καθηγητής, ΕΜΠ	Αριστείδης Παγουρτζής Καθηγητής, ΕΜΠ	Άγγελος Κιαγιάς Καθηγητής, Πανεπιστήμιο του Εδιμβούργου

Αθήνα, Φεβρουάριος 2026

.....
Δημήτριος Βασιλείου
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Ευχαριστίες

Με την ολοκλήρωση της παρούσας διπλωματικής εργασίας, αισθάνομαι την ανάγκη να ευχαριστήσω θερμά όλους όσοι συνέβαλαν στην επιτυχή περάτωσή της.

Αρχικά, θα ήθελα να εκφράσω τις ειλικρινείς μου ευχαριστίες προς τον καθηγητή μου, κ. Νεκτάριο Κοζύρη, για την εμπιστοσύνη που μου έδειξε με την ανάθεση αυτού του ιδιαίτερα σύγχρονου θέματος. Η ενασχόληση με την τεχνολογία του Midnight blockchain αποτέλεσε για μένα μια μοναδική ευκαιρία να εμβαθύνω σε τεχνολογίες αιχμής.

Ιδιαίτερη μνεία οφείλω στον υποψήφιο διδάκτορα, κ. Αναστάσιο Κατσιγιάννη, ο οποίος αποτέλεσε την πηγή έμπνευσης για το θέμα της εργασίας. Τον ευχαριστώ θερμά για την αμέριστη συμπαράσταση, την υπομονή και την καθοδήγησή του σε κάθε στάδιο της έρευνας, καθώς η βοήθειά του στην επίλυση σύνθετων προβλημάτων υπήρξε καθοριστική.

Επιπλέον, θα ήθελα να ευχαριστήσω τον κ. Χρήστο Παλάσκα, μηχανικό της Input Output Global (IOG), για τις πολύτιμες τεχνικές του συμβουλές και τις καίριες παρεμβάσεις του. Η βαθιά γνώση του αντικειμένου και οι κατευθύνσεις που μου έδωσε ήταν ζωτικής σημασίας για την υλοποίηση της εφαρμογής.

Ακόμα, ευχαριστώ την ερευνήτρια, κ. Αικατερίνη Δόκα, για τις ουσιαστικές παρατηρήσεις και τις εύστοχες συμβουλές της κατά τη διάρκεια της συγγραφής της εργασίας.

Τέλος, θα ήθελα να ευχαριστήσω την οικογένεια μου στην οποία αφιερώνω και την παρούσα εργασία, για τη στήριξη, καθοδήγηση και αγάπη που μου προσφέρει καθ'όλη τη διάρκεια της ζωής μου αλλά και τους φίλους μου οι οποίοι παραμένουν πάντα δίπλα μου τόσο στις ευχάριστες αλλά και στις δύσκολες στιγμές.

Περίληψη

Οι παραδοσιακές μέθοδοι διεξαγωγής ψηφοφοριών, τόσο οι έγχαρτες όσο και οι συμβατικές ηλεκτρονικές, παρουσιάζουν σημαντικές προκλήσεις που αφορούν την ασφάλεια και τη διαφάνεια. Η εξάρτηση από κεντρικούς διακομιστές και βάσεις δεδομένων εγκυμονεί κινδύνους παραβίασης της ανωνυμίας των ψηφοφόρων, ενώ η συγκεντρωτική διαχείριση καθιστά τα αποτελέσματα ευάλωτα σε αλλοιώσεις ή ανθρώπινα σφάλματα καταμέτρησης, χωρίς να παρέχεται στους συμμετέχοντες ένας μηχανισμός ανεξάρτητης επαλήθευσης.

Στην παρούσα διπλωματική εργασία, σχεδιάζεται και υλοποιείται μια αποκεντρωμένη λύση βασισμένη στην τεχνολογία blockchain, η οποία διασφαλίζει εκ κατασκευής την αμεταβλητότητα και τη διαφάνεια. Συγκεκριμένα, η εφαρμογή αναπτύχθηκε στο οικοσύστημα του Midnight, ενός σύγχρονου blockchain που επικεντρώνεται στην προστασία της ιδιωτικότητας μέσω προηγμένων κρυπτογραφικών τεχνικών, όπως οι αποδείξεις μηδενικής γνώσης (Zero-Knowledge Proofs).

Η προτεινόμενη εφαρμογή επιτρέπει στα μέλη της φοιτητικής κοινότητας να λειτουργούν τόσο ως ψηφοφόροι όσο και ως διοργανωτές ψηφοφοριών. Το κύριο πλεονέκτημα του συστήματος είναι η πλήρης αποσύνδεση της ταυτότητας του χρήστη από την ψήφο του, η οποία επιτυγχάνεται με τη χρήση Merkle Trees. Η αξιοπιστία της διαδικασίας και οι κανόνες που τη διέπουν διασφαλίζονται μέσω ενός έξυπνου συμβολαίου (Smart Contract) γραμμένου στη γλώσσα Compact, η οποία επιτρέπει τον αυστηρό διαχωρισμό μεταξύ ιδιωτικών και δημόσιων δεδομένων. Η διεπαφή χρήστη και η λογική της εφαρμογής υλοποιήθηκαν με τις τεχνολογίες ReactJS και TypeScript, προσφέροντας ένα σύγχρονο και λειτουργικό περιβάλλον για την ασφαλή διεξαγωγή φοιτητικών διαδικασιών.

Λέξεις Κλειδιά: Ψηφοφορία, Ανωνυμία, Διαφάνεια, Blockhain, Αποκέντρωση, Midnight, Compact, Έξυπνα Συμβόλαια, Δένδρα Merkle, Συναρτήσεις Κατακερματισμού, Αποδείξεις Μηδενικής Γνώσης, Μάρτυρας, Δημόσια Κατάσταση, Ιδιωτική Κατάσταση

Abstract

Traditional voting methods, both paper-based and conventional electronic, present significant challenges in terms of security and transparency. Reliance on central servers and databases carries risks of violating voter anonymity, while centralized management makes results vulnerable to tampering or human counting errors, without providing participants with a mechanism for independent verification.

In this thesis, a decentralized solution based on blockchain technology is designed and implemented, which ensures immutability and transparency by design. Specifically, the application was developed in the Midnight ecosystem, a state of the art blockchain that focuses on privacy protection through advanced cryptographic techniques, such as Zero-Knowledge Proofs.

The proposed application allows members of the student community to act as both voters and vote organizers. The main advantage of the system is the complete separation of the user's identity from their vote, which is achieved through the use of Merkle Trees. The reliability of the process and the rules governing it are ensured through a Smart Contract written in the Compact language, which allows for a strict separation between private and public data. The user interface and application logic were implemented using ReactJS and TypeScript technologies, providing a modern and functional environment for the secure conduct of student procedures.

Keywords: Voting, Anonymity, Transparency, Blockhain, Decentralization, Midnight, Compact, Smart Contracts, Merkle Trees, Hash Functions, Zero-Knowledge Proofs, Witness, Public State, Private State.

Περιεχόμενα

Κατάλογος σχημάτων

Κεφάλαιο 1

Εισαγωγή

Η τεχνολογία blockchain είναι ένα σύστημα κατανεμημένων κόμβων που επιτρέπει την ασφαλή, διαφανή και ανθεκτική σε παραβιάσεις καταγραφή ψηφιακών συναλλαγών. Σε αντίθεση με τις παραδοσιακές κεντρικές βάσεις δεδομένων, ένα blockchain διατηρείται συλλογικά από ένα δίκτυο συμμετεχόντων, εξαλείφοντας την ανάγκη για μια ενιαία αξιόπιστη αρχή.

Κάθε μπλοκ στην αλυσίδα περιέχει μια ομάδα συναλλαγών, μια χρονική σήμανση και ένα κρυπτογραφικό hash του προηγούμενου μπλοκ. Μόλις προστεθούν οι πληροφορίες, είναι εξαιρετικά δύσκολο να τροποποιηθούν χωρίς τη συναίνεση του δικτύου. [4]

Κύρια χαρακτηριστικά του blockchain περιλαμβάνουν:

- **Αποκεντροποίηση** -- Τα δεδομένα αποθηκεύονται και επαληθεύονται σε πολλαπλούς κόμβους, αντί για έναν κεντρικό server.
- **Διαφάνεια** -- οι συναλλαγές είναι ορατές σε όλους τους συμμετέχοντες στο δίκτυο.
- **Ασφάλεια** -- κρυπτογραφικοί μηχανισμοί το καθιστούν ανθεκτικό σε απάτες και παραβιάσεις.
- **Μηχανισμοί Consensus** -- όπως οι Proof of Work και Proof of Stake, οι οποίοι εξασφαλίζουν συμφωνία μεταξύ των συμμετεχόντων.

Η τεχνολογία blockchain, που εισήχθη αρχικά με το Bitcoin το 2008 ως η ραχοκοκαλιά του ψηφιακού νομίσματος [6], έχει εξελιχθεί από τότε πολύ πέρα από τα κρυπτονομίσματα. Τώρα εφαρμόζεται σε τομείς όπως η διαχείριση της εφοδιαστικής αλυσίδας, η υγειονομική περίθαλψη, η επαλήθευση ταυτότητας και τα συστήματα ψηφοφορίας. Η παρούσα διατριβή επικεντρώνεται στο *Midnight*, μια πλατφόρμα blockchain που έχει σχεδιαστεί με ιδιαίτερη έμφαση στην ασφάλεια, την ιδιωτικότητα των χρηστών και τη διαφάνεια. Αξιοποιώντας προηγμένες τεχνικές κρυπτογράφησης, ιδίως αποδείξεις μηδενικής γνώσης, το Midnight επιτρέπει στους συμμετέχοντες να αποδεικνύουν την εγκυρότητα των πληροφοριών χωρίς να αποκαλύπτουν τα υποκείμενα δεδομένα [5]. Μάλιστα, σε αντίθεση με άλλα δημόσια blockchain συστήματα, το Midnight δίνει έμφαση στην προστασία των δεδομένων και των μετα-δεδομένων όπως είναι οι πληροφορίες μιας συναλλαγής, ενώ ταυτόχρονα φροντίζει ώστε τα προσωπικά δεδομένα των χρηστών να μην εκτίθεται ποτέ στο δίκτυο, αντιθέτως να παραμένουν στις προσωπικές τους συσκευές [7]. Αυτά τα χαρακτηριστικά καθιστούν το Midnight ιδανική βάση για την ανάπτυξη μιας αποκεντρωμένης εφαρμογής ψηφοφορίας, όπου η εμπιστοσύνη, η εμπιστευτικότητα και η επαληθευσιμότητα είναι βασικές απαιτήσεις.

1.1 Κίνητρο

Η ψηφοφορία είναι μια θεμελιώδης διαδικασία που ενσωματώνει την πρακτική εφαρμογή της δημοκρατίας. Χρησιμεύει ως μηχανισμός λήψης αποφάσεων σε ένα ευρύ φάσμα πλαισίων,

συμπεριλαμβανομένων των κυβερνητικών εκλογών, των επαγγελματικών ενώσεων, των κοινωνικών οργανώσεων και των εκπαιδευτικών ιδρυμάτων. Η ακεραιότητα της διαδικασίας ψηφοφορίας είναι επομένως κρίσιμη, καθώς διασφαλίζει ότι τα αποτελέσματα αντικατοπτρίζουν την πραγματική βούληση των συμμετεχόντων. Για το λόγο αυτό, ένα σύστημα ψηφοφορίας πρέπει να είναι διαφανές, δίκαιο και ανώνυμο, ώστε να προστατεύεται η ταυτότητα και η ιδιωτικότητα των συμμετεχόντων. Τα παραδοσιακά συστήματα ψηφοφορίας, είτε σε χαρτί είτε ηλεκτρονικά, συχνά αντιμετωπίζουν σημαντικές προκλήσεις. Τα χαρτιά ψηφοφορίας είναι ευάλωτα σε ανθρώπινα λάθη και χειραγώγηση, ενώ τα κεντρικά ηλεκτρονικά συστήματα μπορεί να είναι ευάλωτα σε κυβερνοεπιθέσεις, παραβιάσεις και διαρροές δεδομένων . Στο πλαίσιο των φοιτητικών εκλογών, τα συμβατικά ηλεκτρονικά συστήματα απαιτούν συνήθως έναν κεντρικό διακομιστή και μια βάση δεδομένων όπου αποθηκεύονται τα προσωπικά στοιχεία και οι ψήφοι των φοιτητών .Αυτό δημιουργεί μια κατάσταση όπου οι φοιτητές πρέπει να αποκαλύψουν τις επιλογές τους στο ίδρυμα, γεγονός που θέτει σε κίνδυνο την αρχή της ανωνυμίας των ψηφοφόρων και μπορεί να υπονομεύσει την εμπιστοσύνη στη διαδικασία.

Επιπλέον, τα κεντρικά συστήματα εισάγουν ένα ενιαίο σημείο αστοχίας, πράγμα που σημαίνει ότι οποιαδήποτε παραβίαση, κακοδιαχείριση ή τεχνική δυσλειτουργία θα μπορούσε να θέσει σε κίνδυνο ολόκληρες τις εκλογές. Αυτές οι ευπάθειες υπογραμμίζουν την ανάγκη για εναλλακτικές προσεγγίσεις που μπορούν να διασφαλίσουν τόσο την ασφάλεια όσο και την ιδιωτικότητα, διατηρώντας παράλληλα τη διαφάνεια και την επαληθευσιμότητα. Μια λύση βασισμένη στο blockchain , ιδίως μια λύση που αξιοποιεί προηγμένες κρυπτογραφικές τεχνικές όπως οι αποδείξεις μηδενικής γνώσης , προσφέρει μια πολλά υποσχόμενη οδό για την αντιμετώπιση αυτών των ζητημάτων. Καταργώντας την ανάγκη για μια κεντρική αρχή και επιτρέποντας κρυπτογραφικά επαληθεύσιμη αλλά ιδιωτική ψηφοφορία, ένα τέτοιο σύστημα μπορεί να προστατεύσει την ανωνυμία, να αποτρέψει την απάτη και να ενισχύσει την εμπιστοσύνη στις ψηφοφορίες των φοιτητών.

Κεφάλαιο 2

Θεωρητικό Υπόβαθρο

Σε αυτή την ενότητα, παρουσιάζουμε το βασικό θεωρητικό υπόβαθρο που σχετίζεται με την παρούσα διατριβή. Ξεκινάμε με την εισαγωγή των βασικών εννοιών της τεχνολογίας blockchain, παρέχοντας τα θεμέλια για την κατανόηση των αποκεντρωμένων συστημάτων που αποτελούν τη βάση της προτεινόμενης εφαρμογής. Στη συνέχεια, εξετάζουμε τα βασικά εργαλεία από την επιστήμη της κρυπτογραφίας που χρησιμοποιούνται στην DApp, συμπεριλαμβανομένων των αποδείξεων μηδενικής γνώσης, των συναρτήσεων κατακερματισμού και των δέντρων Merkle, επισημαίνοντας τον ρόλο τους στην εξασφάλιση της ασφάλειας, της ακεραιότητας και της ιδιωτικότητας. Τέλος, εξετάζουμε λεπτομερώς το Midnight blockchain, εστιάζοντας στους μηχανισμούς του για τη διατήρηση της ιδιωτικότητας και της ανωνυμίας των χρηστών, οι οποίοι αποτελούν τη βάση για το ασφαλές και εμπιστευτικό σύστημα ψηφοφορίας που αναπτύχθηκε σε αυτή την εργασία.

2.1 Τεχνολογία Blockchain

Το blockchain είναι μία κατανεμημένη βάση δεδομένων που επιτρέπει την ασφαλή και ανθεκτική σε παραβιάσεις τήρηση δεδομένων χωρίς να υπάρχει η ανάγκη για μια κεντρική αρχή όπως για παράδειγμα η ύπαρξη ενός server. Ο έλεγχος των δεδομένων διαχέεται σε ένα δίκτυο από πολλαπλούς κόμβους (nodes). Κάθε κόμβος στο δίκτυο κρατάει ένα αντίγραφο όλων των δεδομένων και ακολουθεί ένα συμφωνημένο πρωτόκολλο για να ελέγχει την εγκυρότητα συναλλάγων και block. Αυτό που κάνει την τεχνολογία αυτή πολύ ισχυρή είναι ότι είναι πρακτικά αδύνατο να αλλοιώσει κάποιος τα δεδομένα , καθώς θα πρέπει να έχει στον έλεγχό του την πλειονότητα των κόμβων του δικτύου. Έτσι, τα δεδομένα παραμένουν πρακτικά αναλλοίωτα και οι χρήστες μπορούν να χρησιμοποιούν το δίκτυο δίχως την ανάγκη ύπαρξης μιας κεντρικής αρχής [8].

2.1.1 Μηχανισμοί consensus

Ένα σημαντικό στοιχείο στα συστήματα blockchain είναι οι μηχανισμοί consensus, οι οποίοι αποτελούν τα πρωτόκολλα που ακολουθούν οι συμμετέχοντες κόμβοι προκειμένου να έρθουν σε συμφωνία για την τρέχουσα κατάσταση των δεδομένων στο δίκτυο. Διακρίνουμε τους δύο πιο γνωστούς τέτοιους μηχανισμούς:

- **Proof Of Work(PoW)**: Το πρωτόκολλο αυτό χρησιμοποιείται από το Bitcoin και απαιτεί την ύπαρξη κόμβων που λέγονται miners οι οποίοι καλούνται να λύσουν κάποιο πολύπλοκο μαθηματικό πρόβλημα, το οποίο έγκυται στην αντιστροφή μιας συνάρτησης κατακερματισμού. Ο κόμβος που θα λύσει πρώτος το πρόβλημα λαμβάνει μια αμοιβή σε κρυπτονόμισα και είναι αυτός που θα προσθέσει το επόμενο μπλοκ στην αλυσίδα. Το PoW, παρότι είναι πολύ ασφαλές, απαιτεί κόμβους με πολύ υπολογιστική ισχύ και είναι

δαπανηρό σε ενέργεια ενώ ταυτόχρονα καθιστά πρακτικά ακατόρθωτο σε κάποιον απλό χρήστη να συμμετέχει στο πρωτόκολλο [5].

- **Proof Of Stake(PoS)**: Το PoS είναι πιο σύγχρονο από το PoW και δεν βασίζεται στη λύση πολύπλοκων μαθηματικών προβλημάτων. Αντιθέτως, ο κόμβος που θα εκλεγεί ως αυτός που θα προσθέσει στην αλυσίδα το επόμενο μπλοκ βασίζεται σε έναν συνδυασμό τύχης αλλά και του ποσού του οποίου ο κόμβος έχει δεσμεύσει [5]. Χρησιμοποείται σε blockchains όπως το Ethereum και το Cardano και ενώ είναι πιο φιλικό προς το περιβάλλον, ταυτόχρονα δίνει την δυνατότητα στον απλό χρήστη να συμμετέχει στο πρωτόκολλο καθώς μπορεί ο ίδιος να δεσμεύσει ένα ποσό κρυπτονομισμάτων σε κάποιο ευρύτερο σύνολο οντοτήτων που ονομάζονται **stakepools**. Ένα **stakepool** αποτελεί ένα σύνολο κόμβων και χρηστών που αθροίζουν συνολικά που το ποσό που κάνουν stake έτσι ώστε να έχουν μεγαλύτερη πιθανότητα εκλογής. Αν το pool είναι τελικά ο νικητής του γύρου, τα κέρδη μοιράζονται ποσοστιαία σε όλους τους συμμετέχοντες [2].
- **Proof Of Authority(PoA)**: Το PoA είναι μία ενεργειακά αποδοτικότερη λύση από τα δύο προηγούμενα πρωτόκολλα. Σε αντίθεση με το PoS όπου ο επόμενος validator εκλέγεται ψευδοτυχαία σε συνδυασμό με το ποσό που έχει δεσμεύσει, στο PoA οι validators καθορίζονται με βάση τη φήμη που έχουν στο δίκτυο. Κριτήρια για την επιλογή των validators αποτελούν η εγκυρότητα τους ως οντότητες, η αξιοπιστία τους καθώς και η θέλησή τους να ρισκάσουν τη φήμη τους επενδύοντας στο δίκτυο [9] [1].

2.1.2 Smart Contracts

Η ιδέα των Smart Contracs εμφανίστηκε το 1994 από τον Nick Szabo ο οποίος τα περιγράφει ως πρωτόκολλα συναλλαγών που εκτελούν τους όρους ενός συμβολαίου. Σκοπός τους είναι να ικανοποιήσουν κάποιους όρους όπως για είναι για παράδειγμα όροι πληρωμής ή όροι εμπιστευσιμότητα δίχως την ανάγκη ύπαρξης μιας κεντρικής αρχής [10].

Στα μοντέρνα blockchain συστήματα, τα Smart Contracts είναι προγράμματα γραμμένα σε κάποια γλώσσα προγραμματισμού όπως η Plutus, η Solidity ή η Compact, τα οποία εκτελούνται στο blockchain όταν συγκεριμένες συνθήκες που έχουν ήδη προκαθοριστεί λαμβάνουν χώρα. Τα smart contracts περιλαμβάνουν κανόνες για το πως αλληλεπιδρούν οι συμμετέχοντες του δικτύου, ωστόσο στο Midnight όπως θα δούμε παρακάτω, παίζουν καθοριστικό ρόλο στην προστασία των δεδομένων των χρηστών [5].

2.1.3 Αποκεντρωμένες Εφαρμογές (dApps)

Μία αποκεντρωμένη εφαρμογή είναι μια εφαρμογή η οποία δε βασίζεται σε κάποια κεντρική βάση δεδομένων και κάποιο κεντρικό server, αντιθέτως λειτουργεί πάνω σε ένα δίκτυο blockchain. Πολύ σημαντικό ρόλο σε αυτού του είδους τις εφαρμογές παίζουν τα smart contracts τα οποία και καθορίζουν το τι μπορεί να κάνει κάποιος χρήστη στην αποκεντρωμένη εφαρμογή [5], ενώ συγκεκριμένα στο Midnight διασφαλίζουν και ότι οι προσωπικές πληροφορίες κάθε χρήστη παραμένουν ασφαλείς και δεν εκτίθενται ποτέ στο δίκτυο, αντί αυτού παραμένουν στον τοπικό του υπολογιστή.

2.2 Εργαλεία από την επιστήμη της κρυπτογραφίας

2.2.1 Συναρτήσεις Κατακερματισμού

Οι συναρτήσεις κατακερματισμού είναι μαθηματικές συναρτήσεις που λαμβάνουν μια συμβολοσειρά εισόδου A και παράγουν μια συμβολοσειρά εξόδου σταθερού μεγέθους $B = H(A)$.

Το κύριο πλεονέκτημά τους έγκειται στη μονοκατευθυντική τους φύση: είναι υπολογιστικά αδύνατο να αντιστραφεί η συνάρτηση και να ανακτηθεί η αρχική είσοδος από την έξοδο. Με άλλα λόγια, δεδομένου του B, είναι πρακτικά αδύνατο να προσδιοριστεί το A [12]. Αυτή η ιδιότητα καθιστά τις συναρτήσεις κατακερματισμού ένα θεμελιώδες εργαλείο στην κρυπτογραφία για τη διασφάλιση της ακεραιότητας και της ασφάλειας των δεδομένων. Θα αναφέρουμε τις επιθυμητές ιδιότητες που πρέπει να έχει μια συνάρτηση κατακερματισμού H:

- **Αντίσταση Πρώτου Ορίσματος**: Είναι υπολογιστικά αδύνατο δεδομένης μιας τιμής y στο πεδίο τιμών της H, να βρεθεί x τέτοιο ώστε $(x) = y$. Δηλαδή είναι δύσκολη η αντιστοφή της συνάρτησης.
- **Αντίσταση Δεύτερου Ορίσματος**: Είναι υπολογιστικά δύκολο για κάποιο συγκεκριμένο στοιχείο x, ένα στοιχείο x' διαφορετικό του x, αλλά με $H(x) = H(x')$.
- **Δυσκολία Εύρεσης Συγκρούσεων**: Είναι υπολογιστικά δύσκολο να βρεθούν δύο διαφορετικές τιμές x, x' τέτοιες ώστε $H(x) = H(x')$. [12]

Αρκετές γνωστές συναρτήσεις κατακερματισμού χρησιμοποιούνται ευρέως στην πράξη, συμπεριλαμβανομένων των SHA-256, SHA-3 και του αλγορίθμου MD5 (αν και ο MD5 θεωρείται ανασφαλής για σύγχρονες εφαρμογές). Αυτές οι συναρτήσεις χρησιμοποιούνται σε ψηφιακές υπογραφές, συστήματα blockchain, κατακερματισμό κωδικών πρόσβασης και άλλες εφαρμογές κρίσιμης σημασίας για την ασφάλεια.

Μια θεμελιώδης εφαρμογή των συναρτήσεων κατακερματισμού είναι τα *σχέδια δέσμευσης*. Ας υποθέσουμε ότι ένας χρήστης διαθέτει μια μυστική τιμή S που πρέπει να παραμείνει ιδιωτική. Ο χρήστης μπορεί να δεσμευτεί για αυτό το μυστικό υπολογίζοντας τον κατακερματισμό του, $H(S)$. Η προκύπτουσα τιμή, $H(S)$, μπορεί στη συνέχεια να δημοσιευτεί, για παράδειγμα, σε ένα blockchain, λειτουργώντας ως κρυπτογραφική δέσμευση για το S χωρίς να αποκαλύπτεται το ίδιο το μυστικό. Αργότερα, οποιοσδήποτε μπορεί να επαληθεύσει μια δηλωθείσα δέσμευση ελέγχοντας ότι ο παρεχόμενος κατακερματισμός ταιριάζει με την προηγουμένως δημοσιευμένη τιμή $H(S)$, διασφαλίζοντας την ακεραιότητα της δέσμευσης και διατηρώντας το μυστικό S εντελώς κρυφό.

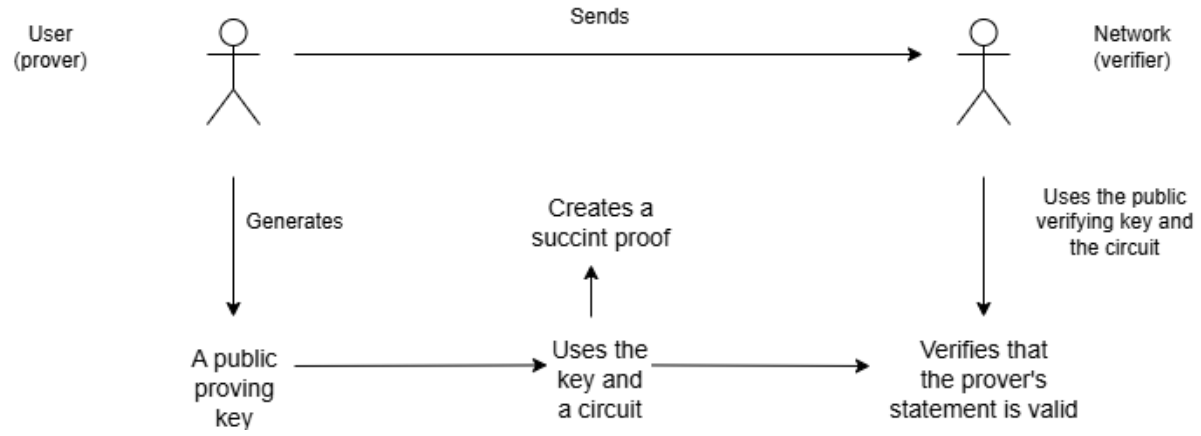
2.2.2 Αποδείξεις Μηδενικής Γνώσης

Η απόδειξη μηδενικής γνώσης είναι μια μέθοδος με την οποία ο αποδεικνύων(prover) μπορεί να πείσει τον επαληθευτή(verifier) ότι μια συγκεκριμένη δήλωση είναι αληθής, χωρίς να αποκαλύψει καμία πρόσθετη πληροφορία πέραν της αλήθειας της δήλωσης. Με άλλα λόγια, ο prover αποδεικνύει τη γνώση ενός μυστικού (ή της λύσης ενός προβλήματος) χωρίς να μοιράζεται το ίδιο το μυστικό.[5] Για παράδειγμα, μια ZKP θα μπορούσε να σας επιτρέψει να αποδείξετε ότι έχετε δικαίωμα ψήφου σε φοιτητικές εκλογές χωρίς να αποκαλύψετε την ταυτότητά σας ως φοιτητής.

Αν και το μαθηματικό υπόβαθρο των αποδείξεων μηδενικής γνώσης ξεφεύγει από το σκοπό αυτής της εργασίας, αξίζει να περιγραφεί σε υψηλό επίπεδο ο τρόπος λειτουργίας τους:

- Συμμετέχουν δύο μέρη: ο **prover**, που κατασκευάζει την απόδειξη θέλοντας να αποδείξει έναν ισχυρισμό ή μια πρόταση, και ο **verifier**, που ελέγχει την εγκυρότητά της. Ο prover κατέχει μια μυστική πληροφορία, τον *witness*, και πρέπει να πείσει τον verifier για έναν ισχυρισμό χωρίς να αποκαλύψει τον witness [5].
- Ο ισχυρισμός του prover (ο οποίος θα μπορούσε να είναι μια πρόταση της μορφής: <<ανήκω στο σύνολο των έγκυρων ψηφοφόρων>>) αναπαρίσταται μέσω ενός μαθηματικού κυκλώματος, γνωστού ως zero-knowledge circuit [5].

- Στην αρχή πραγματοποιείται μια *φάση αρχικοποίησης* , κατά την οποία δημιουργούνται τα αντίστοιχα *proving* και *verifying keys*. [5]
- Ο prover χρησιμοποιεί τον witness και το proving key για να κατασκευάσει την απόδειξη.
- Ο verifier, χρησιμοποιώντας το verifying key αλλά και το μαθηματικό κύκλωμα, ελέγχει αν η απόδειξη είναι έγκυρη. Αν επαληθευθεί, τότε ο ισχυρισμός του prover θεωρείται αληθής· διαφορετικά, είτε ο ισχυρισμός είναι λανθασμένος είτε ο prover δεν γνωρίζει στην πραγματικότητα έναν έγκυρο witness [5].



Σχήμα 2.1: Ένα πρωτόκολλο μηδενικής γνώσης σε high level

2.2.3 Δένδρα Merkle

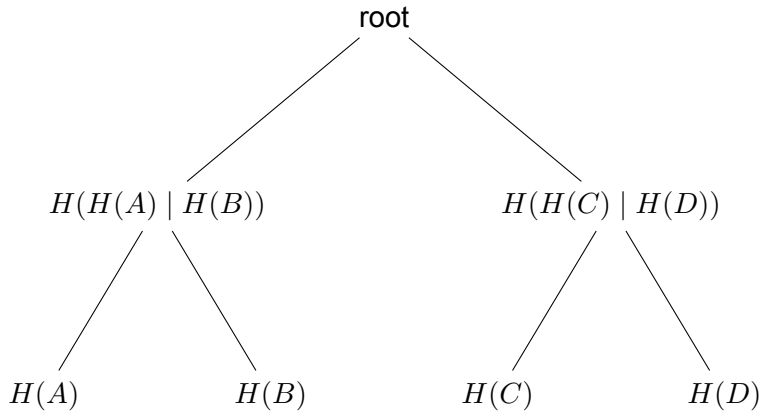
Τα δέντρα Merkle είναι μια κρυπτογραφική δομή δεδομένων που χρησιμοποιείται ευρέως σε συστήματα blockchain και άλλες εφαρμογές που απαιτούν επαλήθευση της ακεραιότητας των δεδομένων. Το βασικό τους πλεονέκτημα είναι ότι επιτρέπουν σε κάποιον να αποδείξει ότι μια συγκεκριμένη τιμή ανήκει σε ένα σύνολο χωρίς να αποκαλύψει την τιμή αυτή. Αυτή η ιδιότητα είναι απαραίτητη για την αποτελεσματική και διακριτική επαλήθευση.

Δομικά, ένα δέντρο Merkle είναι ένα δυαδικό δέντρο όπου κάθε **leaf node** περιέχει το hash ενός στοιχείου δεδομένων και κάθε **parrent node** περιέχει το hash της παράθεσης των κόμβων παιδιών του. Αυτή η ιεραρχία κατακερματισμού συνεχίζεται μέχρι τη ρίζα, που ονομάζεται **Merkle root**, η οποία αντιπροσωπεύει με μοναδικό τρόπο ολόκληρο το σύνολο των δεδομένων. Η επαλήθευση ότι μια τιμή ανήκει στο σύνολο απαιτεί μόνο έναν λογαριθμικό αριθμό υπολογισμών τιμών κατακερματισμού, καθιστώντας τα δέντρα Merkle εξαιρετικά αποδοτικά [12].

Στο παρακάτω σχήμα, βλέπουμε ένα δέντρο Merkle με κόμβους φύλλων $H(A)$, $H(B)$, $H(C)$ και $H(D)$. Κάθε γονικός κόμβος περιέχει τον κατακερματισμό των παιδιών του, με αποκορύφωμα τη ρίζα Merkle, η οποία αντιπροσωπεύει τον κατακερματισμό ολόκληρου του συνόλου δεδομένων.

Ας υποθέσουμε ότι τα A, B, C και D αντιπροσωπεύουν μυστικά κλειδιά διαφορετικών χρηστών και ο χρήστης D θέλει να αποδείξει ότι ανήκουν στο δέντρο χωρίς να αποκαλύψει το μυστικό κλειδί D. Αυτό μπορεί να επιτευχθεί χρησιμοποιώντας μια απόδειξη μηδενικής γνώσης με βάση την ακόλουθη διαδικασία.

Το πρωτόκολλο παρέχει στον χρήστη D μόνο τους κατακερματισμούς που απαιτούνται για την ανακατασκευή της ρίζας, συγκεκριμένα $H(C)$ και $H(H(A) \mid H(B))$. Χρησιμοποιώντας αυτές τις τιμές μαζί με τον δικό του κατακερματισμό $H(D)$, ο χρήστης D υπολογίζει τον κατακερματισμό της συνένωσης $(H(C) \mid H(D))$. Στη συνέχεια, συνδυάζει αυτόν τον νέο υπολογισμένο



Σχήμα 2.2: Δομή δέντρου Merkle: Κάθε lead node περιέχει το hash ενός στοιχείου δεδομένων και κάθε parrent node περιέχει το hash των παιδιών του. Ο root node αντιπροσωπεύει το hash ολόκληρου του συνόλου δεδομένων.

κατακερματισμό με τον κατακερματισμό που παρέχεται από το πρωτόκολλο $H(H(A) \mid H(B))$ για να ανακατασκευάσει τη ρίζα:

$$H\big(H(H(A) \mid H(B)) \mid (H(C) \mid H(D))\big)$$

Εάν ο προκύπτων κατακερματισμός ταιριάζει με τη ρίζα Merkle, η απόδειξη είναι έγκυρη και οποιοσδήποτε μπορεί να επαληθεύσει ότι ο χρήστης D ανήκει στο δέντρο χωρίς να αποκαλύψει ποτέ το μυστικό D. Αυτή η διαδικασία δείχνει πώς τα δέντρα Merkle, σε συνδυασμό με τις αποδείξεις μηδενικής γνώσης, επιτρέπουν την αποτελεσματική επαλήθευση της ιδιότητας μέλους με διατήρηση της ιδιωτικότητας.

2.3 Midnight

Σε αυτό το υπό-κεφάλαιο θα εξετάσουμε τα βασικά χαρακτηριστικά του Midnight, δίνοντας έμφαση στον τρόπο λειτουργίας των smart contracts, στον τρόπο με τον οποίο διασφαλίζεται η ανωνυμία των χρηστών των αποκεντρωμένων εφαρμογών που αναπτύσσονται σε αυτό το blockchain, καθώς και στις μοναδικές δυνατότητες που προσφέρει για ασφαλείς και αυτοματοποιημένες συναλλαγές χωρίς την ανάγκη κεντρικών διαμεσολαβητών.

2.3.1 Lace Wallet

Το Midnight χρησιμοποιεί το **Lace Wallet** ως κύριο μέσο διεπαφής των χρηστών με τις αποκεντρωμένες εφαρμογές, αλλά και ως μέσο διεξαγωγής συναλλαγών μεταξύ τους. Κάθε wallet διαθέτει μια μοναδική διεύθυνση, ορατή στον έξω κόσμο, καθώς και ένα ζεύγος κλειδιών — το **public key** και το **private key** — που αναπαριστώνται ως δεκαεξαδικές συμβολοσειρές. Το **public key** λειτουργεί ως αναφορά για το ποιος χρήστης εκτέλεσε ένα συγκεκριμένο transaction, ενώ το **private key**, που παραμένει μυστικό ακόμα και στον ίδιο τον χρήστη, χρησιμοποιείται για την υπογραφή των συναλλαγών που πραγματοποιεί το wallet. [5].

2.3.2 Νομίσματα - Tokens

Τα περισσότερα blockchain χρησιμοποιούν ένα μοναδικό token (δηλαδή νόμισμα) ως καύσιμο και ανταμοιβή, γεγονός που συχνά εκθέτει τα δεδομένα των χρηστών. Το Midnight καινοτομεί διαχωρίζοντας αυτές τις λειτουργίες σε δύο tokens[7] :

- **NIGHT**: Ένα «φανερό» (unshielded) token που χρησιμοποιείται για τη διακυβέρνηση και τις ανταμοιβές των παραγωγών των blocks.[7]
- **DUST**: Ένα προστατευμένο (shielded) token που λειτουργεί ως το «καύσιμο» της αλυσίδας, επιτρέποντας στους χρήστες να πραγματοποιούν συναλλαγές με πλήρη ιδιωτικότητα στα δεδομένα τους Όλες οι συναλλαγές που πραγματοποιούν οι χρήστες μέσω του lace wallet είναι σε DUST.[7]

2.3.3 Proof Server

Οποιοσδήποτε επιθυμεί να χρησιμοποιήσει μια αποκεντρωμένη εφαρμογή βασισμένη στο Midnight πρέπει να εγκαταστήσει και να τρέξει τοπικά στον υπολογιστή του τον proof server, ο οποίος είναι ενθυλακωμένος μέσα σε ένα docker container. Ο proof server παράγει τις αποδείξεις μηδενικής γνώσης για κάθε transaction ενός smart contract. Οι αποδείξεις αυτές αποτελούν το εχέγγυο ότι ο χρήστης μιας αποκεντρωμένης εφαρμογής, και κατά συνέπεια του αντίστοιχου smart contract, έχει ακολουθήσει όλους τους κανόνες που ορίζει κάθε transaction. Στη συνέχεια, η απόδειξη προωθείται στο δίκτυο ώστε να επαληθευθεί η ορθότητά της [5].

Αξίζει να τονιστεί ότι ο proof server δεν συνδέεται ποτέ στο διαδίκτυο ούτε αποστέλλει δεδομένα σε τρίτους αποδέκτες. Λειτουργεί αποκλειστικά τοπικά για κάθε χρήστη και παράγει αποδείξεις μηδενικής γνώσης με πλήρη ασφάλεια.

2.3.4 Το μοντέλο Kachina

Το Kachina είναι ένα μοντέλο δημιουργίας smart contracts που χρησιμοποιείται στο Midnight και έχει ως κύριο χαρακτηριστικό το διαχωρισμό μεταξύ της δημόσιας κατάστασης, δηλαδή των δεδομένων που υπάρχουν on-chain και της ιδιωτικής κατάστασης, η οποία βρίσκεται στον προσωπικό υπολογιστή κάθε χρήστη [11]. Αυτές ονομάζονται ιδιωτική κατάσταση και δημόσια κατάσταση. Η ιδιωτική κατάσταση ορίζεται εξ ολοκλήρου από τον κάθε χρήστη και πρέπει να ληφθεί υπόψιν ότι ένας κακόβουλος χρήστης μπορεί να την ορίσει όπως αυτός θέλει[11]. Η γενική ιδέα λειτουργίας του μοντέλου είναι η εξής:

- Κάθε smart contract έχει μία public state σ και για κάθε χρήστη p μία private state ρ_p. Όπως είπαμε η ρ_p δεν βρίσκεται στο ledger αλλά στον τοπικό υπολογιστή κάθε χρήστη.
- Όταν ο χρήστης εκκινεί ένα transaction, στην πραγματικότητα υπολογίζει την μεταβολή από την κατάσταση (σ, ρ_p) σε μια νέα κατάσταση (σ', ρ'_p). Κατασκευάζει μια απόδειξη μηδενικής γνώσης η οποία δηλώνει ότι η μεταβολή από σ σε σ' είναι έγκυρη, δηλαδή πως υπάρχει μια ιδιωτική κατάσταση ρ'_p και μία είσοδος για τα οποία η μεταβολή είναι έγκυρη. Τοπικά, ενημερώνει την ιδιωτική κατάσταση ως ρ'_p.

Με πιο απλά λόγια, η απόδειξη που κατασκευάζει ο χρήστης δηλώνει πως "Κατέχω μια έγκυρη ιδιωτική κατάσταση και κάποια είσοδο που μου επιτρέπουν να κάνω αλλαγές στην δημόσια κατάσταση του contract, υπακόυοντας στους κανόνες που αυτό ορίζει, δίχως να αποκαλύψω αυτή την ιδιωτική κατάσταση και την είσοδο". Η απόδειξη μετουσιώνεται σε transaction και μόλις αυτή καταχωρηθεί, το δίκτυιο (δηλαδή οι κόμβοι/validators) εξετάζουν την εγκυρότητα της απόδειξης. Αν είναι ορθή η απόδειξη, τότε σημαίνει πως η νέα δημόσια κατάσταση που ορίζει ο χρήστης είναι έγκυρη σε συνδυασμό με την ιδιωτική του κατάσταση και το δίκτυο προβαίνει σε ενημέρωση της δημόσιας κατάστασης. Μετά την αλλαγή της δημόσιας κατάστασης, ο χρήστης ενημερώνει και την ιδιωτική του κατάσταση[5] [11].

2.3.5 Smart Contracts στο Midnight

Το Midnight χρησιμοποιεί την Compact ως γλώσσα προγραμματισμού για ανάπτυξη smart contracts, η οποία έχει αναπτυχθεί από την ομάδα του Midnight και καίριο χαρακτηριστικό της

είναι η δυνατότητα διαχωρισμού των δημόσιων δεδομένων και της ιδιωτικής κατάστασης ενός χρήστη (public και private state) [7]. Θα κάνουμε μια σύντομη περιήγηση στα βασικά στοιχεία της compact μέσω ενός απλού smart contract. Εκτενέστερη ανάλυση θα πραγματοποιηθεί κατά την παρουσίαση του smart contract που αναπτύχθηκε για τη δική μας εφαρμογή. Παρουσιάζουμε παρακάτω ένα απλό smart contract το οποίο ορίζει μια public μεταβλητή και επιτρέπει μόνο στον δημιουργό του contract να την αλλάξει [5].

```
import CompactStandardLibrary;
witness secretKey(): Bytes<32>;
export ledger organizer: Bytes<32>;
export ledger restrictedCounter: Counter;
constructor() {
  organizer = publicKey(secretKey());
}

export circuit increment(): [] {
  assert(organizer == publicKey(secretKey()), "not authorized");
  restrictedCounter.increment(1);
}

circuit publicKey(sk: Bytes<32>): Bytes<32> {
  return persistentHash<Vector<2, Bytes<32>>>([pad(32, "some-domain-
      seperator"), sk]);
}
```

Σχήμα 2.3: Ένα απλό smart contract γραμμένο σε compact

Το ledger και η public state

Όσες μεταβλητές έχουν μπροστά τη λέξει-κλειδί **ledger** αποτελούν την public state του smart contract και είναι ορατές σε όλο το blockchain. Στο παράδειγμά μας, η μεταβλητή organizer είναι public και αποτελεί την διεύθυνση του δημιουργού του contract μέσω μιας δεκαεξαδικής συμβολοσειράς. Επίσης η μεταβλητή resrictedCounter είναι δημόσια και είναι ένας απλός μετρητής.

Ο witness και η private state

Όσες μεταβλητές έχουν μπροστά τη λέξη κλειδί **witness** αποτελούν την private state του smart contract και ζουν αποκλειστικά στον υπολογιστή του χρήστη. Είναι σημαντικό να τονιστεί ότι η πραγματική τιμή τους δεν είναι ποτέ ορατή στο smart contract, απλά όσες μεταβλητές έχουν τη λέξη witness μπροστά αποτελούν μια διεπαφή στην private state του χρήστη [5]. Στο παράδειγμά μας η μεταβλητή secretKey αποτελεί το μυστικό κλειδί του χρήστη και η πραγματική της τιμή δεν αποκαλύπτεται ποτέ δημόσια, ούτε στο contract. Μόνο ο χρήστης γνωρίζει την τιμή της και είναι στη δική του δικαιοδοσία να ορίσει ποια θα είναι αυτή. Δηλαδή μπορεί διαφορετικοί χρήστες μιας αποκεντρωμένης εφαρμογής να ορίσουν ο καθένας τον δικό του witness, στη δική μας περίπτωση το δικό του secretKey.

Τα circuits

Τα circuits μπορούν να παρομοιαστούν περίπου σαν τις συναρτήσεις σε μια γλώσσα προγραμματισμού σαν την C. Μπορούν να επιστρέφουν ή όχι τιμές, ανάλογα με αυτό που ορίζει ο προγραμματιστής. Χωρίζονται σε δύο κατηγορίες:

- **pure circuits**: δεν αλλάζουν ούτε χρησιμοποιούν την private ή την public state (δηλαδή τις ledger μεταβλητές και τους witnesses),

- **impure circuits**: μπορούν να τροποποιούν και να χρησιμοποιούν την public και την private state. Τα impure circuits κατά κανόνα αποτελούν τις ενέργειες που μπορεί να κάνει ένας χρήστης στο smart contract, δηλαδή είναι τα entry points στο smart contract καθώς και τα transactions που μπορεί να εκτελέσει ένας χρήστης. Κάθε impure circuit μεταγλωττίζεται σε γλώσσα Typescript και μπορεί να χρησιμοποιηθεί σαν μια συνάρτηση Typescript [5].

Στο παράδειγμά μας pure circuit είναι το publicKey το οποίο λειτουργεί απλά σαν hash function αξιοποιώντας την συνάρτηση persistentHash της βιβλιοθήκης. Αυτό το cirtcuit χρησιμοποιείται για να υπολογιστεί το hash του secretKey.
Το impure circuits είναι το increment το οποίο προσπαθεί να αυξήσει την τιμή της ledger μεταβλητής counter. Ωστόσο μέσα σε αυτο το circuit παρατηρούμε ότι χρησιμοποείται η έκφραση **assert**. Κάθε τέτοια έκφραση που είναι σε ένα circuit αποτελεί τους κανόνες που ορίζει το smart contract και διέπουν τη λειτουργία του [5]. Το συγκεκριμένο assert επιβάλλει ότι μόνο ο organizer του smart contract, δηλαδή αυτός που γνωρίζει ένα secretKey τέτοιο ώστε το hash αυτού του secretKey να ισούται με τη μεταβλητή organizer, έχει δικαίωμα να αυξήσει τον restrictedCounter. Διαφορετικά, αν ένας χρήστης που δεν γνωρίζει το έγκυρο secretKey προσπαθήσει να εκτελέσει ένα transaction με το increment circuit, αυτό θα αποτύχει και θα εμφανιστεί το μήνυμα λάθους "not authorized".
Σε αυτό το σημείο πρέπει να σημειώσουμε ότι μόλις ένα contract μεταγλωττιστεί, τότε για κάθε impure circuit που περιλαμβάνεται στο contract δημιουργούνται τα εξής αρχεία:

- **.zkir αρχεία**: Αποτελούν την ενδιάμεση αναπαράσταση του circuit σε μηδενική γνώση και ουσιαστικά αναπαριστούν τους κανόνες που θέτει το συγκεκριμένο circuit. Για κάθε τέτοιο αρχείο, υπάρχει και η δυαδική το αναπαράσταση σε ένα .bzkir αρχείο
- **.prover αρχεία**: Είναι τα proving keys στο πλαίσιο ενός πρωτόκολλου αποδείξεων μηδενικής γνώσης.
- **.verifier αρχεία**: Είναι τα verifying keys στο πλαίσιο ενός πρωτόκολλου αποδείξεων μηδενικής γνώσης. [5]

Στο δικό μας απλό smart contract θα είχαμε τα αρχεία increment.zkir, increment.bzkir, increment.prover, increment.verifier.
Κατά την δημιουργία ενός transaction μέσω του increment circuit ο χρήστης, χρησιμοποιώντας .prover κλειδί και το .zkir αρχείο κατασκευάζει μέσω του proof server μια απόδειξη μηδενικής γνώσης. Αυτή η απόδειξη, μαζί με το .verifier κλειδί διαχέονται στο δίκτυο και αναλόγως με την ορθότητα της απόδειξης, το transction γίνεται αποδεκτό και πραγματοποιείται αλλαγή στην public state του contract ή απορρίπτεται.

Κεφάλαιο 3

Μεθοδολογία και Υλοποίηση

Σε αυτή την ενότητα θα περιγράψουμε την μεθοδολογία που ακολουθήθηκε για το σχεδιασμό της αποκεντρωμένης εφαρμογής. Θα ξεκινήσουμε με μια γενική περιγραφή της εφαρμογής και παρουσιάζοντας τον τρόπο με τον οποίο τα επιμέρους κομμάτια που την αποτελούν επικοινωνούν και αλληλεπιδρούν μεταξύ τους. Στη συνέχεια θα γίνει εκτενής αναφορά στο smart contract που αναπτύχθηκε, αναλύοντας λεπτομερώς όλα τα στοιχεία του και εξηγώντας πως μέσω αυτού διασφαλίζονται τα επιθυμητά χαρακτηριστικά ενός ασφαλούς e-voting συστήματος, όπως είναι η ανωνυμία και η μοναδικότητα στην ψήφο. Επίσης, θα πραγματοποιηθεί γενική επισκόπηση του κώδικα που χρησιμοποιήθηκε και το πώς από την compact δημιουργείται κώδικας Typescript ο οποίος στη συνέχεια αξιοποιείτα για την ανάπτυξη της εφαρμογής. Τέλος, θα παρουσιαστεί ο κώδικας του server του πανεπιστημίου, δηλαδή πως αυθεντικοποιούνται οι φοιτητές και πως δημιουργείται το smart contract.

3.1 Γενική επισκόπηση της αποκεντρωμένης εφαρμογής

3.1.1 Stakeholders

Η συγκεκριμένη εφαρμογή περιλαμβάνει τους εξής εμπλεκόμενους:

- **Φοιτητές ως ψηφοφόρους**: Όλοι οι φοιτητές που είναι εγγεγραμμένοι στο πανεπιστήμιο έχουν δικαίωμα ψηφοφορίας σε όλες τις διαθέσιμες ψηφοφορίες της εφαρμογής.
- **Φοιτητές ως διοργανωτές**: Όλοι οι φοιτητές που είναι εγγεγραμμένοι στο πανεπιστήμιο έχουν δικαίωμα να δημιοργήσουν οι ίδιοι ψηφοφορίες καθορίζοντας τις ερωτήσεις και τις διαθέσιμες επιλογές.
- **Το πανεπιστήμιο**: Το πανεπιστήμιο λειτουργεί ως μια έμπιστη κεντρική αρχή και το μόνο που έχει δικαιοδοσία να κάνει ως εμπλεκόμενος είναι να δημιουργήσει το smart contract εξασφαλίζοντας ότι έγκυροι ψηφοφόροι θα είναι μόνο όλοι οι εγγεγραμένοι φοιτητές του πανεπιστημίου. Άπαξ και δημιουργηθεί και δημοσιευθεί στο δίκτυο το smart contract, το πανεπιστήμιο δεν μπορεί να συμμετέχει στην εφαρμογή, δηλαδή ούτε να ψηφίσει ούτε να δημιουργήσει ψηφοφορίες.

3.1.2 Σχεδιασμός και συστατικά του συστήματος

Το σύστημά μας αποτελείται από τα εξής συστατικά:

- **Smart Contract**: Λειτουργεί ως μια μικρή βάση δεδομένων και αποθηκεύει on-chain σε δομές δεδομένων όλα τα χαρακτηριστικά των ψηφοφοριών, δηλαδή τις ερωτήσεις μιας ψηφοφορίας, τις διαθέσιμες επιλογές, τους επιτρεπτούς ψηφοφόρους, τα αποτελέσματα

και τον κάθε οργανωτή μιας ψηφοφορίας. Ταυτόχρονα, όπως θα αναλυθεί παρακάτω, το contract περιλαμβάνει όλους τους κανόνες που πρέπει να διέπουν τη λειτουργία της εφαρμογής, εξασφαλίζοντας μεταξύ άλλων ότι μόνο εγγεγραμένοι φοιτητές μπορούν να ψηφίσουν, ότι οι ψήφοι παραμένουν ανώνυμοι και ότι κανείς δε μπορεί να διπλο-ψηφίσει. Τα παραπάνω πραγματοποιούνται με τη χρήση τεχνικών από την επιστήμη της κρυπτογραφίας και με την ενσωμάτωση αποδείξεων μηδενικής γνώσης.

- **University Server**: Περιλαμβάνει μια βάση δεδομένων με τα στοιχεία των φοιτητών (για απλούστευση μόνο όνομα χρήστη, κωδικό καθώς και το public key κάθε φοιτητή, δηλαδή το hash του secret key του) καθώς και έναν backend server ο οποίος χρησιμοποιείται για την επαλήθευση ότι ένας φοιτητής ανήκει στο πανεπιστήμιο. Ο server παρέχει ένα api μέσω του οποίου οι χρήστες αυθεντικοποιούνται όπως θα δούμε στη συνέχεια. Επίσης, περιλαμβάνει την λειτουργία έναρξης της ψηφοφορίας καθώς το πανεπιστήμιο δημοσιεύει το smart contract στο δίκτυο του Midnight έχοντας εξασφαλίσει εκ των προτέρων ότι οι ικανοί ψηφοφόροι θα είναι μόνο οι εγγεγραμένοι στο πανεπιστήμιο. Σημειώνουμε εδώ ότι η βάση δεδομένων είναι βάση *MongoDB* και ο κώδικας για το backend του πανεπιστημίου γράφτηκε σε *NodeJS*

- **Lace Wallet**: Το lace wallet αποτελεί το μέσο επικοινωνίας κάθε χρήστη με το δίκτυο. Μέσω αυτού πραγματοποιούνται όλα τα transactions και υπογράφονται από κάθε χρήστη. Το lace wallet αλληλεπιδρά με τον proof server στέλνοντας τις αποδείξεις μηδενικής γνώσης στο δίκτυο προς επαλήθευση.

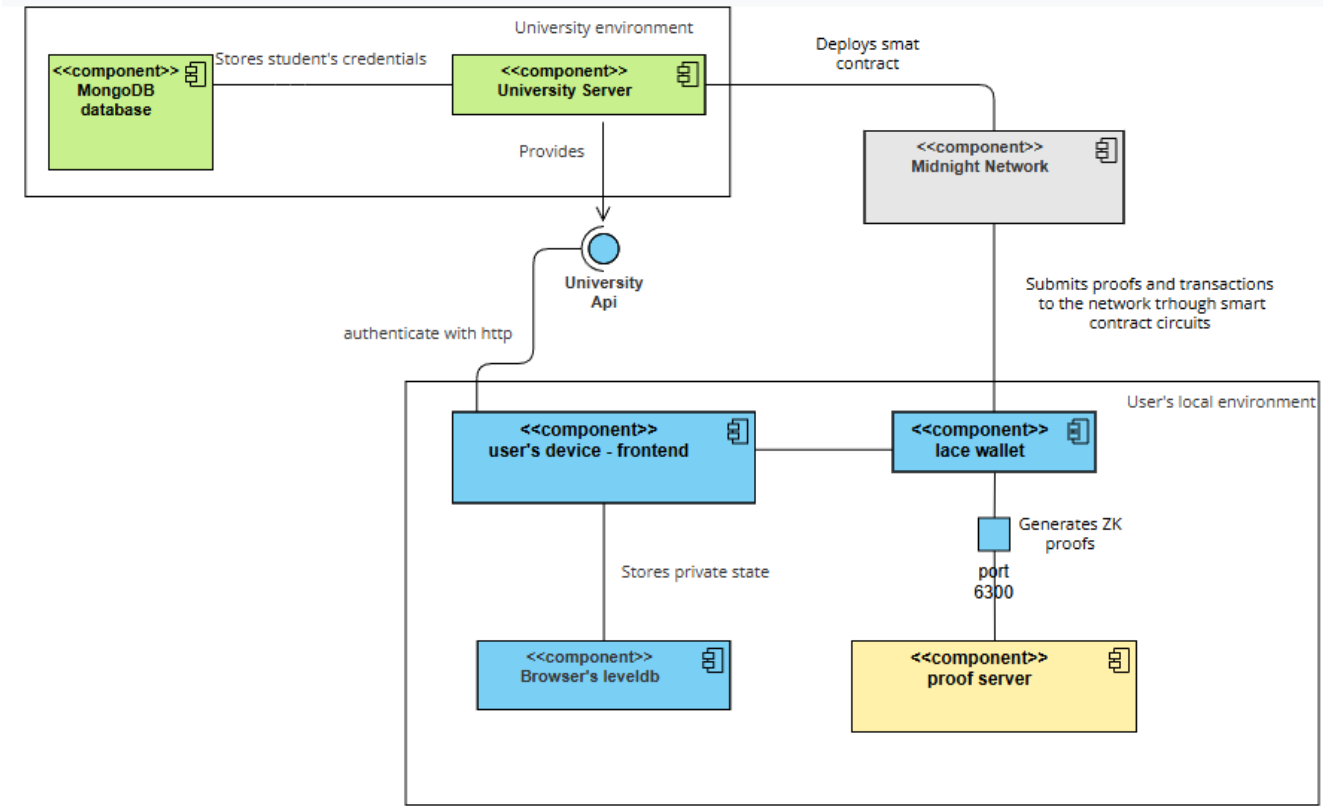
- **Frontend**: Το frontend αποτελεί τη διεπαφή χρηστών μέσω της οποίας οι φοιτητές μπορούν να χρησιμοποιήσουν την εφαρμογή αλληλεπιδρώντας με το smart contract (χρησιμοποιώντας ως μέσο επικοινωνίας το lace wallet). Πρακτικά, κάθε transaction του smart contract είναι και μια περίπτωση χρήσης η οποία πραγματοποιείται μέσω κώδικα *Typescript* και *ReactJS*.

- **Proof Server**: Όπως αναφέρθηκε και προηγουμένως, κάθε χρήστης της αποκεντρωμένης εφαρμογής πρέπει να τρέχει στον τοπικό του πολογιστή τον *Midnight Proof Server* έτσι ώστε για κάθε transaction που επιθυμεί να πραγματοποιήσει, να δημιουργεί μια απόδειξη μηδενικής γνώσης η αποία αποστέλνεται για επαλήθευση στο δίκτυο μέσω του lace wallet. Σημειώνουμε ότι και το πανεπιστήμιο, κατά το transaction δημοσίευσης του contract, πρέπει να τρέχει τον δικό του Proof Server.

- **LevelDB**: Η LevelDB αποτελεί μία γρήγορη βάση δεδομένων η οποία βασίζεται σε key-value ζευγάρια [3]. Το Midnight χρησιμοποιεί την LevelDB ως μέσο αποθήκευσης της ιδιωτικής κατάστασης κάθε χρήστη στον browser. Το ίδιο συμβαίνει και στη δική μας εφαρμογή.

- **Midnight Network**: Η εφαμοργή λειτουργεί στο TestNet του Midnight, μέσω του οποίου πραγματοποιούνται όλα τα transactions και υπάρχει αποθηκεύμενη πολλαπλές φορές όλη η δημόσια πληροφορία που απαρτίζει την εφαρμογή.

Παρακάτω παρουσιάζουμε και το component diagram για μια γενική επισκόπηση του συστήματος:



Σχήμα 3.1: Το component διάγραμμα του συστήματος

Σημειώνουμε ότι η βάση δεδομένων μαζί με τον server αποτελούν το περιβάλλον του πανεπιστημίου, ενώ το frontend, το lace wallet, η βάση leveldb και ο proof server αποτελούν το τοπικό περιβάλλον κάθε χρήστη.

3.1.3 Περιγραφή της λογικής και των κανόνων που διέπουν τη χρήση της εφαρμογής

Θα περιγράψουμε σύντομα την λογική που διέπει την εφαρμογή καθώς και τις ενέργειες που μπορεί να εκτελέσει ένας χρήστης. Θα δούμε στη συνέχεια λεπτομερώς πως αυτή η λογική και οι κανόνες αποτυπώνονται μέσω του smart contract

- Οι φοιτητές είναι καταχωρημένοι στη βάση των δεδομένων του πανεπιστημίου με το username και το password τους. Κάθε φοιτητής για να αυθεντικοποιηθεί, παράγει τοπικά ένα τυχαίο secret key το hash του οποίου (public key) αποστέλλει στο πανεπιστήμιο μαζί με τα username, password. Αν ο φοιτητής υπάρχει στη βάση τότε το public key καταχωρείται στη βάση

- Όταν αυθεντικοποιηθούν οι φοιτητές το πανεπιστήμιο δημοσιεύει το smart contract στο δίκτυο και οι φοιτητές μπορούν να χρησιμοποιήσουν την εφαρμογή.

- Κάθε φοιτητής μπορεί να δημιουργήσει όσες ψηφοφορίες θέλει και να επιλέξει το θέμα (ερώτηση) της ψηφοφορίας καθώς και τις διαθέσιμες επιλογές της. Καθορίζει επίσης κατά τη δημιουργία δύο χρονικά deadline, ένα για την υποβολή ψήφου και ένα για τη δημοσίευση ψήφου, καθορίζοντας έτσι δύο περιόδους: την περίοδο υποβολής ψήφων και την περίοδο δημοσίευσης ψήφων. Το deadline δημοσίευσης ψήφου πρέπει να είναι μεταγενέστερο από το deadline υποβολής ψήφου. Κάθε ψηφοφορία ξενικάει σε κατάσταση

CLOSED. Όταν ο διοργανωτής θέσει την ερώτηση και τις διαθέσιμες επιλογές μπορεί να την μετατρέψει σε OPEN. Άπαξ και η ψηφοφορία μεταβεί σε κατάσταση OPEN δεν μπορεί να αλλάξει η ερώτηση ή οι επιλογές. Προφανώς μόνο ο διοργανωτής μιας ψηφοφορίας μπορεί να θέσει την ερώτηση και τις επιλογές.

- Οι χρήστες μπορούν να ψηφίσουν μόνο όταν η ψηφοφορία είναι OPEN ενώ απαγορεύεται το double-voting. Όταν ο χρήστης υποβάλει την ψήφο του, στα αλήθεια υποβάλει το hash της ψήφου του και σε αυτή τη φάση η ψήφος του παραμένει τοπικά. Σημαντικό είναι να τονιστεί ότι τα αποτελέσματα δεν είναι ορατά κατά την περίοδο υποβολής ψήφων, γίνονται έπειτα με τις δημοσιεύσεις ψήφων.

- Όταν παρέλθει η περίοδος υποβολής ψήφου έρχεται η περίοδος δημοσίευσης ψήφων. Όταν ο χρήστης δημοσιεύει την ψήφο του, στην ουσία την αποκαλύπτει δημόσια χωρίς ωστόσο να αποκαλύψει το public key του. Με αυτόν τον τρόπο αποφεύγεται η σύνδεση του public key με την ψήφο κάποιου φοιτητή. Κατά την περίοδο δημοσίευσης ψήφων γίνονται ορατά και τα αποτελέσματα της ψηφοφορίας. Θεωρούμε ότι αν ένας χρήστης δε δημοσιεύσει την ψήφο του εντός του deadline τότε αυτή δεν θα μετρήσει στα αποτελέσματα

3.2 Ανάλυση του Smart Contract

3.2.1 Οι μεταβλητές ledger

Το smart contract αποτελείται από την δήλωση των μεταβλητών οι οποίες θα είναι δημόσιες στο δίκτυο:

- **university_public_key**: Είναι το δημόσιο κλειδί του πανεπιστημίου, συγκεκριμένα το hash του secret key του πανεπιστημίου.

- **votings**: Είναι ένα σύνολο με τα ids όλων των ψηφοφοριών. Κάθε ψηφοφορία όταν δημιουργείται διακρίνεται από ένα μοναδικό δεαεξαδικό id.

- **voting_options**: Είναι ένα Map που περιέχει σαν κλειδί το id της ψηφοφορίας και σαν τιμή την διαθέσιμη επιλογή. Για τεχνικούς λόγους η διαθέσιμη επιλογή αναπαρίσταται και αυτή ως Map με κλειδί τον αριθμό της επιλογής και τιμή την ίδια την επιλογή.

- **voting_questions**: Είναι ένα Map για τις ερωτήσεις με κλειδί το id της ψηφοφορίας και τιμή την ερώτηση ή γενικότερα το θέμα της ψηφοφορίας.

- **voting_results**: Είναι ένα Map για το αποτέλεσμα κάθε ψηφοφορίας με κλειδί το id της και τιμή ένα δεύτερο Map. Το Map αυτό έχει κλειδί την επιλογή και ως τιμή έναν counter που αναπαριστά το πλήθος των φορών που επιλέχθηκε η συγκεκριμένη επιλογή.

- **eligible_voters**: Οι συμμετέχοντες στις ψηφοφορίες αναπαρίστανται ως ένα Merkle Tree που περιέχει τα δημόσια κλειδιά τους, δηλαδή τα hash των μυστικών κλειδιών τους. Η χρήση Merkle Tree όπως θα περιγραφεί στη συνέχεια είναι καθοριστική για τη διατήρηση της ανωνυμίας.

- **voting_states**: Είναι ένα Map με κλειδί το id της ψηφοφορίας και τιμή τύπου VOTE_STATE(OPEN ή CLOSED) που δηλώνει αν η ψηφοφορία είναι ανοικτή ή κλειστή. Σημειώνουμε εδώ ότι όλες οι ψηφοφορίες όταν δημιουργούνται αρχικοποιούνται ως CLOSED. Όσο η ψηφοφορία είναι CLOSED ο διοργανωτής μπορεί να προσθέσει την ερώτηση και οι επιλογές και οι ψηφοφόροι δε μπορούν να ψηφίσουν. Όταν ο διοργανωτής ανοίξει την ψηφοφορία και η κατάσταση γίνει OPEN , δε μπορεί πλέον να αλλάξει την ερώτηση ή να προσέσει επιλογές.

- **voting_nulifiers**: Είναι ένα Map με κλειδί το id της ψηφοφορίας και τιμή το σύνολο από μοναδικές δεκαεξαδικές τιμές που δρουν σαν tokens προς κάθε ψηφοφόρο προς αποφυγή διπλο-ψηφίσματος.
- **publish_voting_nulifiers**: Είναι ένα Map με κλειδί το id της ψηφοφορίας και τιμή το σύνολο από μοναδικές δεκαεξαδικές τιμές που δρουν σαν tokens προς κάθε ψηφοφόρο ώστε να μην μπορεί ο κάθε ψηφοφόρος να δημοσιεύεσει 2 φορές την ψήφο του. Θα μιλήσουμε εκτενέστερα στη συνέχεια για τη δημοσίευση ψήφου.
- **hashed_votes**: Είναι ένα Map όπου κλειδί είναι το id της ψηφοφορίας και τιμή το σύνολο που περιέχει όλες τις ψήφους της ψηφοφορίας, αλλά κατακερματισμένες από hash function.
- **voting_organizers**: Είναι ένα Map όπου κλειδί είναι το id της ψηφοφορίας και τιμή το δημόσιο κλειδί (δηλαδή το hash του ιδιωτικού κλειδιού) του διοργανωτή της συγκεκριμένης ψηφοφορίας.
- **cast_vote_expiration_time**: Είναι ένα Map όπου κλειδί είναι το id της ψηφοφορίας και τιμή η τελική ημερομηνία μέχρι την οποία ο χρήστης μπορεί να υποβάλλει την ψήφο του. Η τελική ημερομηνία είναι ακέραιος και μετριέται σε Unix epoch.
- **publish_vote_expiration_time**: Είναι ένα Map όπου κλειδί είναι το id της ψηφοφορίας και τιμή η τελική ημερομηνία μέχρι την οποία ο χρήστης μπορεί να δημοσιεύσει την ψήφο του. Η τελική ημερομηνία είναι ακέραιος και μετριέται σε Unix epoch.

```
export ledger votings: Set<Bytes<32>>;
export ledger voting_options: Map<Bytes<32>, Set<Bytes<32>>>;
export ledger voting_questions: Map<Bytes<32>, Opaque<"string">>;
export ledger voting_results: Map<Bytes<32>, Map<Bytes<32>, Counter>>;
export ledger eligible_voters: HistoricMerkleTree<5, Bytes<32>>;
export ledger voting_states: Map<Bytes<32>, VOTE_STATE>;
export ledger voting_nulifiers: Map<Bytes<32>, Set<Bytes<32>>>;
export ledger publish_voting_nulifiers: Map<Bytes<32>, Set<Bytes<32>>>;
export ledger voting_organizers: Map<Bytes<32>, Bytes<32>>;
export ledger hashed_votes: Map<Bytes<32>, Set<Bytes<32>>>;
export ledger publish_vote_expiration_time: Map<Bytes<32>, Uint<64>>;
export ledger cast_vote_expiration_time: Map<Bytes<32>, Uint<64>>;
```

Σχήμα 3.2: Οι ledger μεταβλητές του smart contract

3.2.2 Ο κατασκευαστής

Το smart contract έχει έναν κατασκευαστή ο οποίος αρχικοποιεί τις ledger μεταβλητές στις default τιμές και θέτει το university_public_key να είναι ίσο με το hash του secret key του τρέχοντος χρήστη ο οποίος θα είναι το πανεπιστήμιο. Έτσι, αρχικοποιείται το public key του πανεπιστημίου. Τέλος, ο κατασκευαστής λαμβάνοντας ως είσοδο ένα Vector με τα public keys των έγκυρων ψηφοφόρων, πραγματοποιεί την είσοδο αυτών στο Merkle Tree των έγκυρων ψηφοφόρων.

```
constructor(eligible_voter_public_keys: Vector<5, Bytes<32>>) {
    university_public_key = disclose(public_key(local_secret_key()));
    votings.resetToDefault();
    voting_options.resetToDefault();
    voting_results.resetToDefault();
    voting_states.resetToDefault();
```

```
    voting_nulifiers.resetToDefault();
    voting_organizers.resetToDefault();
    voting_questions.resetToDefault();
    hashed_votes.resetToDefault();
    publish_voting_nulifiers.resetToDefault();
    publish_vote_expiration_time.resetToDefault();
    cast_vote_expiration_time.resetToDefault();
    count.increment(1);

    for (const v of disclose(eligible_voter_public_keys)) {
        eligible_voters.insert(v);
    }
}
```

Σχήμα 3.3: Ο κατασκευαστής του smart contract

3.2.3 Οι witness μεταβλητές και η private state

Όπως έχουμε αναφέρει οι witness μεταβλητές αποτελούν την ιδιωτική κατάσταση κάθε χρήστη και γι αυτό το λόγο υλοποιούνται από τους διαφορετικούς χρήστες της εφαρμογής. Το αποτέλεσμα που θα δώσουν δε θα πρέπει να θεωρείται εμπιστεύστιμο. Στη δική μας υλοποίηση, για την ιδιωτική κατάσταση χρειαζόμαστε ένα αντικείμενο τύπου **VoteGuardianPrivateState** και μια συνάρτηση που θα κατασκευάζει την ιδιωτική κατάσταση, δηλαδή αντικείμενα του παραπάνω τύπου, την **createVoteGuardianPrivateState**.

```
export type VoteGuardianPrivateState = {
  readonly secretKey: Uint8Array;
  voterPublicKeyPath: MerkleTreePath<Uint8Array>;
  votesPerVotingMap: Map<String, String>;
};

export const createVoteGuardianPrivateState = (
  secretKey: Uint8Array,
  voterPublicKeyPath: MerkleTreePath<Uint8Array>,
  votesPerVotingMap: Map<String, String>,
) => ({
  secretKey,
  voterPublicKeyPath,
  votesPerVotingMap,
});
```

Σχήμα 3.4: Η private state του smart contract

Όπως έχουμε αναφέρει οι μεταβλητές witness ζουν αποκλειστικά τοπικά για κάθε χρήστη και δε δημοσιεύονται ποτέ στο δίκτυο ενώ ταυτόχρονα αποτελούν την μυστική πληροφορία που κατέχει ο prover σε ένα πρωτόκολλο μηδενικής γνώσης η οποία είναι σημαντική για την ορθότητα της απόδειξης. Στην εφαρμογή μας διακρίνονται τρεις τέτοιες μεταβλητές:

- **local_secret_key**: Αποτελεί το secret key κάθε συμμετέχοντος ως μια δεκαεξαδική συμβολοσειρά. Δεν παίρνει κάποιο όρισμα, απλά επιστρέφει την ίδια private state και το ίδιο secret key, καθώς δεν αλλάζει την ιδιωτική κατάσταση.
  Τι θα γινόταν αν ένας κακόβουλος χρήστης έκανε τη δική του υλοποίηση? Πρακτικά αν ένας χρήστης όριζε για παράδειγμα η local_secret_key να επιστρέφει κάθε φορά ένα τυχαίο δεκαεξαδικό, τότε θα ήταν σχεδόν απίθανο να μαντέχει ένα secret key που να ανήκει σε αυτά των έγκυρων χρηστών οπότε δε θα μπορούσε να συμμετέχει στις ψηφορορίες.

Για να υπήρχε πρόβλημα ασφάλειας, θα έπρεπε το secret key που θα επέστρεφε να ήταν ένα εκ των έγκυρων, κάτι το οποίο είναι υπολογιστικά αδύνατο.

- **find_voter_public_key**: Παίρνει ως όρισμα το δημόσιο κλειδί ενός χρήστη και η τιμή επιστροφής του είναι τύπου **MerkleTreePath** το οποίο περιέχει μια λίστα με όλα τα hashes που χρειάζεται να έχει ο κάτοχος του τρέχοντος φύλλου στο δέντρο (δηλαδή του τρέχοντος δημόσιου κλειδιού) ώστε να ανακατασκευάσει τη ρίζα. Ουσιαστικά αυτός ο witness αποτελεί τη γνώση που κατέχει ο έγκυρος ψηφοφόρος ότι γνωρίζει ένα έγκυρο μονοπάτι μέσα στο Merkle tree. Όπως θα δούμε στη συνέχεια, η απόδειξη πως ένας χρήστης είναι έγκυρος βασίζεται στην κλήση της συνάρτησης από τη βιβλιοθήκη της compact **checkRoot** πάνω στο δένδρο των ψηφοφόρων με όρισμα το αποτέλεσμα που θα επιστρέψει ο witness.
  Στην υλοποίηση της εφαρμογής, η find_voter_public_key χρησιμοποεί την **findPathForLeaf** για το public key που λαμβάνει ως είσοδο και επιστρέφει το **MerkleTreePath** για αυτό το public key. Αν το public key που δεχθεί ως όρισμα δεν υπάρχει στο δέντρο τότε επιστρέφει **undefined**.

- **secret_vote**: Αποτελεί την μυστική ψήφο ενός ψηφοφόρου για ψηφοφορία με συγκεκριμένο id. Παίρνει ως όρισμα το id της ψηφοφορίας και επιστρέφει την ψήφο για τη συγκεκριμένη ψηφοφορία ψάχνοντας στο Map **votesPerVotingMap** το οποίο είναι αποθηκευμένο στην private state του ψηφοφόρου. Αν δεν υπάρχει ψήφος για ση συγκεκριμένη ψηφοφορία τότε θα επιστρέψει default τιμή, δηλαδή ένα Uint8Array με 32 μηδενικά. Η ουσία εδώ είναι η ψήφος να παραμένει μυστική για κάθε χρήστη και να δημοσιεύεται το hash αυτής. Στο τέλος της ψηφοφορίας ο χρήστης θα δημοσιεύει την ψήφο του. Στη συνέχεια θα γίνεται έλεγχος αν το hash της ψήφου που δημοσιεύει είναι ίδιο με το hash που υπάρχει αποθηκευμένο στο ledger. Αν ισχύει αυτό, σημαίνει ότι η ψήφος είναι σωστή.

Επίσης παρουσιάζουμε την υλοποίηση των witness function ως συναρτήσεις *Typescript* στο σχήμα 3.5.

```
local_secret_key: ({
    privateState,
    ledger,
  }: WitnessContext<Ledger, VoteGuardianPrivateState>): [
     VoteGuardianPrivateState, Uint8Array] => [
    privateState,
    privateState.secretKey,
  ],

  find_voter_public_key: (
    { privateState, ledger }: WitnessContext<Ledger,
       VoteGuardianPrivateState>,
    item: Uint8Array,
  ): [VoteGuardianPrivateState, MerkleTreePath<Uint8Array>] => [
    createVoteGuardianPrivateState(privateState.secretKey, ledger.
       eligible_voters.findPathForLeaf(item)!),
    ledger.eligible_voters.findPathForLeaf(item)!,
  ],

  secret_vote: (
    { privateState, ledger }: WitnessContext<Ledger,
       VoteGuardianPrivateState>,
    votingId: Uint8Array,
  ): [VoteGuardianPrivateState, Uint8Array] => [
    privateState,
    toBytes32FromString(privateState.votesPerVotingMap.get(toHex(votingId))
       ?.toString()),
```

```
  ],
```

Σχήμα 3.5: Οι συναρτήσεις witness

3.2.4 Τα pure circuits

Το smart contract περιλαμβάνει τα ακόλουθα pure circuits:

- **public_key**: Υπολογίζει το hash του secret key επιστρέφοντας μια δεκαεξαδική τιμή.
- **organizer**: Υπολογίζει το hash του secret key επιστρέφοντας μια δεκαεξαδική τιμή. Διαφοροποιείται από τα άλλα hashes μέσω του "organizer-domain".
- **nullifier**: Παίρνει ως είσοδο το secret key του χρήστη και το id της τρέχουσας ψηφοφορίας και με αυτό τον τρόπο δημιουργεί ένα μοναδικό hash για τον χρήστη για τη συγκεκριμένη ψηφοφορία. Διαφοροποιείται από τα άλλα hashes μέσω του "nulifier-domain".
- **generate_voting_id**: Χρησιμοποιείται για την απόδοση ενός μοναδικού id για μια ψηφοφορία. Παίρνει ως είσοδο το secret key του οργανωτή της ψηφοφορίας καθώς και την δημόσια μεταβλητή count η οποία αυξάνεται κατά 1 κατά τη δημιουργία κάθε ψηφοφορίας. Έτσι επιτυγχάνεται η μοναδικότητα στα ids κάθε ψηφοφορίας.
- **prove_eligibility**: Αυτό το circuit χρησιμοποείται για να ελέγχξει αν ο τρέχων χρήστης είναι έγκυρος επιστρέφοντας true ή false αντίστοιχα. Υπολογίζεται η μεταβλητή path που είναι το αποτέλεσμα της επιστροφής της find_voter_public_key για το public key του τρέχοντος χρήστη και πρέπει να αποτελεί ένα έγκυρο μονοπάτι μέσα στο δέντρο. Στη συνέχεια, με χρήση της συνάρτηρης checkRoot ελέγχεται το συγκεκριμένο path μπορεί να ανακατασκευαστεί η ρίζα. Αν ναι σημαίνει πως το path είναι έγκυρο και επιστρέφεται true, διαφορετικά false.
- **publish_nullifier**: Παίρνει ως είσοδο το secret key του χρήστη και το id της τρέχουσας ψηφοφορίας και με αυτό τον τρόπο δημιουργεί ένα μοναδικό hash για τον χρήστη για τη συγκεκριμένη ψηφοφορία. Διαφοροποιείται από τα άλλα hashes μέσω του "publish-nulifier-domain".
- **hash_secret_vote**: Παίρνει ως είσοδο την ψήφο, το id της ψηφοφορίας και το secret key του ψηφοφόρου και δημιουργεί ένα μοναδικό hash για την ψήφο, ανά ψηφοφορία και ανά χρήστη.

```
circuit public_key(sk: Bytes<32>): Bytes<32> {
    return persistentHash<Bytes<32>>(sk);
}

circuit organizer (sk: Bytes<32>): Bytes<32> {
    return persistentHash<Vector<2, Bytes<32>>>(
        [pad(32, "organizer-domain"), sk]
    );
}

circuit nullifier (sk: Bytes<32>, voting_id: Bytes<32>): Bytes<32> {
   return persistentHash<Vector<3, Bytes<32>>>(
           [pad(32, "nulifier-domain"), voting_id as Bytes<32>, sk]);
}

circuit publish_nullifier (sk: Bytes<32>, voting_id: Bytes<32>): Bytes<32>
   {
```

```
    return persistentHash<Vector<3, Bytes<32>>>(
            [pad(32, "publish-nulifier-domain"), voting_id as Bytes<32>, sk
               ]);
}

circuit generate_voting_id (count: Field, sk: Bytes<32>) : Bytes<32> {
    return persistentHash<Vector<3, Bytes<32>>>(
            [pad(32, "generate_id"), count as Bytes<32>, sk]);
}

circuit hash_secret_vote(
    secret_vote: Bytes<32>,
    voting_id: Bytes<32>,
    secret_key: Bytes<32>
): Bytes<32> {
    return persistentHash<Vector<4, Bytes<32>>>(
        [pad(32, "secret-vote-domain"), voting_id, secret_vote, secret_key]
    );
}

circuit prove_eligibility(): Boolean {
    const participant_public_key = public_key(local_secret_key());
    const path = find_voter_public_key(participant_public_key);

    return eligible_voters.checkRoot(disclose(merkleTreePathRoot<5, Bytes
       <32>>(path)));

}
```

Σχήμα 3.6: Τα pure circuits του smart contract

3.2.5 Τα impure circuits

Κάθε impure circuit που περιλαμβάνεται στο smart contract μπορεί να χρησιμοποιεί τις witness functions και να μεταβάλλει τις ledger μεταβλητές, αλλάζοντας την δημόσια κατάσταση του smart contract. Κάθε τέτοιο circuit μετά την μεταγλώττιση από Compact σε Typescript είναι διαθέσιμο να κληθεί και αποτελεί ένα transaction στο Midnight. Πριν ξεκινήσουμε την ανάλυση των circuits, θα εξηγήσουμε μια κοινή λογική που υπάρχει σε όλα τα circuits και κατά την ανάλυση θα αναφέρουμε ποια από τα παρακάτω ισχύουν σε κάθε circuit. Συγκεκριμένα:

1. Το πανεπιστήμιο δεν μπορεί να κάνει κανένα transaction, η "δουλειά" του τελειώνει μετά την δημοσίευση του contract. Γι' αυτό το λόγο, στην αρχή κάθε circuit ελέγχεται με τη δήλωση **assert** ότι το public key του πανεπιστημίου (το οποίο υπάρχει ως ledger μεταβλητή) πρέπει να διαφέρει από το public key του τρέχοντας χρήστη.

2. Κάθε άλλο transaction πρέπει να πραγματοποιείται μόνο από έγκυρους χρήστες, δηλαδή από τους φοιτητές. Γι' αυτό το λόγο στην αρχή των circuits που απαιτούν αυθεντικοποίηση πραγματοποιείται έλεγος με δήλωση *assert* ότι ο χρήστης ανήκει στο merkle tree, μέσω του **prove_eligibility** circuit.

3. Κάθε transaction που πραγματοποιείται από τον διοργανωτή της ψηφοφορίας απαιτεί έλεγχο ότι ο συγκεκριμένος χρήστης είναι ο διοργανωής, μέσω **assert** το συγκεκριμένο id της ψηφοφορίας "ανήκει" στο τρέχον hash του secret kty, δηλαδή στο hash του τρέχοντος secret key χρησιμοποιώντας για hashing το organizer circuit. Με άλλα λόγια ότι το Map voting_organizers περιέχει το ζεύγος voting_id, organizer_public_key.

Για όλους τους παραπάνω κανόνες, σε περίπτωση που δεν ισχύει κάποιος από αυτούς κατά την πραγματοποίηση ενός transaction μέσω ων circuits, τότε το transaction δεν ολοκληρώνεται και εμφανίζεται αντίστοιχο μήνυμα λάθους στον μη έγκυρο χρήστη που το εκτέλεσε. Συγκεκριμένα, το smart contract περιλαμβάνει τα ακόλουθα impure circuits:

- **create_voting**: Με αυτό το transaction δημιουργείται μια νέα ψηφοφορία. Ισχύουν τα (1) και (2) από παραπάνω. Η ψηφοφορία αποκτά ένα μοναδικό id και γίνεται είσοδος αυτού του id στη μεταβλητή votings, και σαν διοργανωτής ορίζεται το τρέχων hash(secret_key) κατακερματισμένο με το organizer circuit. Αυτό πραγματοποιείται με είσοδο στη μεταβλητή voting_organizers. Η ψηφοφορία αρχικοποιείται ως CLOSED και οι υπόλοιπες ledger μεταβλητές αρχικοποιούνται με default τιμές έχοντας ως κλειδί σε κάθε Map το id. Το circuit αυτό παίρνει ως παραμέτρους δύο ακέραιες τιμές, οι οποίες αντιστοιχούν στις ημερομηνίες λήξης για υποβολή ψήφου και για δημοσίευση ψήφου. Μετά το πέρας αυτών των ημερομηνιών οι χρήστες δε θα μπορούν να υποβάλλουν ή να δημοσιεύσουν την ψήφο τους. Το circuit περιέχει έλεγχο ότι η ημερομηνία λήξης δημοσίευσης πρέπει να είναι μεταγενέστερη της ημεροηνίας λήξης υποβολής, καθώς πρώτα οι χρήστες ψηφίζουν και μετά δημοσιεύουν την ψήφο.(θα μιλήσουμε εκτενέστερα στη συνέχεια για τη δημοσίευση ψήφου).

- **edit_question**: Με αυτό το transaction καθορίζεται η ερώτηση της συγκεκριμένης ψηφοφορίας. Ισχύουν τα (1), (2) και (3) από παραπάνω. Το circuit δέχεται ορίσματα το id της ψηφοφορίας και την ερώτηση και εισάγει στη ledger μεταβλητή voting_questions το ζεύγος αυτό. Επίσης μέσω assert ελέγχεται αν η κατάσταση της ψηφοφορίας είναι OPEN. Αν δεν είναι, εμφανίζεται κατάλληλο μήνυμα λάθους

- **add_option**: Με αυτό το transaction εισάγεται μια επιλογή για την ψηφοφορία. Ισχύουν τα (1), (2) και (3) από παραπάνω. Το circuit δέχεται ως όρισμα το id της ψηφοφορίας και την επιλογή. Γίνεται εισαγωγή στη ledger μεταβλητή voting_options το id της ψηφοφορίας σαν κλειδί και προστίθεται στο σύνολο η επιλογή Επίσης, αρχικοποιείται με τον default

μετρητή και η ledger μεταβλητή voting_results για τη συγκεκριμένη επιλογή. Επίσης μέσω assert ελέγχεται αν η κατάσταση της ψηφοφορίας είναι OPEN. Αν δεν είναι, εμφανίζεται κατάλληλο μήνυμα λάθους

- **open_voting**: Με αυτό το transaction ανοίγει η συγκεκριμένη ψηφοφορία, δηλαδή τίθεται η κατάστασή της ως OPEN. Ισχύουν τα (1), (2) και (3) από παραπάνω. Το transaction δέχεται ως όρισμα το id της ψηφοφορίας και ελέγχει μέσω *assert* ότι η ψηφοφορία είναι κλειστή, διαφορετικά εμφανίζει μήνυμα λάθους. Στη συνέχεια, γίνεται εισαγωγή στη ledger μεταβλητή voting_states το ζεύγος του id της ψηφοφορίας με την τιμή OPEN.
- **cast_vote**: Με αυτό το transaction ένας έγκυρος χρήστης μπορεί να ψηφίσει.
    1. Αρχικά ελέγχεται ότι η τρέχουσα ημερομηνία είναι προγενέστερη της ημερομηνίας λήξης υποβολής ψήφου για τη συγκεκριμένη ψηφοφορία, χρησιμοποιώντας τη συνάρτηση βιβλιοθήκης **blockTimeLte** μέσω δήλωσης assert. Αν δεν είναι τότε εμφανίζεται κατάλληλο μήνυμα λάθους.
    2. Ελέγχεται ότι η ψηφοφορία είναι ανοικτή μέσω σχετικής δήλωσης *assert*. Αν δεν είναι ανοικτή, εμφανίζεται σχετικό μήνυμα λάθους και το transaction εγκαταλείπεται.
    3. Πραγματοποιούνται οι έλεγχοι (1), (2) από παραπάνω και αν κάποιος αποτύχει δηλαδή αν ο χρήστης είναι το πανεπιστήμιο, ή αν ο χρήστης δεν είναι έγκυρος τότε το transaction εγκαταλείπεται με σχετικό μήνυμα λάθους. Σημειώνεται εδώ ότι όλοι οι έγκυροι χρήστες μπορούν να ψηφίσουν σε όλες τις ψηφοφορίες.
    4. Στη συνέχεια γίνεται ο έλεγχος για double-voting δηλαδή έλεγχος για το αν ο τρέχων χρήστης έχει ήδη ψηφίσει για αυτή την ψηφοφορία. Για να δούμε πως γίνεται αυτό υποθέτουμε ότι έχουμε φτάσει στο τέλος των ελέγχων όπου ο χρήστης έχει καταχωρίσει την ψήφο του. Μόλις γίνει αυτό, υπολογίζεται το **voting_nullifier** το οποίο είναι hash του secret key με διαφορετικό domain για να ξεχωρίζει από το public key. Το hash αυτό πραγματοποιείται μέσω του pure circuit nullifier και δέχεται επίσης ως όρισμα μαζί με το secret key και το **voting_id** ώστε να είναι μοναδικό ανά χρήστη ανά ψηφοφορία. Το **voting_nullifier** καταχωρείται στο ledger σύνολο **voting_nulifiers**. Όταν λοιπόν το circuit φτάσει στον έλεγχο για double-voting ελέγχεται αν το hash με τη nullifier του secret key του τρέχοντος χρήστη για τη συγκεκριμένη ψηφοφορία υπάρχει στο σύνολο των voting_nulifiers για το συγκεκριμένο voting_id. Αν ναι σημαίνει πως ο χρήστης έχει ήδη ψηφίσει οπότε εμφανίζεται μήνυμα λάθους και εγκαταλείπεται το transaction, διαφορετικά συνεχίζονται οι ενέργειες.
    5. Στη συνέχεια, ο typescript κώδικας της dapp φροντίζει ώστε να ενημερωθεί η private state του χρήστη με τη συγκεκριμένη ψήφο, έτσι ώστε η κλήση της witness μεταβλητής **secret_vote(voting_id)** να επιστρέψει την ψήφο του ψηφοφόρου για τη συγκεκριμένη ψηφοφορία. Υπενθυμίζουμε ότι παρότι ο κώδικας της dapp θα ενημερώσει την private state με την επιλεγμένη ψήφο, στην πραγματικότητα ο χρήστης μπορεί να τροποποιήσει τον κώδικα και να αναθέσει στην επιλεγμένη ψήφο του όποια τιμή θέλει. Ο έλεγχος για το αν η ψήφος είναι έγκυρη, θα πραγματοποιηθεί σε επόμενο transaction.
    6. Έχοντας οριστεί και το private state, το επόμενο βήμα του transaction είναι να υπολογιστεί το hash της επιλεγμένης ψήφου, καλώντας το circuit *hash_secret_vote*. Η κλήση σε αυτό το circuit επιστρέφει ένα hash για την ψήφο αλλά δεν αποκαλύπτει την πραγματική τιμή της ψήφου, δηλαδή τη witness μεταβλητή **secret_vote(voting_id)**.
    7. Τελευταίο βήμα είναι η εισαγωγή του hash που υπολογίστηκε πριν στη ledger μεταβλητή **hashed_votes**, συγκεκριμένα στο σύνολο για το συγκεκριμένο voting_id .
    8. Τέλος, γίνεται καταχώρηση της ψήφου στη ledger μεταβλητή **voting_results** όπου για το συγκεκριμένο **voting_id** αναζητάται το συγκεκριμένο **vote_option** και αυξάνεται ο μετρητής που του αντιστοιχεί κατά 1.

- **publish_vote**:

    1. Αρχικά ελέγχεται ότι η τρέχουσα ημερομηνία είναι προγενέστερη της ημερομηνίας λήξης δημοσίευσης ψήφου για τη συγκεκριμένη ψηφοφορία, χρησιμοποιώντας τη συνάρτηση βιβλιοθήκης **blockTimeLte** μέσω δήλωσης assert. Αν δεν είναι τότε εμφανίζεται κατάλληλο μήνυμα λάθους.
    2. Στη συνέχεια ελέγχεται ότι η τρέχουσα ημερομηνία είναι μεταγενέστερη της ημερομηνίας λήξης υποβολής ψήφου για τη συγκεκριμένη ψηφοφορία μέσω της συνάρτησης βιβλιοθήκης **blockTimeGt** μέσω δήλωσης assert. Αυτό γίνεται για να μην μπορεί κάποιος να δημοσιεύσει την ψήφο του χωρίς να την έχει υποβάλλει πρώτα. Έτσι μάλιστα επιτυγάνεται και ο διαχωρισμός στις δύο περιόδους ψηφοφορίας δηλαδή την περίοδο υποβολής ψήφου και την περίοδο δημοσίευσης ψήφου.
    3. Ελέγχεται ότι η ψηφοφορία είναι ανοικτή μέσω σχετικής δήλωσης **assert**. Αν δεν είναι ανοικτή, εμφανίζεται σχετικό μήνυμα λάθους και το transaction εγκαταλείπεται.
    4. Πραγματοποιούνται οι έλεγχοι (1), (2) από παραπάνω και αν κάποιος αποτύχει δηλαδή αν ο χρήστης είναι το πανεπιστήμιο, ή αν ο χρήστης δεν είναι έγκυρος τότε το transaction εγκαταλείπεται με σχετικό μήνυμα λάθους.
    5. Ελέγχεται ότι ο χρήστης δεν έχει εκτελέσει δεύτερη φορά το ίδιο transaction , ακολουθώντας ίδια λογική με το cast_vote transaction χρησιμοποιώντας nulifiers.
    6. Υπολογίζεται το hash της ψήφου, καλώντας το circuit hash_secret_vote με ορίσματα την ψήφο του χρήστη η οποία εξάγεται από τη witness μεταβλητή secret_vote, το id της ψηφοφορίας και το secret key. Μετά ελέγχεται αν αυτό το hash υπάρχει στο σύνολο που προκύπτει από την μεταβλητή hashed_votes, για τη συγκεκριμένη ψηφοφορία. Η ουσία εδώ είναι πως, αν τα δύο hashes δεν ταυτίζονται, σημαίνει πως μετά την εκτέλεση του cast_vote ο χρήστης άλλαξε το private state του και δηλαδή την ψήφο του, γιατί όπως έχουμε εξηγήσει, την private state μπορεί να την αλλάζει ο κάθε χρήστης όπως θέλει. Τότε απορρίπτεται το transaction και εμφανίζεται κατάλληλο μήνυμα λάθους. Αν από την άλλη ταυτίζονται τα 2 hashes, τότε η ψήφος δεν άλλαξε και συνεχίζονται οι έλεγχοι.
    7. Ελέγχεται αν η ψήφος είναι έγκυρη, δηλαδή αν το secret_vote για τη συγκεκριμένη ψηφοφορία ανήκει στο map voting_options του συγκεκριμένου id ψηφοφορίας.
    8. Τέλος, γίνεται καταχώρηση της ψήφου στη ledger μεταβλητή **voting_results** όπου για το συγκεκριμένο **voting_id** αναζητάται το συγκεκριμένο *vote_option* και αυξάνεται ο μετρητής που του αντιστοιχεί κατά 1.

Σε αυτό το σημείο, αξίζει να σημειωθεί πως οι ψήφοι σαν οντότητες δεν κρυπτογραφούνται, αλλά εισάγονται στο ledger σύνολο **voting_results**. Ωστόσο, υπάρχει πλήρης ανωνυμία καθώς η χρήση Merkle Tree εγγυάται ότι όταν ο τρέχων χρήστης αποδεικνύει εγκυρότητα βρίσκοντας ένα έγκυρο μονοπάτι μέσα στο δέντρο, δεν αποκαλύπτεται καμία πληροφορία για το public key του, και προφανώς για το secet key που χρησιμοποιήθηκε. Στο zero-knowledge transcript του publish_vote circuit η μόνη σημαντική πληροφορία που αποκαλύπτεται είναι η τιμή της ψήφου, μέσω της δήλωσης **disclose(secret_vote(voting_id))**. Ωστόσο, στο transcript δε φαίνεται πουθενά ποιο public key εκτέλεσε αυτό το transaction συνεπώς αποκρύπτεται η σύνδεση μεταξύ ενός ψηφοφόρου και της ψήφου του.
Επίσης, με τη χρήση του μοτίβου cast_vote - publish_vote, δηλαδή ψήφος και δημοσίευση της, λύνουμε ένα θέμα που θα υπήρχε αν χρησιμοποιούσαμε μόνο το cast_vote, το οποίο έγκειται στην εμφάνιση των αποτελεσμάτων της ψηφοφορίας. Αν υπήρχε μόνο cast_vote τότε, με το που ψηφίζει ένας χρήστης, καταχωρείται η ψήφους του στο voting_results, και στο interface της εφαρμογής , μπορεί οποιοσδήποτε να δει τα αποτελέσματα ενώ η ψηφοφορία είναι ακόμα ανοιχτή. Με αυτό τον τρόπο, θα μπορούσε να επηρεαστεί κάποιος ψηφοφόρος βλέποντας την

πορεία της ψηφοφορίας και είτε να μην ψηφίσει καθόλου, είτε να αλλάξει γνώμη. Με το μοτίβο publish_vote όμως, ο χρήστης όταν ψηφίζει κρατάει μυστική την ψήφο του και απλά δημοσιεύει μια δέσμευση σε αυτή, δηλαδή ένα hash. Η μεταβλητή voting_results δεν αλλάζει, συνεπώς δεν αλλάζει το αποτέλεσμα της ψηφοφορίας. Όταν η ψηφοφορία κλείσει, τότε οι χρήστες δημοσιεύουν την ψήφο τους (χωρίς να είναι δυνατή η σύνδεση του ποιος ψήφισε με το τι ψήφισε) και τα αποτελέσματα γίνονται ορατά.

```
export circuit create_voting(publish_vote_expiration_time_v: Uint<64>,
    cast_vote_expiration_time_v: Uint<64>): [] {
    const organizer_public_key = disclose(organizer(local_secret_key()));

    assert (publish_vote_expiration_time_v > cast_vote_expiration_time_v,
        "Publish vote deadline must be after cast vote deadline");

    assert (university_public_key != organizer_public_key,
        "University cannot create votings");

    assert (prove_eligibility(),
        "Not authorized!");

    const voting_id = generate_voting_id(count, local_secret_key());

    voting_organizers.insert(disclose(voting_id), organizer_public_key);
    votings.insert(disclose(voting_id));
    voting_states.insert(disclose(voting_id), VOTE_STATE.closed);
    voting_options.insert(disclose(voting_id), default<Set<Bytes<32>>>);
    voting_results.insert(disclose(voting_id), default<Map<Bytes<32>,
        Counter>>);
    publish_vote_expiration_time.insert(disclose(voting_id), disclose(
        publish_vote_expiration_time_v));
    cast_vote_expiration_time.insert(disclose(voting_id), disclose(
        cast_vote_expiration_time_v));
    count.increment(1);
}


export circuit edit_question(voting_id: Bytes<32>, voting_question: Opaque
    <"string">): [] {
    const organizer_public_key = disclose(organizer(local_secret_key()));

    assert (prove_eligibility(),
        "Not authorized");

    assert (voting_organizers.lookup(disclose(voting_id)) ==
        organizer_public_key,
        "Not authorized");

    const current_voting_state = voting_states.lookup(disclose(voting_id));

    assert (current_voting_state == VOTE_STATE.closed,
        "Cannot edit the question since the voting is open");

    voting_questions.insert(disclose(voting_id), disclose(voting_question))
        ;
}

export circuit add_option(voting_id: Bytes<32>, vote_option: Bytes<32>): []
    {
```

```
    const organizer_public_key = organizer(local_secret_key());

    assert (prove_eligibility(),
        "Not authorized");

    assert (voting_organizers.lookup(disclose(voting_id)) ==
       organizer_public_key,
        "Not authorized");

    const current_voting_state = voting_states.lookup(disclose(voting_id));

    assert (current_voting_state == VOTE_STATE.closed,
        "Cannot add option since the voting is open");

    voting_options.lookup(disclose(voting_id)).insert(disclose(vote_option)
       );
    voting_results.lookup(disclose(voting_id)).insert(disclose(vote_option)
       , default<Counter>);

}


export circuit open_voting(voting_id: Bytes<32>): [] {
    const organizer_public_key = organizer(local_secret_key());

    assert (prove_eligibility(),
        "Not authorized");

    assert (voting_organizers.lookup(disclose(voting_id)) ==
       organizer_public_key,
        "Not authorized");

    const current_voting_state = voting_states.lookup(disclose(voting_id));

    assert (current_voting_state == VOTE_STATE.closed,
        "Voting phase is not closed or has already opened");

    voting_states.insert(disclose(voting_id), VOTE_STATE.open);
}


export circuit cast_vote(voting_id: Bytes<32>): [] {
    const current_voting_state = voting_states.lookup(disclose(voting_id));
    const current_expiration_time = cast_vote_expiration_time.lookup(
       disclose(voting_id));
    assert (blockTimeLte(disclose(current_expiration_time)),
        "The deadline for vote casting has expired");

    assert (current_voting_state == VOTE_STATE.open,
        "Voting is not open");

    const voter_public_key = disclose(public_key(local_secret_key()));

    assert (prove_eligibility(),
        "Not authorized!");

    const voting_nullifier = disclose(nullifier(local_secret_key(),
       voting_id));
```

```
    assert (!voting_nulifiers.lookup(disclose(voting_id)).member(
        voting_nullifier),
        "Already voted for this voting");

    voting_nulifiers.lookup(disclose(voting_id)).insert(disclose(
        voting_nullifier));
    const hashed_vote = disclose(hash_secret_vote(secret_vote(voting_id),
        voting_id, local_secret_key()));
    hashed_votes.lookup(disclose(voting_id)).insert(disclose(hashed_vote));
}

export circuit publish_vote(voting_id: Bytes<32>): [] {
    const current_voting_state = voting_states.lookup(disclose(voting_id));

    const current_publish_expiration_time = publish_vote_expiration_time.
        lookup(disclose(voting_id));
    assert (blockTimeLte(disclose(current_publish_expiration_time)),
        "The deadline for vote publishing has expired");

    const current_cast_expiration_time = cast_vote_expiration_time.lookup(
        disclose(voting_id));
    assert (blockTimeGt(disclose(current_cast_expiration_time)),
        "The deadline for vote casting has not expired yet");

    assert (current_voting_state == VOTE_STATE.open,
        "Voting is not open");

    assert (prove_eligibility(),
        "Not authorized!");

    const publish_voting_nullifier = disclose(publish_nullifier(
        local_secret_key(), voting_id));

    assert (!publish_voting_nulifiers.lookup(disclose(voting_id)).member(
        publish_voting_nullifier),
        "Already published the vote for this voting");

    const hashed_vote = disclose(hash_secret_vote(secret_vote(voting_id),
        voting_id, local_secret_key()));

    assert (hashed_votes.lookup(disclose(voting_id)).member(disclose(
        hashed_vote)),
        "Vote not correct!");

    assert (voting_options.lookup(disclose(voting_id)).member(disclose(
        secret_vote(voting_id))),
        "Not a valid option!");
    voting_results.lookup(disclose(voting_id)).lookup(disclose(secret_vote(
        voting_id))).increment(1);
    publish_voting_nulifiers.lookup(disclose(voting_id)).insert(
        publish_voting_nullifier);
}
```

Σχήμα 3.7: Τα impure circuits του smart contract

3.3 Ανάλυση του κώδικα της dapp

3.3.1 Από Compact σε Typescript

Μετά την μεταγλώττιση του contract, δημιουργείται για κάθε impure circuit το αντίστοιχο .zkir αρχείο που είναι μια αναπαράσταση του κυκλώματος μηδενικής γνώσεις, καθώς και τα αρχεία .prover και .verifier που αποτελούν τα κλειδιά απόδειξης και επαλήθευσης του πρωτόκολλου μηδενικής γνώσης. Επίσης, δημιουργείται αρκετός κώδικα Typescript ο οποίος μπορεί να θεωρηθεί ως μαύρο κουτί καθώς πραγματοποιεί low-level διεργασίες όπως η υλοποίηση των κυκλωμάτων μηδενικής γνώσης. Ωστόσο, σημαντικό είναι το αρχείο **index.d.ts** που δημιουργεί απαραίτητους τύπους για τη συνέχεια. Εστιάζουμε στον τύπο **Ledger** που ουσιαστικά είναι η αναπαράσταση σε Typescript της public state, καθώς περιέχει σαν πεδίο κάθε ledger μεταβλητή που ορίστηκε στο contract. Με αυτό τον τρόπο, η εφαρμογή μπορεί να έχει πρόσβαση στην public state. Επίσης η κλάση **Contract** περιέχει τους witnesses, όλα τα circuits, τον constructor του contract και την αρχική κατάσταση.

```
export type Ledger = {
  readonly count: bigint;
  readonly university_public_key: Uint8Array;
  votings: {
    isEmpty(): boolean;
    size(): bigint;
    member(elem_0: Uint8Array): boolean;
    [Symbol.iterator](): Iterator<Uint8Array>
  };
  voting_options: {
    isEmpty(): boolean;
    size(): bigint;
    member(key_0: Uint8Array): boolean;
    lookup(key_0: Uint8Array): {
      isEmpty(): boolean;
      size(): bigint;
      member(elem_0: Uint8Array): boolean;
      [Symbol.iterator](): Iterator<Uint8Array>
    }
  };
  voting_questions: {
    isEmpty(): boolean;
    size(): bigint;
    member(key_0: Uint8Array): boolean;
    lookup(key_0: Uint8Array): string;
    [Symbol.iterator](): Iterator<[Uint8Array, string]>
  };
  voting_results: {
    isEmpty(): boolean;
    size(): bigint;
    member(key_0: Uint8Array): boolean;
    lookup(key_0: Uint8Array): {
      isEmpty(): boolean;
      size(): bigint;
      member(key_1: Uint8Array): boolean;
      lookup(key_1: Uint8Array): { read(): bigint }
    }
  };
  eligible_voters: {
    isFull(): boolean;
    checkRoot(rt_0: { field: bigint }): boolean;
```

```
    root(): __compactRuntime.MerkleTreeDigest;
    firstFree(): bigint;
    pathForLeaf(index_0: bigint, leaf_0: Uint8Array): __compactRuntime.
        MerkleTreePath<Uint8Array>;
    findPathForLeaf(leaf_0: Uint8Array): __compactRuntime.MerkleTreePath<
        Uint8Array> | undefined;
    history(): Iterator<__compactRuntime.MerkleTreeDigest>
  };
  voting_states: {
    isEmpty(): boolean;
    size(): bigint;
    member(key_0: Uint8Array): boolean;
    lookup(key_0: Uint8Array): VOTE_STATE;
    [Symbol.iterator](): Iterator<[Uint8Array, VOTE_STATE]>
  };
  voting_nulifiers: {
    isEmpty(): boolean;
    size(): bigint;
    member(elem_0: Uint8Array): boolean;
    [Symbol.iterator](): Iterator<Uint8Array>
  };
  publish_voting_nulifiers: {
    isEmpty(): boolean;
    size(): bigint;
    member(elem_0: Uint8Array): boolean;
    [Symbol.iterator](): Iterator<Uint8Array>
  };
  voting_organizers: {
    isEmpty(): boolean;
    size(): bigint;
    member(key_0: Uint8Array): boolean;
    lookup(key_0: Uint8Array): Uint8Array;
    [Symbol.iterator](): Iterator<[Uint8Array, Uint8Array]>
  };
  hashed_votes: {
    isEmpty(): boolean;
    size(): bigint;
    member(elem_0: Uint8Array): boolean;
    [Symbol.iterator](): Iterator<Uint8Array>
  };
  publish_vote_expiration_time: {
    isEmpty(): boolean;
    size(): bigint;
    member(key_0: Uint8Array): boolean;
    lookup(key_0: Uint8Array): bigint;
    [Symbol.iterator](): Iterator<[Uint8Array, bigint]>
  };
  cast_vote_expiration_time: {
    isEmpty(): boolean;
    size(): bigint;
    member(key_0: Uint8Array): boolean;
    lookup(key_0: Uint8Array): bigint;
    [Symbol.iterator](): Iterator<[Uint8Array, bigint]>
  };
}

export declare class Contract<T, W extends Witnesses<T> = Witnesses<T>> {
  witnesses: W;
```

```
  circuits: Circuits<T>;
  impureCircuits: ImpureCircuits<T>;
  constructor(witnesses: W);
  initialState(context: __compactRuntime.ConstructorContext<T>,
               eligible_voter_public_keys_0: Uint8Array[]):
                  __compactRuntime.ConstructorResult<T>;
}
```

Σχήμα 3.8: Το αρχείο index.d.ts

3.3.2 Το api

Έχοντας διαθέσιμους όλους τους απαραίτητους τύπος που έχουν προκύψει από τη μεταγλώττιση της Compact σε Typescript, σημαντικό ρόλο στην εφαρμογή κατέχει το api το οποίο ουσιαστικά λειτουργεί ως μια διεπαφή μέσω της οποίας ο χρήστης μπορεί να δημοσιεύσει ένα contract, να αναζητήσει ένα ήδη δυμοσιευμένο contract και να πραγματοποιήσει τα διαθέσιμα transcations του contract στο δίκτυο. Συγκεκριμένα, το api περιέχει το interface **DeployedVoteGuardianAPI** που περιλαμβάνει την διεύθυνση του contract, την κατάστασή του, και όλα τα transactions ως typescript συναρτήσεις αλλά με τα ίδια ορίσματα και αντίστοιχους τύπους επιστροφής με τα ανάλογα της compact.

```
export interface DeployedVoteGuardianAPI {
  readonly deployedContractAddress: ContractAddress;
  readonly state$: Observable<VoteGuardianDerivedState>;

  cast_vote: (voting_id: Uint8Array) => Promise<void>;
  close_voting: (voting_id: Uint8Array) => Promise<void>;
  open_voting: (voting_id: Uint8Array) => Promise<void>;
  edit_question: (voting_id: Uint8Array, vote_question: string) => Promise<
     void>;
  create_voting: (publish_vote_expiration_time: bigint,
     cast_vote_expiration_time: bigint) => Promise<void>;
  add_option: (voting_id: Uint8Array, vote_option: Uint8Array) => Promise<
     void>;
  publish_vote: (voting_id: Uint8Array) => Promise<void>;
}
```

Σχήμα 3.9: Το DeployedVoteGuardianAPI interface

Η κλάση **VoteGuardianAPI** υλοποιεί το παραπάνω interface, αρχικοποιώντας την κατάστασή του με την τρέχουσα κατάσταση που υπάρχει στο contract και τη διεύθυνσή του με τη διεύθυνση του contract. Για κάθε transaction παρέχεται η υλοποίησή του, η οποία στον πυρήνα της καλεί την αντίστοιχη συνάρτηση που έχε προκύψει από την compact έχοντας μεταγλωττιστεί σε typescript. Ενδεικτικά, παρουσιάζουμε την υλοποίηση του transaction **edit_question** καθώς η λογική είναι ίδια στα υπόλοιπα.

```
async edit_question(voting_id: Uint8Array, vote_question: string): Promise<
   void> {
    try {
      this.logger?.info(`vote question: ${vote_question}`);
      const txData = await this.deployedContract.callTx.edit_question(
         voting_id, vote_question);

      this.logger?.trace({
        transactionAdded: {
          circuit: 'edit_question',
```

```
          txHash: txData.public.txHash,
          blockHeight: txData.public.blockHeight,
        },
      });
    } catch (error) {
      console.log(error);
      this.logger?.error('Error editing a question', {
        message: (error as Error).message,
        stack: (error as Error).stack,
        details: error,
      });
    }
  }
```

Σχήμα 3.10: Η κλήση του edit_question μέσω typescript

Επίσης, η κλάση **VoteGuardianAPI** παρέχει δύο στατικές μεθόδους, την **deploy** για δημοσίευση ενός contract και την **join** για την αναζήτηση ενός contract και συμμετοχή σε αυτό. Και οι δύο αυτές μέθοδοι επιστρέφουν ένα αντικείμενο της κλάσης **VoteGuardianAPI**. Και οι δύο αυτές μέθοδοι πρέπει να αρχικοποιήσουν την private state καθώς όταν κάποιος είτε δημοσιεύει ένα contract είτε κάνει join σε αυτό, πρέπει να έχει μια αρχική private state. Στη δική μας περίπτωση, όπως έχουμε πει η private state αποτελείται από το secret key, το έγκυρο μονοπάτι μέσα στο merkle tree (μεταβλητή **voterPublicKeyPath**) καθώς και τις ψήφους του χρήστη για κάθε ψηφοφορία (μεταβλητή **votesPerVotingMap**). Το secret key δίνεται ως όρισμα στις deploy, join ώστε ο χρήστης που τις καλεί να δώσει το δικό του secret key (ως γνωστόν δε φεύγει ποτέ από το δικό του υπολογιστή).
Για τις άλλες δύο μεταβλητές, ελέγχεται αν υπάρχει ήδη κάποια αποθηκευμένη τιμή της μεταβλητής στην τοπική leveldb. Αν υπάρχει, σημαίνει πως υπάρχει ήδη κάποια private state για αυτές τις δύο μεταβλητές, συνεπώς επιστρφεται αυτή. Αν δεν υπάρχει, σημαίνει πως πρώτη φορά ο χρήστης κάνει join στο smart contract συνεπώς αρχικοποιούνται κατάλληλα αυτές οι μεταβλητές. Η voterPublicKeyPath θα πάρει μια dummy τιμή με μηδενικά, καθώς σε αυτό το σημείο δε μας ενδιαφέρει να είναι έγκυτο το μονοπάτι στο δέντρο, η σωστή τιμή θα δοθεί όταν ο χρήστης πραγματοποιήσει κάποιο transaction που απαιτή αυθεντικοποίηση όπου το αντίστοιχο circuit θα αλλάξει την τιμή της voterPublicKeyPath. Η votesPerVotingMap αρχικοποιείται ως ένα άδειο map από συμβολοσειρά σε συμβολοσειρά **Map<String, String>**.
Παραθέτουμε την υλοποίηση της join, καθώς και τις συναρτήσεις **getPrivateStateMerklePath**, **getPrivateStateVotesMap** οι οποίες ψάχνουν αν υπάρχει ήδη τιμή για την αντίστοιχη witness μεταβλητή και αν δεν υπάρχει την αρχικοποιούν όπως περιγράψαμε παραπάνω.

```
static async join(
    providers: VoteGuardianProviders,
    contractAddress: ContractAddress,
    secretKey: string,
    logger?: Logger,
  ): Promise<VoteGuardianAPI> {
    logger?.info({
      joinContract: {
        contractAddress,
      },
    });

    const privateStateMerklePath = await this.getPrivateStateMerklePath(
       providers);
    const privateStateVotersMap = await this.getPrivateStateVotesMap(
```

```
      providers);
  const deployedVoteGuardianContract = await findDeployedContract(
      providers, {
    contractAddress,
    contract: VoteGuardianContractInstance,
    privateStateId: 'voteGuardianPrivateState',
    initialPrivateState:
      // (await providers.privateStateProvider.get('
          voteGuardianPrivateState')) ??
      createVoteGuardianPrivateState(utils.hexToBytes(secretKey),
          privateStateMerklePath, privateStateVotersMap),
  });

  logger?.trace({
    contractJoined: {
      finalizedDeployTxData: deployedVoteGuardianContract.deployTxData.
          public,
    },
  });

  return new VoteGuardianAPI(deployedVoteGuardianContract, providers,
      logger);
}

private static async getPrivateStateMerklePath(
  providers: VoteGuardianProviders,
): Promise<MerkleTreePath<Uint8Array>> {
  const existingPrivateState = await providers.privateStateProvider.get('
      voteGuardianPrivateState');
  return (
    existingPrivateState?.voterPublicKeyPath ?? {
      leaf: new Uint8Array(32),
      path: [
        {
          sibling: { field: BigInt(0) },
          goes_left: false,
        },
      ],
    }
  );
}

private static async getPrivateStateVotesMap(providers:
    VoteGuardianProviders): Promise<Map<String, String>> {
  const existingPrivateState = await providers.privateStateProvider.get('
      voteGuardianPrivateState');
  return existingPrivateState?.votesPerVotingMap ?? new Map<String,
      String>();
}
```

Σχήμα 3.11: Ημέθοδος join, και οι getPrivateStateMerklePath, getPrivateStateVotesMap

3.3.3 To user interface

Το user interface της εφαρμογής γράφτηκε χρησιμοποιώντας *ReactJS* και στην ουσία αξιοποιεί τους τύπους και τα αντικείμενα που προσφέρονται από το api και από τον κώδικα Typescript που παράγεται από την compact. Εκτενής παρουσίαση του interface θα γίνει σε επόμενο κεφά-

λαιο, παρουσιάζοντας διάφορες περιπτώσεις χρήσης. Ωστόσο, μια σημαντική λειτουργία που πραγματοποιείται από τον κώδικα του ui είναι η αρχικοποίηση σημαντικών παραμέτρων, οι οποίες ονομάζονται **providers** για την σωστή λειτουργία της εφαρμογής στο Midnight, όπως είναι η σύνδεση του Lace Wallet με την εφαρμογή, η διαχείριση της private state κάθε χρήστη και η σύνδεση με τον proofServer. Αυτές οι λειτουργίες πραγματοποιούνται μέσω της συνάρτησης **initializeProviders** που φαίνεται στο σχήμα 3.11. Συγκεκριμένα αυτή η συνάρηση:

1. Επιστρέφει τιμή τύπου **MidnightProviders**.
2. Ως privateStateProvider, δηλαδή ως μέσο αποθήκευσης της private state κάθε χρήστη ορίζει τη level-db του browser της οποίας η υλοποίηση έρχεται από τη βιβλιοθήκη του Midnight.
3. Ορίζει ως proof server αυτόν που έχει ορίσει ο χρήστης μέσω του wallet του, δηλώνοντας εκεί τη διεύθυνση στην οποία τρέχει ο proof server (η οποία είναι συνήθως localhost:6000).
4. Δηλώνει τη διεύθυνση του indexer, δηλαδή του συστατικού εκείνου μέσω του οποίου γίνεται η αναζήτηση για δεδομένα στο δίκτυο, δεδομένα όπως η public state του contract. Τη διεύθυνση του indexer τη δίνει ο χρήστης μέσω του lace wallet.
5. Συνδέει το lace wallet με την εφαρμογή, αρχικοποιώντας τον walletProvider.
6. Δηλώνει από που θα λαμβάνονται τα κλειδιά και τα circuits που αφορούν τις αποδείξεις μηδενικής γνώσεις, αρχικοποιώντας τον **zkConfigProvider**.
7. Αρχικοποιεί τον **midnightProvider** ώστε να μπορούν να πραγματοποιούνται transactions στο Midnight.

```
const initializeProviders = async (logger: Logger): Promise<
   VoteGuardianProviders> => {
  const { wallet, uris } = await connectToWallet(logger);
  const walletState = await wallet.state();

  return {
    privateStateProvider: levelPrivateStateProvider({
      privateStateStoreName: 'voteGuardian-private-state',
    }),
    zkConfigProvider: new FetchZkConfigProvider(window.location.origin,
       fetch.bind(window)),
    proofProvider: httpClientProofProvider(uris.proverServerUri),
    publicDataProvider: indexerPublicDataProvider(uris.indexerUri, uris.
       indexerWsUri),
    walletProvider: {
      coinPublicKey: walletState.coinPublicKey,
      encryptionPublicKey: walletState.encryptionPublicKey,
      balanceTx(tx: UnbalancedTransaction, newCoins: CoinInfo[]): Promise<
         BalancedTransaction> {
        return wallet
          .balanceTransaction(
            ZswapTransaction.deserialize(tx.serialize(getLedgerNetworkId())
               , getZswapNetworkId()),
            newCoins,
          )
          .then((tx) => wallet.proveTransaction(tx))
          .then((zswapTx) => Transaction.deserialize(zswapTx.serialize(
             getZswapNetworkId()), getLedgerNetworkId()))
```

```
          .then(createBalancedTx);
      },
    },
    midnightProvider: {
      submitTx(tx: BalancedTransaction): Promise<TransactionId> {
        return wallet.submitTransaction(tx);
      },
    },
  };
};
```

Σχήμα 3.12: Η αρχικοποίηση των Midnight Providers

3.4 Ανάλυση του server του πανεπιστημίου

3.4.1 Αυθεντικοποίηση των φοιτητών

Στην αρχική οθόνη της εφαρμογής υπάρχει η επιλογή Authenticate μέσω της οποίας πραγματοποιείται η αυθεντικοποίηση των έγκυρων ψηφοφόρων. Η διαδικασία έχει ως εξής:

1. Ο χρήστης καταχωρεί τα στοιχεία του, δηλαδή username και password, ενώ ταυτόχρονα υπολογίζεται μια τυχαία συμβολοσειρά από 32 byte η οποία θα είναι το υποψήφιο secret key του χρήστη. Ο χρήστης δεν αναμειγνύεται στη δημιουργία αυτής της συμβολοσειράς, αντιθέτως γίνεται αυτόματα από τν κώδικα της dapp.

2. Έχοντας υπολογίσει το υποψήφιο secret key, υπολογίζεται και το hash του μέσω της συνάρτησης καρακερματισμού sha256 και στέλνεται http request στο endpoint του server του πανεπιστημίου **/login**.

3. Αν για τον συγκεκριμένο χρήστη, δηλαδή για το συγκεκριμένο ζεύγος username, password δεν υπάρχει καταχωρημένη τιμή στο πεδίο publicKey, σημαίνει πως ο χρήστης πρώτη φορά αυθεντικοποιείται, συνεπώς η διαδικασία είναι επιτυχής. Ταυτόχρονα εμφανίζεται στον χρήστη το secret key του με προτροπή να το αποθηκεύσει κάπου με ασφάλεια. Είναι σημαντικό να τονίσουμε εδώ ότι το secret key δεν φεύγει ποτέ από το τοπικό περιβάλλον του χρήστη, διότι υπολογίζεται τοπικά και αποστέλεται με request μόνο το hash του.

4. Αν ο χρήστης δεν υπάρχει στη βάση ή αν έχει ήδη δηλώσει κάποιο άλλο secret key τότε η διαδικασία αποτυγχάνει και εμφανίζεται αντίστοιχο μήνυμα λάθους.

Σημειώνουμε ότι υπάρχει και ένα δεύτερο endpoint που παρέχει ο server του πανεπιστημίου, το **/register**, μέσω του οποίου καταχωρούνται οι φοιτητές στη βάση δεδομένων.
Σε αυτό το σημείο πρέπει να σημειώσουμε ότι η αποθήκευση των public keys κάθε φοιτητή στη βάση δεδομένων αυξάνει το βαθμό κεντρικοποίησης του συστήματός μας. Ωστόσο, κατά την υποβολή και δημοσίευση ψήφων, επειδή η αυθεντικοποίηση γίνεται με χρήση Merkle Tree, δεν γίνεται ορατό στον έξω κόσμο το τρέχον public key που χρησιμοποείται. Δηλαδή, το γεγονός ότι υπάρχει συσχέτιση μεταξύ της ταυτότητας κάθε φοιτητή με το public key του δεν μπορεί να αποκαλύψει το τι ψήφισε κάθε φοιτητής επειδή η απόδειξη μηδενικής γνώσης που υποβάλλεται δεν περιέχει στις δημόσιες πληροφορίες το public key του φοιτητή που ψηφίζει ακόμα κι αν περιέχει την ψήφο του. Έτσι, ποτέ δεν υπάρχει συσχετισμός public key φοιτητή με ψήφο φοιτητή.

3.4.2 Δημοσίευση του smart contract από το πανεπιστήμιο

Όπως έχουμε προαναφέρει η συμμετοχή του πανεπιστημίου στο σύστημα έγγειται στη δημιουργία και δημοσίευση του smart contract στο Midnight, έχοντας συγκεντρώσει τα public keys των έγγυρων χρηστών από τη βάση δεδομένων. Για να δημοσιεύσει ένα smart contract στο Midnight, το πανεπιστήμιο θα πρέπει να τρέξει τοπικά τον proof server όπως και όλοι οι voters ενώ θα χρειαστεί να έχει το δικό του wallet, καθώς το deploy ενός smart contract στο δίκτυο χρεώνεται. Καθώς όμως το πανεπιστήμιο δεν έχει user interface, δε θα χρησιμοποιήσει το lace wallet, αντιθέτως θα δημιουργήσει ένα δικό του wallet μέσω βιβλιοθηκών του Midnight, το οποίο ωστόσο θα έχει τις ίδιες δυνατότητες με το lace wallet.

```
const wallet = await buildWallet(config, rli, logger);

try {
  if (wallet !== null) {
    const walletAndMidnightProvider = await createWalletAndMidnightProvider
       (wallet);
    const providers = {
      privateStateProvider: inMemoryPrivateStateProvider<'
         voteGuardianPrivateState', VoteGuardianPrivateState>(),

      publicDataProvider: indexerPublicDataProvider(config.indexer, config.
         indexerWS),

      zkConfigProvider: new NodeZkConfigProvider<
        'cast_vote' | 'close_voting' | 'create_voting' | 'add_option' | '
           open_voting' | 'edit_question' | 'publish_vote'
      >(config.zkConfigPath),
      proofProvider: httpClientProofProvider(config.proofServer),
      walletProvider: walletAndMidnightProvider,
      midnightProvider: walletAndMidnightProvider,
    };
    await mainLoop(providers, rli, logger);
  }
}
```

Σχήμα 3.13: Η δημιουργία του wallet και των providers από το πανεπιστήμιο

Καλλώντας την μέθοδο **buildWallet** το πανεπιστήμιο κατασκευάζει ένα wallet με ίδιες δυνατότητες με αυτές του lace wallet. Στη συνέχεια, δημιουργείται το αντικείμενο providers που περιέχει όλες τις ιδιότητες που χρειάζονται για τη δημιουργία ενός smart contract. Η δημιουργία του smart contract γίνεται μέσω της **mainLoop**.

```
const mainLoop = async (providers: VoteGuardianProviders, rli: Interface,
   logger: Logger): Promise<void> => {
  const secretKeyBytes = new Uint8Array(32);
  const secretKey = toHex(secretKeyBytes);


  let eligibleVoters = [];
  const users = await User.find({}).select('publicKey -_id');

  eligibleVoters = users.map((user) => user.publicKey);
  const eligibleVotersUint8: Uint8Array[] = eligibleVoters.map((voterStr)
     => {
```

```
    return stringToUint8Array(voterStr!);
  });

  const contractAddressFile = path.resolve(process.cwd(), 'contract_address
      .txt');
  const contractAddress = fs.readFileSync(contractAddressFile, 'utf8').trim
      ();
  let VoteGuardianApi: VoteGuardianAPI | null;

  if (contractAddress) {
    VoteGuardianApi = await VoteGuardianAPI.join(providers, contractAddress
        , secretKey, logger);
  } else {
    VoteGuardianApi = await VoteGuardianAPI.deploy(providers, secretKey,
        eligibleVotersUint8, logger);
  }

  if (VoteGuardianApi === null) {
    return;
  }
  console.log(`deployed - joined at address ${VoteGuardianApi.
      deployedContractAddress}`);

  fs.writeFileSync(contractAddressFile, VoteGuardianApi.
      deployedContractAddress, 'utf8');
  let currentState: VoteGuardianDerivedState | undefined;
  const stateObserver = {
    next: (state: VoteGuardianDerivedState) => (currentState = state),
  };
  const subscription = VoteGuardianApi.state$.subscribe(stateObserver);
};
```

Σχήμα 3.14: Η mainLoop

Αρχικά το πανεπιστήμιο βρίσκει από τη βάση δεδομένων όλα τα public keys των φοιτητών και τα αποθηκεύει στη μεταβλητή **eligibleVoters**. Αν έχει ήδη δημοσιεύσει κάποιο smart contract τότε χρησιμοποιεί τη μέθοδο **join** για εύρεση αυτού του smart contract. Αν είναι η πρώτη φορά που δημοσιεύεται κάποιο smart contract τότε χρησιμοποιεί τη μέθοδο deploy του VoteGuardianApi δίνοντας ως παραμέτρους τους providers, ένα τυχαίο secret key το οποίο χρησιμοποιείται ως placeholder και έναν πίνακα με τα public keys των ψηφοφόρων. Παραθέτουμε την υλοποίηση της **deploy**.

```
static async deploy(
    providers: VoteGuardianProviders,
    secretKey: string,
    eligibleVoterPublicKeys: Uint8Array[],
    logger?: Logger,
  ): Promise<VoteGuardianAPI | null> {
    try {

      const DeployedVoteGuardianContract = await deployContract(providers,
          {
        privateStateId: 'voteGuardianPrivateState',
        contract: VoteGuardianContractInstance,
```

```
        initialPrivateState: createVoteGuardianPrivateState(
          utils.hexToBytes(secretKey),
          {
            leaf: new Uint8Array(32),
            path: [
              {
                sibling: { field: BigInt(0) },
                goes_left: false,
              },
            ],
          },
          new Map<String, String>(),
        ),
        args: [eligibleVoterPublicKeys],
      });


      logger?.trace({
        contractDeployed: {
          finalizedDeployTxData: DeployedVoteGuardianContract.deployTxData.
              public,
        },
      });

      return new VoteGuardianAPI(DeployedVoteGuardianContract, providers,
          logger);
    } catch (error) {
      console.log((error as Error));
      return null;
    }
```

Σχήμα 3.15: Η μέθοδος deploy

Η μέθοδος deploy χρησιμοποιεί τη συνάρτηση βιβλιοθήκης deployContract δίνοντας ως ορίσματα τα public keys των φοιτητών (που χρησιμοποιούνται στον κατασκευαστή του contract ως τα στοιχεία του Merkle Tree) καθώς και μια private state ως placeholder. Επειδή το πανεπιστήμιο δε συμμετέχει στις διαδικασίες ψηφοφορίας και δε μπορεί να πραγματοποιήσει transactions, όλες οι witness μεταβλητές αρχικοποιούνται σε dummy τιμές. Μόνος σκοπός του πανεπιστημίου είναι να δημοσιεύει το smart contract.

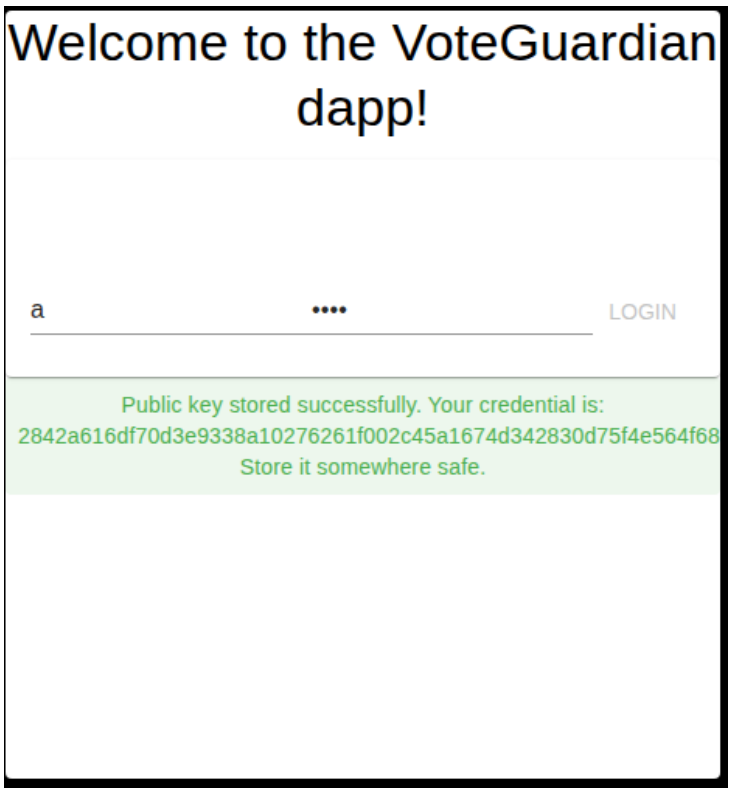
Κεφάλαιο 4

Περιπτώσεις Χρήσης

Σε αυτό το κεφάλαιο θα παρουσιάσουμε τις περιπτώσεις χρήσης της εφαρμογής, δείχνοντας πώς οι χρήστες μπορούν να την αξιοποιήσουν για να πραγματοποιήσουν τα διαθέσιμα transactions, δημιουργώντας νέες ψηφοφορίες, πραγματοποιώντας αλλαγές σε αυτές και συμμετέχοντας σε αυτές με την ψήφο τους. Με αυτόν τον τρόπο θα επιβεβαιώσουμε στην πράξη ότι οι κανόνες που έχουν οριστεί από το smart contract εφαρμόζονται σωστά και λειτουργούν όπως σχεδιάστηκαν.

4.1 Αυθεντικοποίηση φοιτητών

Όπως αναλύσαμε προηγουμένως οι εγγεγραμένοι στο πανεπιστήμιο φοιτητές δημιουργούν μόνοι τους το μυστικό κλειδί και αποστέλλουν μέσω http request το hash του κλειδιού (δηλαδή το public key), το username και το password στον server του πανεπιστημίου. Αν ο φοιτητής είναι έγκυρος χρήστης, καταχωρείται στη βάση το public key και επιστρέφεται στον χρήστη το secret key που ο ίδιος δημιούργησε. Τονίζουμε πάλι ότι το secret key ποτέ δε φεύγει από τον υπολογιστή του χρήστη.
Πατώντας στην επιλογή AUTHENTICATE της αρχικής οθόνης, ο χρήστης παραπέμπεται σε νέα σελίδα ώστε να καταχωρήσει τα στοιχεία του.



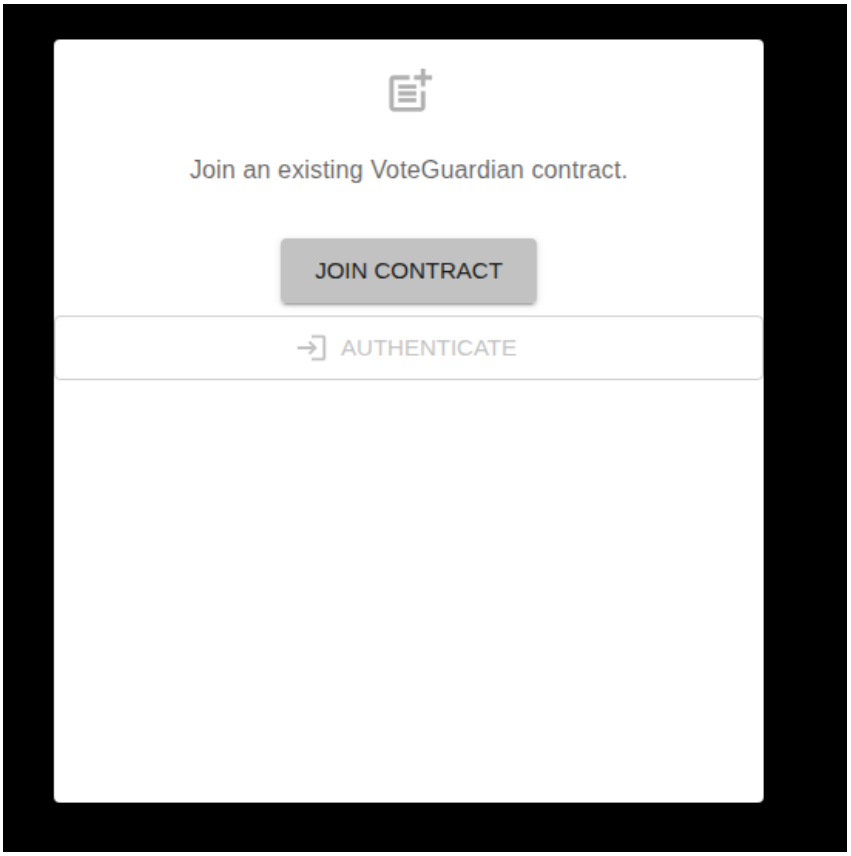
Σχήμα 4.1: Η σελίδα αυθεντικοποίησης

4.2 Δημιουργία Ψηφοφορίας

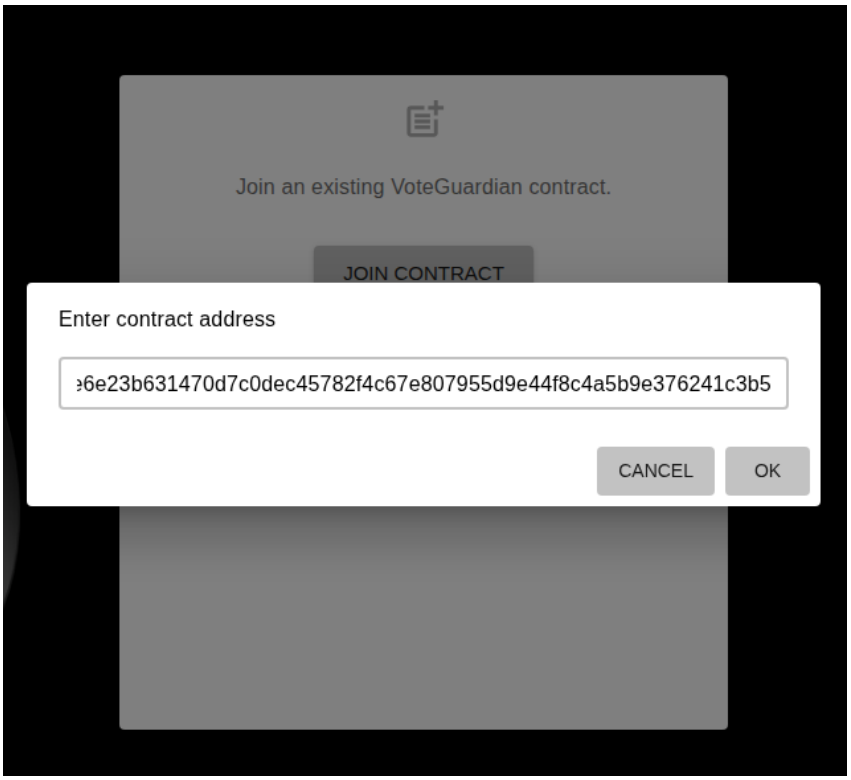
Για τις επόμενες ενότητες θα δουλέψουμε το εξής σενάριο: Έχουμε δύο έγκυρους χρήστες, ας πούμε τους user1, user2 με μυστικά κλειδία ec1d0ad62fbd918ea23a151264a4de5e59ecdce87d36f84ed6402ba4077dc30b και 9f3ec6ed2544939544eb3f231247c4111ed3e5263f5613bf3d8aea4b269a1191 αντίστοιχα. Έστω επίσης πως το smart contract βρίσκεται στη διεύθυνση 0200881de6e23b631470d7c0dec45782f4c67e807955d9e44f8c4a5b9e376241c3b5. Θεωρούμε πως αυτή η διεύθυνση είναι γνωστή και έχει κοινοποιηθεί από το πανεπιστήμιο στους φοιτητές.

4.2.1 Είσοδος στην dapp

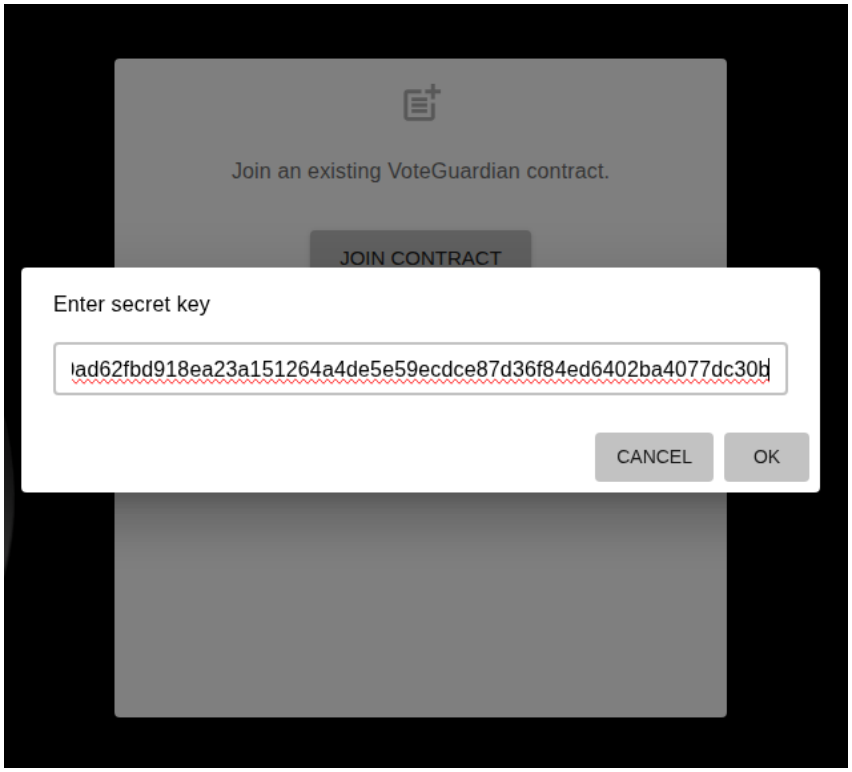
Όπως έχουμε ήδη εξηγήσει, κάθε φοιτητής μπορεί να δημιουργήσει ψηφοφορίες. Στην αρχική σελίδα της εφαρμογής ο χρήστης πρέπει να συμπληρώσει τη διεύθυνση του smart contract. Επίσης, ο φοιτητης συμπληρώνει το secret key του, ώστε να αρχικοποιηθεί η private state του. Σημειώνεται εδώ πως το secret key ποτέ δε "φεύγει" από τον υπολογιστή του χρήστη.΄Έστω λοιπόν πως ο user1 συνδέεται στην εφαρμογή, δίνοντας αρχικά τη διεύθυνση του smart contract καθώς και το secret key του, όπως φαίνεται στα σχήματα 4.3 και 4.4.



Σχήμα 4.2: Η αρχική οθόνη της dapp

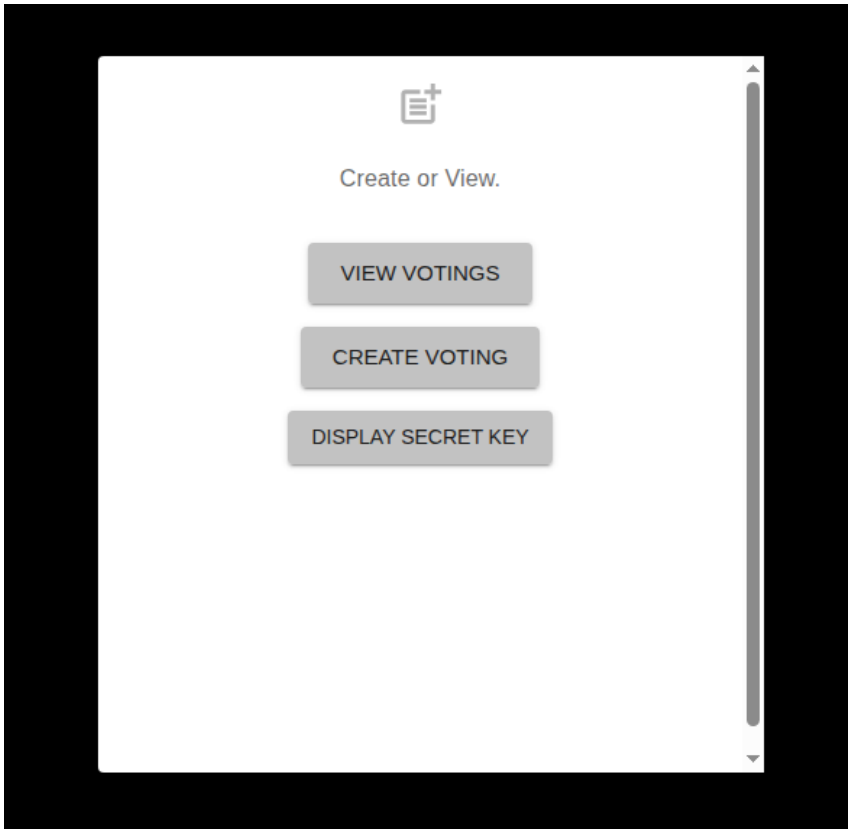


Σχήμα 4.3: Ο χρήστης user1 συμπληρώνει τη διεύθυνση του smart contract



Σχήμα 4.4: Ο χρήστης user1 συμπληρώνει το secret key του

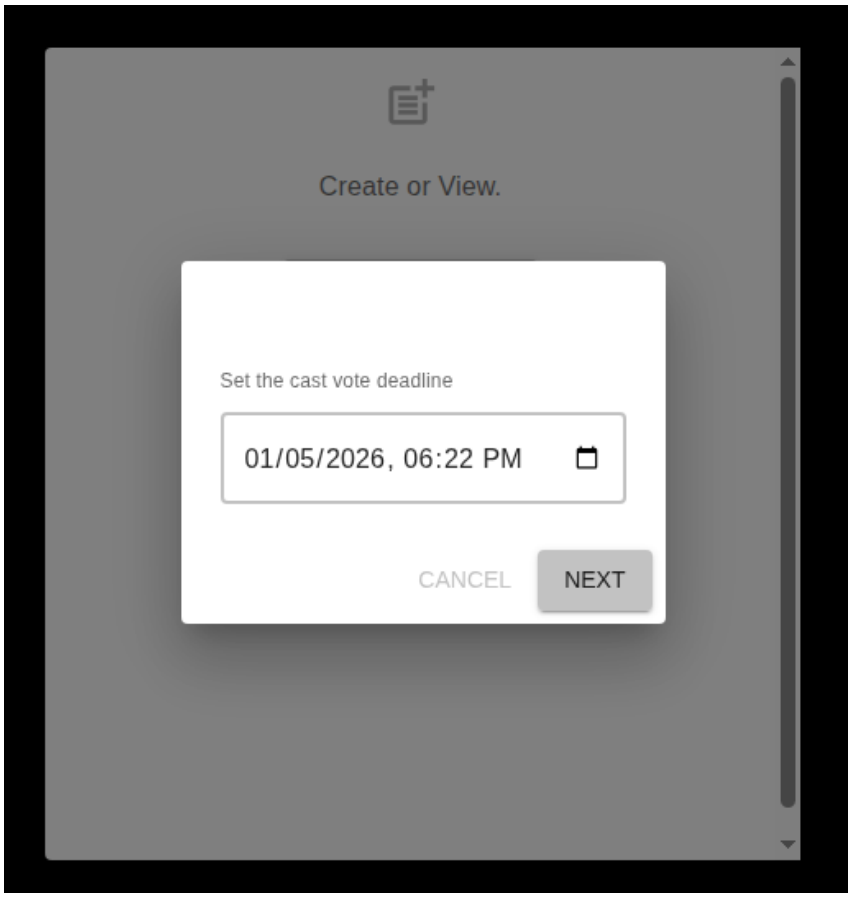
Στη συνέχεια, ο χρήστης οδηγείται στην αρχική οθόνη του smart contract όπου μπορεί να δει τις διαθέσιμες ψηφοφορίες ή να δημιουργήσει ο ίδιος μία, βλ. σχήμα 4.5.



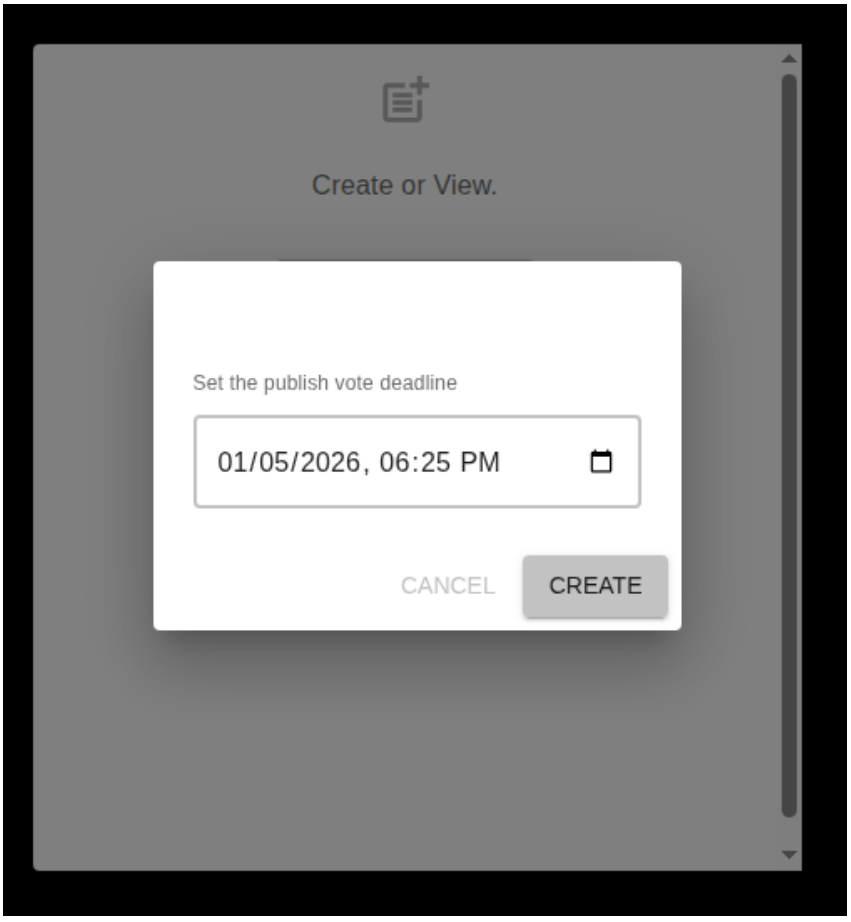
Σχήμα 4.5: Η αρχική οθόνη του smart contract

4.2.2 Δημιουργία ψηφοφορίας

Καθώς δεν υπάρχουν ακόμα ψηφοφορίες, ο χρήστης επιλέγει τη δημιουργία μιας νέας ψηφοφορίας μέσω του transaction create_voting. Πριν την υποβολή του transaction ομως, πρέπει να δηλώσει σε αντίστοιχο popup τις ημερομηνίες λήξης υποβολής ψηφοφορίας και δημοσίευσης ψηφοφορίας.

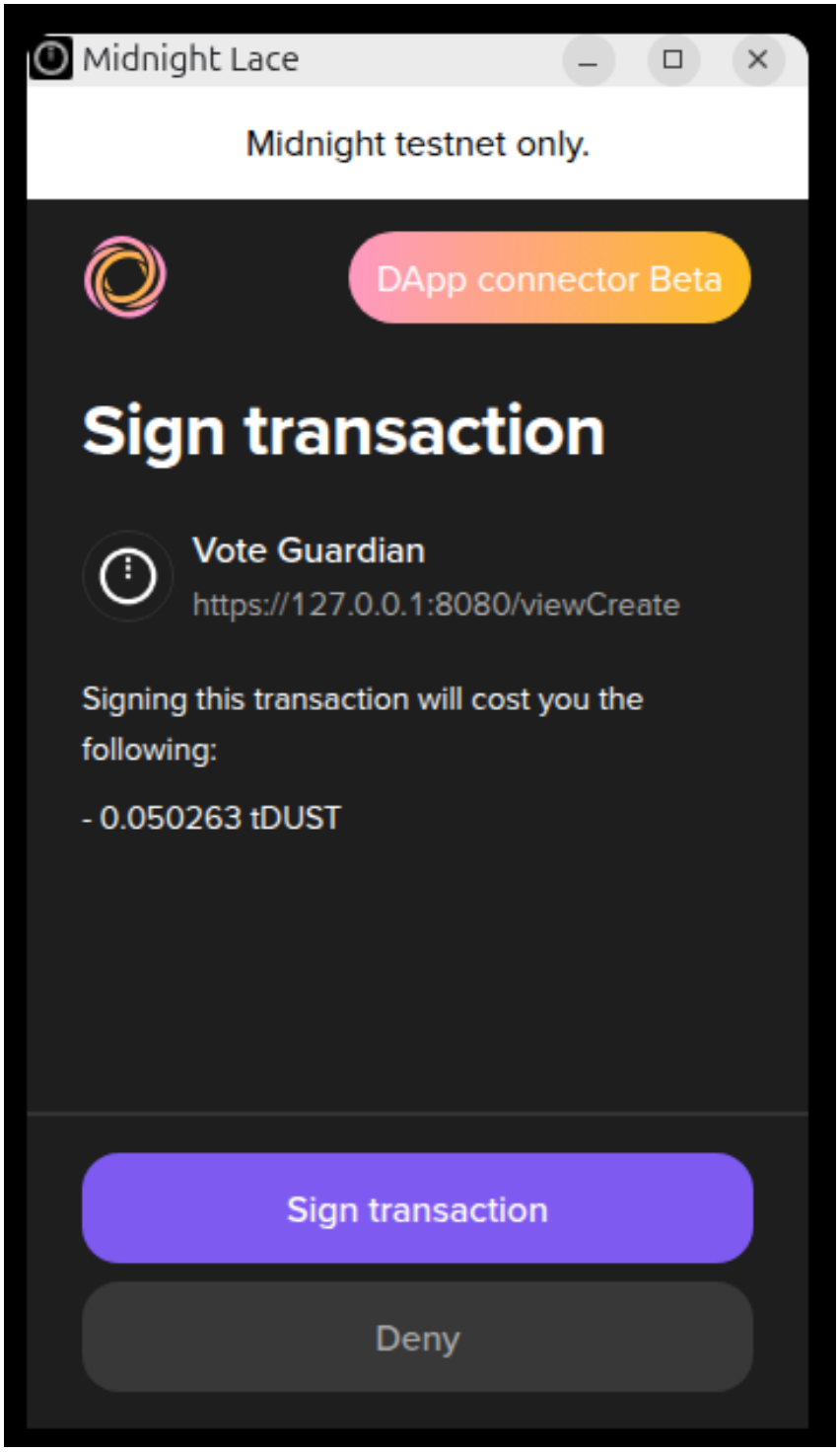


Σχήμα 4.6: Ο χρήστης δηλώνει την ημερομηνία λήξης υποβολής ψήφων



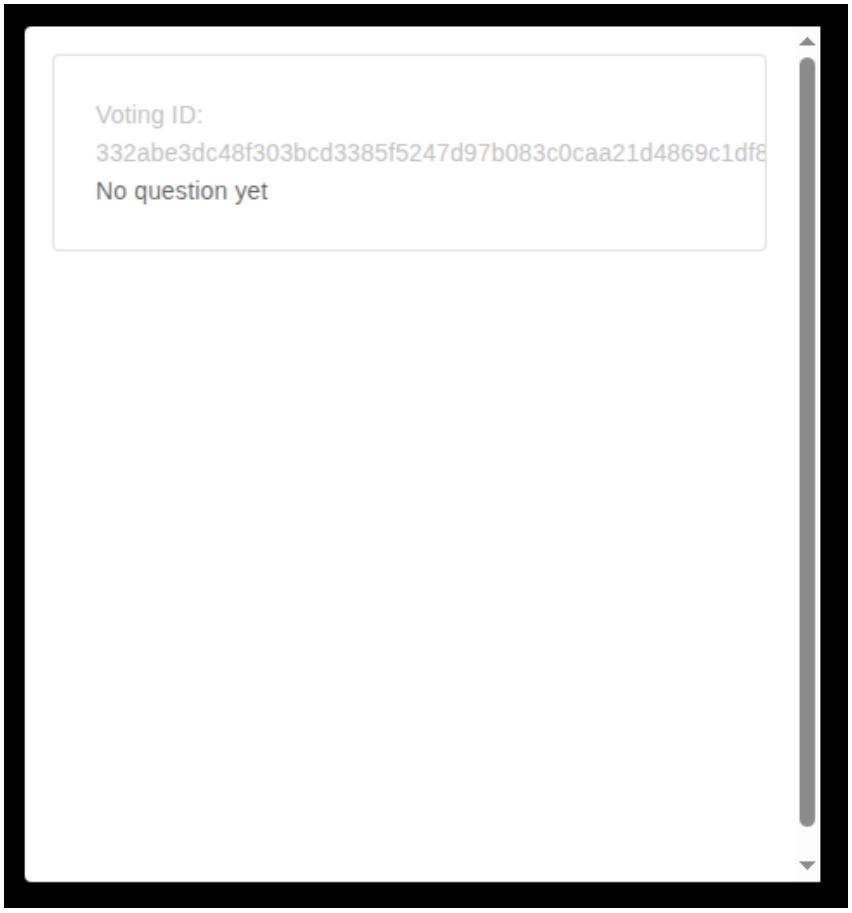
Σχήμα 4.7: Ο χρήστης δηλώνει την ημερομηνία λήξης δημοσίευσης ψήφων

Με αυτές τις επιλογές, θα εμφανιστεί το παράθυρο του lace wallet όπου ο χρήστης πρέπει να υπογράψει το transaction. Αυτό γίνεται σε κάθε transaction που πραγματοποιεί ο χρήστης και φαίνεται στο σχήμα 4.8.



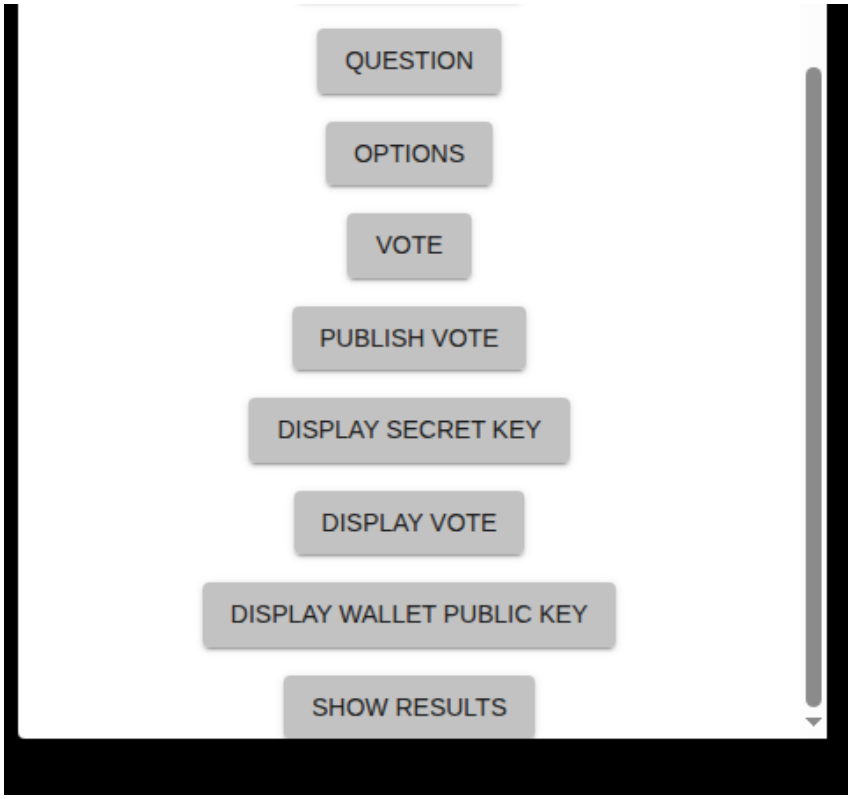
Σχήμα 4.8: Η υπογραφή ενός transaction με το lace wallet

Έχοντας δημιουργήσει την ψηφοφορία και επιστρέφοντας στην αρχική οθόνη, αν ο χρήστης επιλέξει την επιλογή VIEW VOTINGS τότε μπορεί να δει αυτή την ψηφοφορία, η οποία δεν έχει κάποια ερώτηση, και για αυτό αναγράφεται η ένδειξη "No question yet". Αυτό φαίνεται στο σχήμα 4.9.



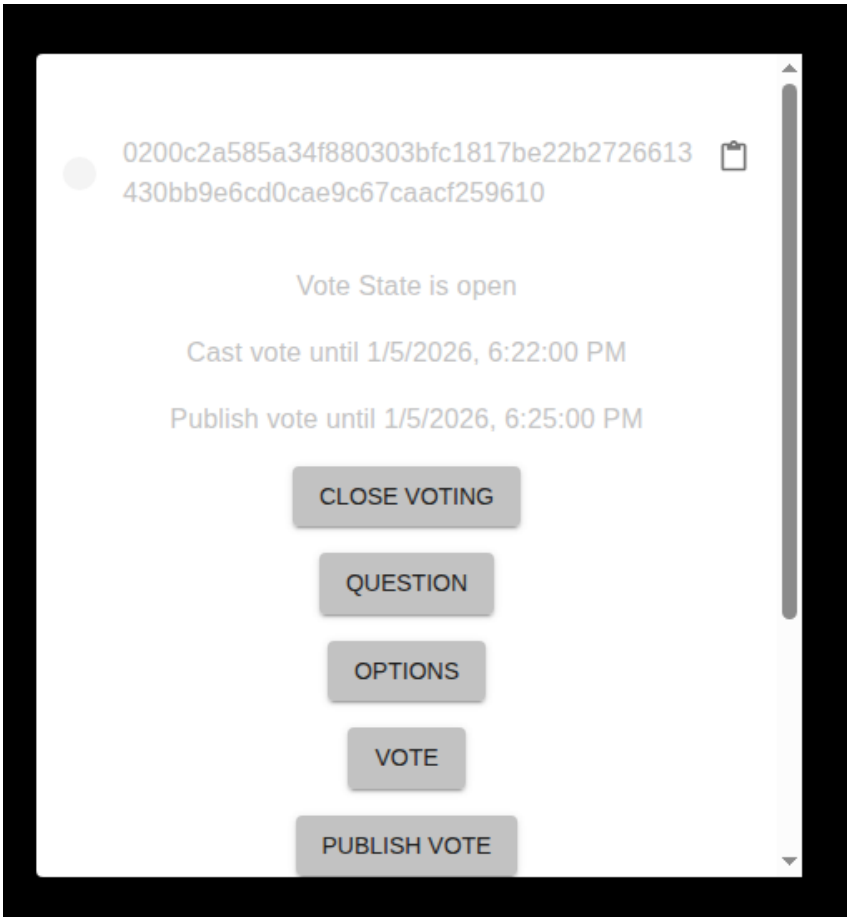
Σχήμα 4.9: Η αρχικά άδεια δημιουργημένη ψηφοφορία

Επιλέγοντας τη συγκεκριμένη ψηφοφορία, ο χρήστης οδηγείται στο μενού της, όπου υπάρχει η ακόλουθη εικόνα.



Σχήμα 4.10: Η αρχικά άδεια δημιουργημένη ψηφοφορία

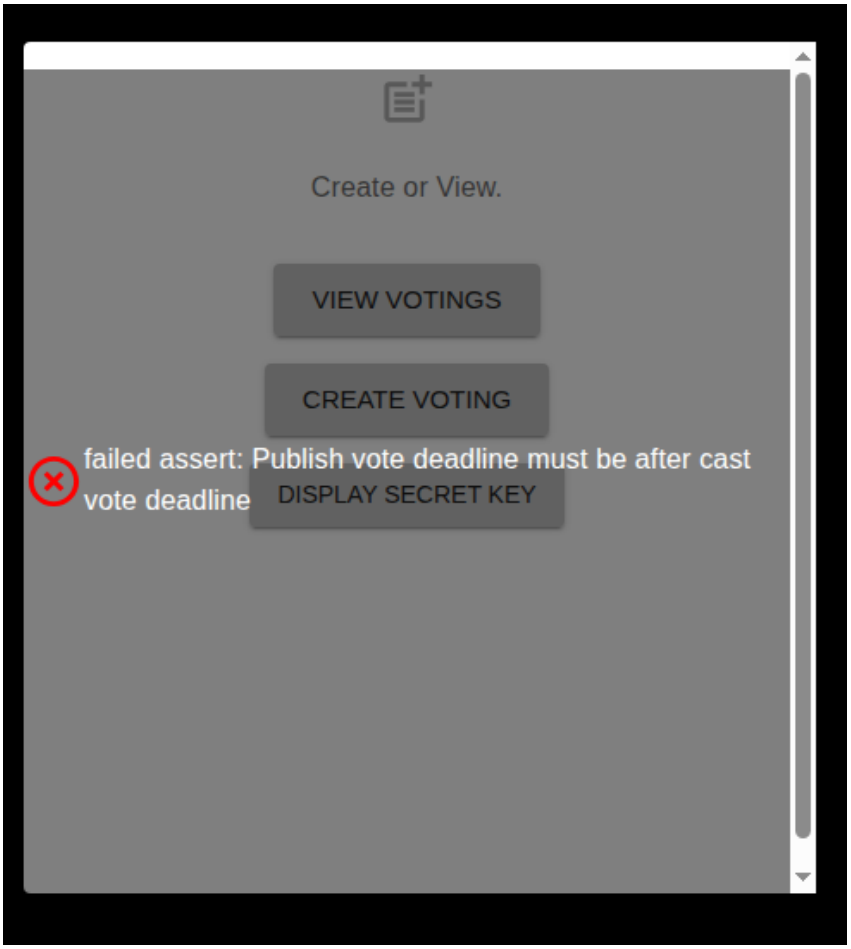
Ο χρήστης μπορεί να ανοίξει ή να κλείσει την ψηφοφορία ανάλογα με την κατάσταση της, μπορεί να προσθέσει την ερώτηση της ψηφοφορίας, να προσθέσει επιλογές, να ψηφίσει, να δημοσιεύσει την ψήφο του, να δει τα αποτελέσματα της ψηφοφορίας (τα οποία όπως έχουμε εγηγήσει εμφανίζονται μετά την λήξη της έχοντας εκτελέσει publish_vote) και να δει την ψήφο που έχει επιλέξει για τη συγκεκριμένη ψηφοφορία.
Επίσης ο χρήστης βλέπει στο μενού της ψηφοφορίας τις ημερομηνίες λήξης υποβολής και δημοσίευσης ψήφου.



Σχήμα 4.11: Οι ημερομηνίες λήξης υποβολής και δημοσίευσης ψήφου

4.2.3 Κακόβουλη χρήση - Ημερομηνία δημοσίευσης πριν την ημερομηνία υποβολής

Αν ο χρήστης που δημιουργεί την ψηφορορία προσπαθήσει να ορίσει την ημερομηνία δημοσίευσης ψήφου να είναι προγενέστερη της ημερομηνίας υποβολής ψήφου τότε θα εμφανιστεί αντίστοιχο μήνυμα λάθους προερχόμενο από το smart contract.



Σχήμα 4.12: Ημερομηνία δημοσίευσης πριν την ημερομηνία υποβολής

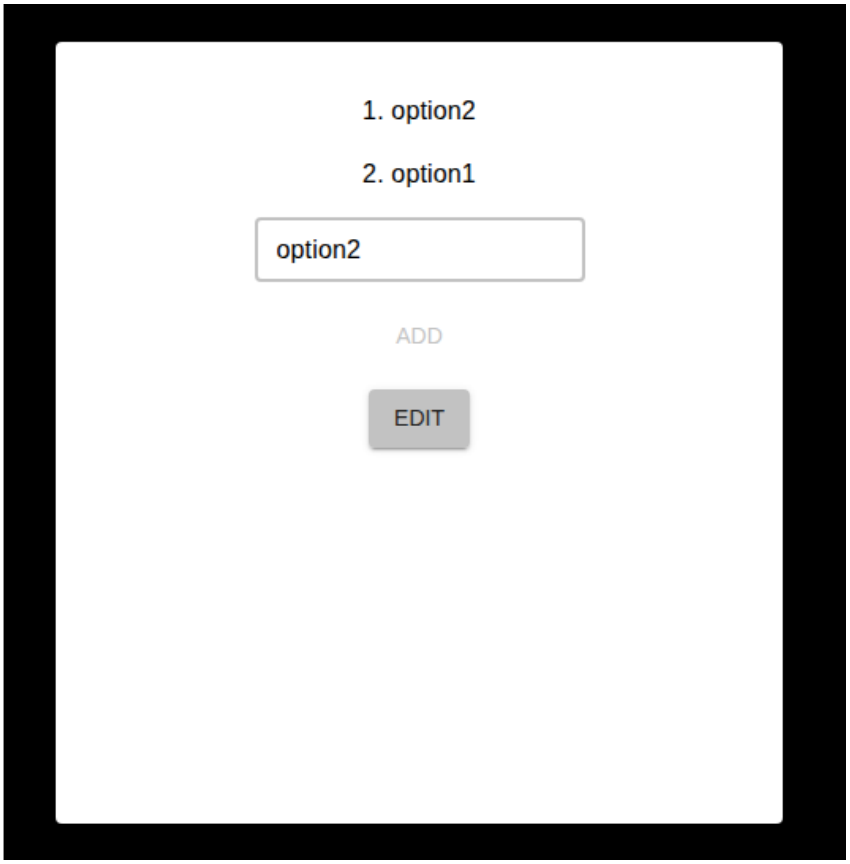
4.2.4 Προσθήκη ερώτησης

Ο χρήστης επιλέγει το κουμπί QUESTION οπότε οδηγείται στο μενού επεξεργασίας της ερώτησης όπως φαίνεται στο σχήμα 4.13. Ας πούμε πως η ερώτηση είναι η question1.

Σχήμα 4.13: Ο χρήστης user1 δημιουργεί την ερώτηση question1

4.2.5 Προσθήκη επιλογών

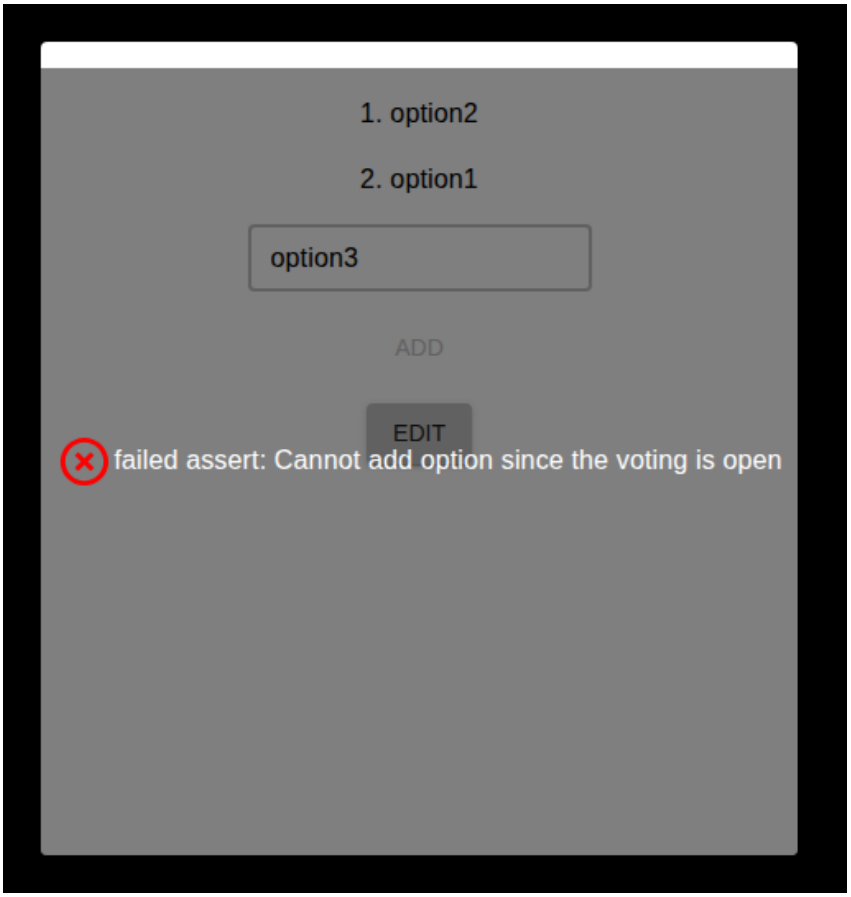
Με την ίδια λογική ο χρήστης προσθέτει στην ψηφοφορία δύο διαθέσιμες επιλογές, τις option1 και opption2.



Σχήμα 4.14: Ο χρήστης user1 δημιουργεί τις επιλογές option1, option2

4.2.6 Κακόβουλη χρήση - Προσπάθεια επεξεργασίας της ψηφοφορίας αφού αυτή γίνει OPEN

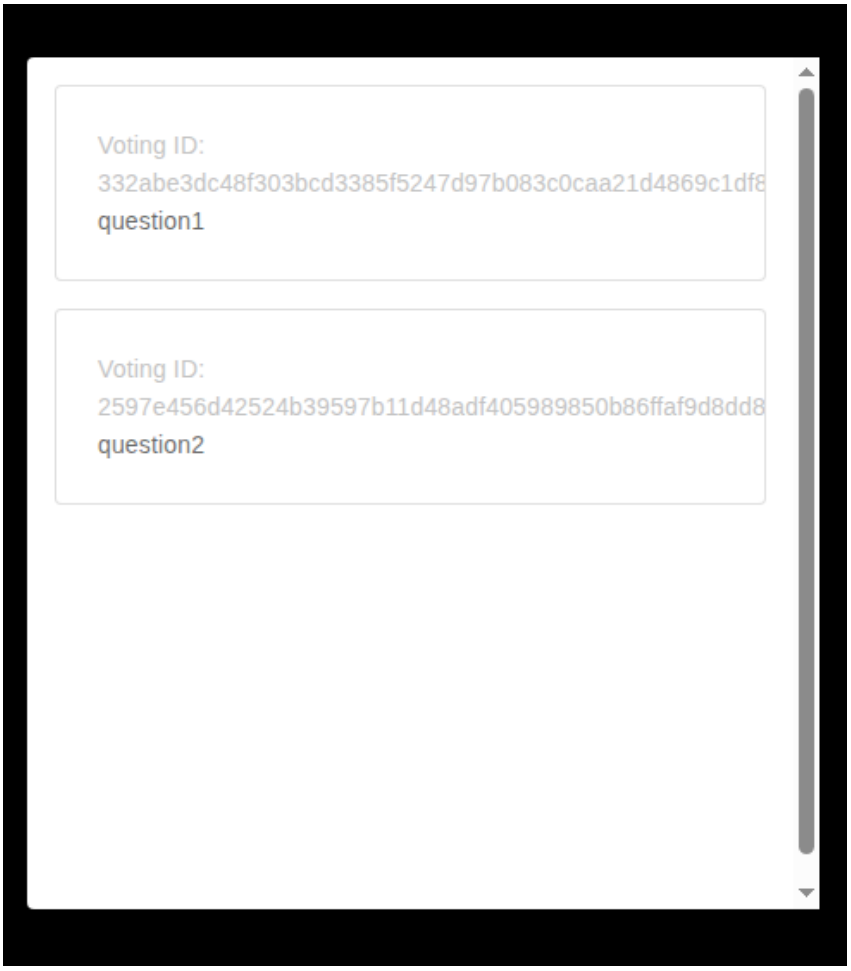
Αν ο χρήστης που δημιούργησε την ψηφοφορία προσπαθήσει να την αλλάξει προσθέτοντας για παράδειγμα μια νέα επιλογή , ενώ η ψηφοφορία έχει μεταβεί σε κατάσταση OPEN θα δει αντίστοιχο μήνυμα λάθους.



Σχήμα 4.15: Ο χρήστης προσπαθεί να προσθέσει μια επιλογή ενώ η ψηφοφορία είναι σε OPEN κατάστηαση

4.2.7 Τελική εικόνα

Τώρα αφού τελειώσαμε με την δημιουργία ψηφοφορίας από τον χρήστη user1, όπως είπαμε προηγουμένως υπάρχει και ο χρήστης user2. Με ακριβώς παρόμοια βήματα με τα προηγούμενα, ο user2 δημιουργεί μια νέα ψηφοφορία με ερώτηση την question2 και επιλογές τις option11, option22. Στο τέλος, η εικόνα που υπάρχει στο smart contract είναι 2 ανοικτές ψηφοφορίες, και αυτό φαίνεται στην αρχική οθόνη, επιλέγοντας το κουμπί VIEW VOTINGS. Προκειμένου οι ψηφοφορίες να είναι έτοιμες να δεχθούν ψήφους, οι δύο organizers εκτελούν το transaction OPEN VOTING ώστε οι ψηφοφορίες να μεταβούν σε κατάσταση OPEN.

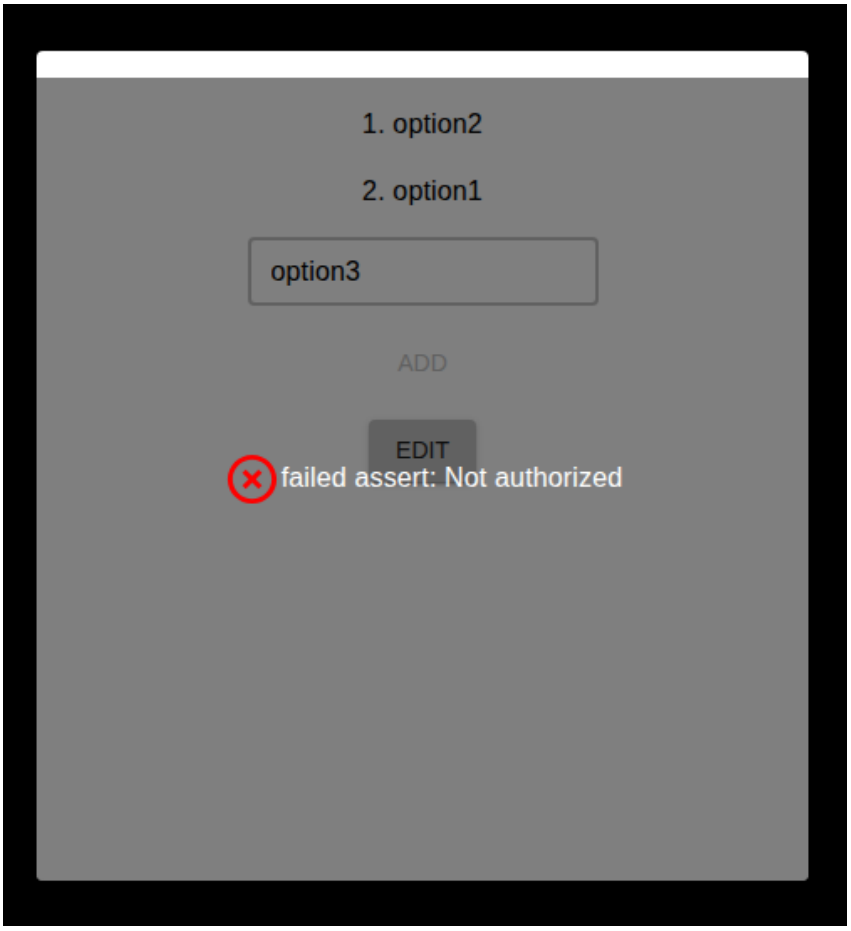


Σχήμα 4.16: Η εικόνα του smart contract με δύο ανοικτές ψηφοφορίες

4.2.8 Κακόβουλη χρήση - Προσπάθεια επεξεργασίας ψηφοφορίας που δεν ανήκει στον χρήστη ή από μη έγκυρο χρήστη

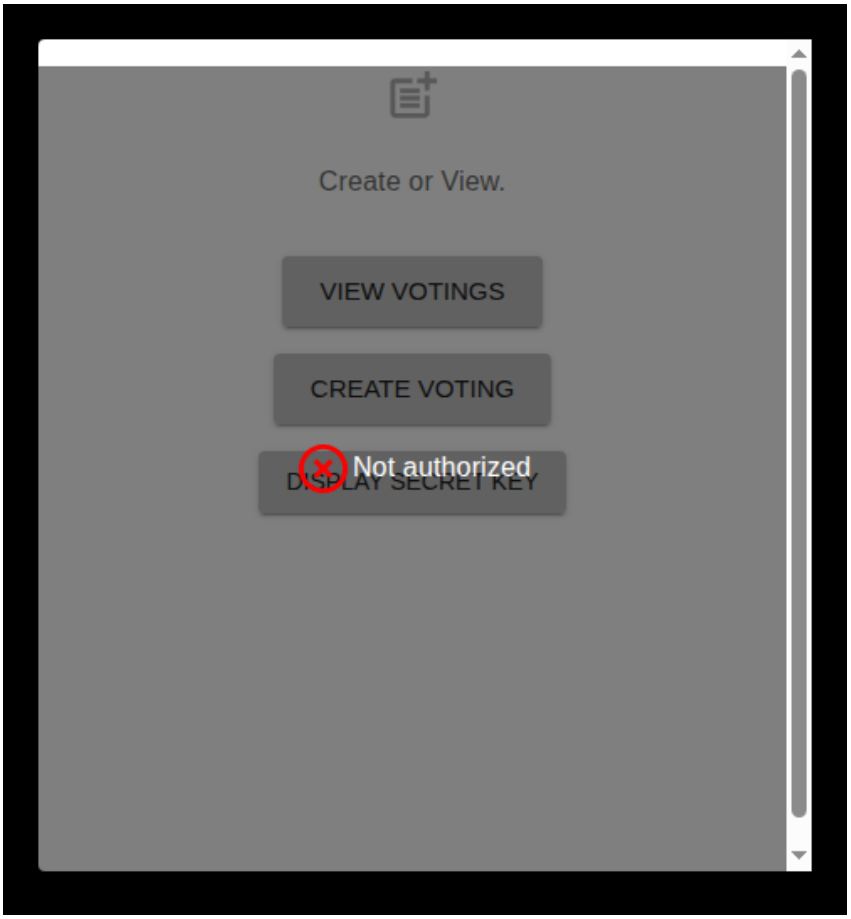
Για να δείξουμε και μερικές περιπτώσεις κακόυβουλης χρήσης, ας υποθέσουμε ότι ο user1 θέλει να τροποποιήσει την ψηφοφορία που δημιούργησε ο user1 προσθέτοντας μια επιλογή, και ας υποθέσουμε επίσης ότι ο user2 Θέλει να κάνει το αντίστοιχο στην ψηφοφορία του user1, τροποποιώντας την ερώτηση. Και στις δύο περιπτώσεις θα εμφανιστεί μήνυμα λάθους, το οποίο προέρχεται από τα assert statements του smart contract, όπως φαίνεται στα δύο επόμενα σχήματα.

Σχήμα 4.17: Ο user1 προσπαθεί να αλλάξει την ερώτηση της ψηφοφορίας που ανήκει στον user2



Σχήμα 4.18: Ο user2 προσπαθεί να προσθέσει μια επιλογή στην ψηφοφορία που ανήκει στον user1

Μια άλλη περίπτωση κακόβουλης χρήσης είναι ένας μη έγκυρος χρήστης, δηλαδή ένας χρήστης του οποίου το hash(secret key) δεν ανήκει στο στο Merkle Tree με τους έγκυρους ψηφοφόρους, να προσπαθήσει να δημιουργήσει μια ψηφοφορία εκτελώντας το transcation create_voting. Όπως έχουμε δει από την υλοποίηση του σχετικού circuit, θα αποτύχει ο έλεγχος prove_eligibility και θα εμφανιστεί μήνυμα λάθους όπως φαίνεται παρακάτω.



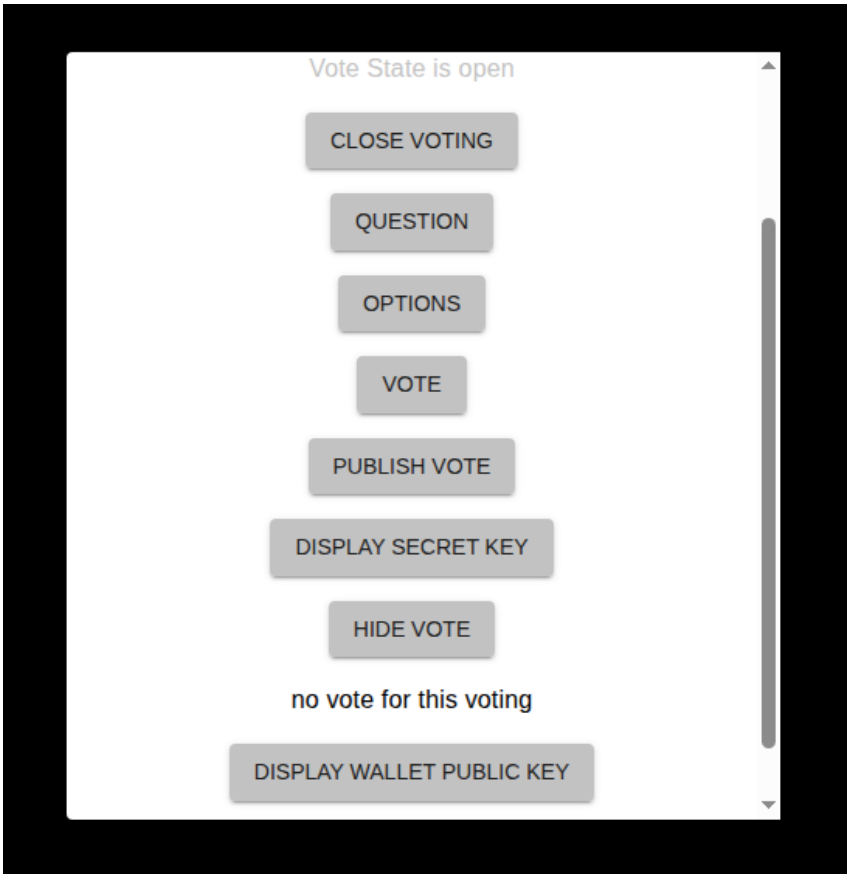
Σχήμα 4.19: Μη έγκυρος χρήστης προσπαθεί να δημιουργήσει μια ψηφοφορία

4.3 Ψήφος σε Ψηφοφορία

Σε αυτή την ενότητα θα συνεχίσουμε με το παράδειγμα που ξεκινήσαμε προηγουμένως και θα δούμε τους δύο χρήστες να ψηφίζουν σε ψηφοφορίες, αλλά και το πως η εφαρμοφή απαγορεύει το double-voting με κατάλληλα μηνύματα λάθους.

4.3.1 Φυσιολογική ροή

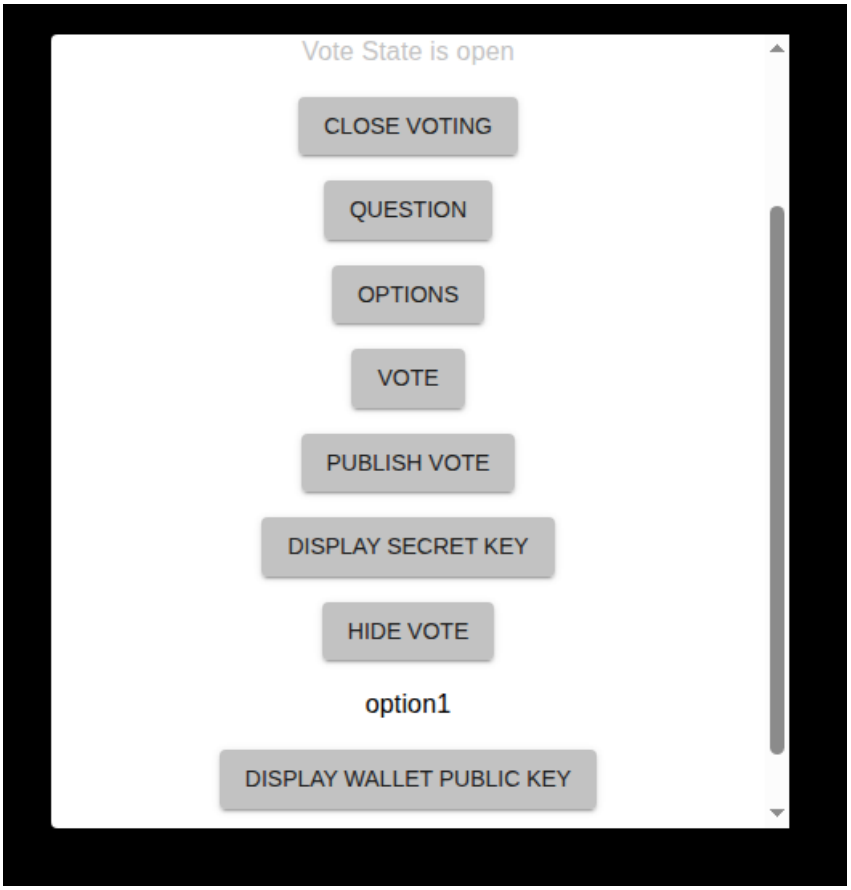
Αρχικά, ο χρήστης user1 ψηφίζει την επιλογή option1 στην ψηφοφορία που δημιούργησε. Ωστόσο, πριν το κάνει αυτό, βλέπουμε πως επιλέγοντας την επιλογή DISPLAY VOTE στο μενού της ψηφοφορίας, φαίνεται πως δεν υπάρχει καταχωρημένη κάποια ψήφος, βλ. σχήμα 4.20



Σχήμα 4.20: Ο user1 πριν ψηφίσει στην ψηφοφορία που δημιούργησε

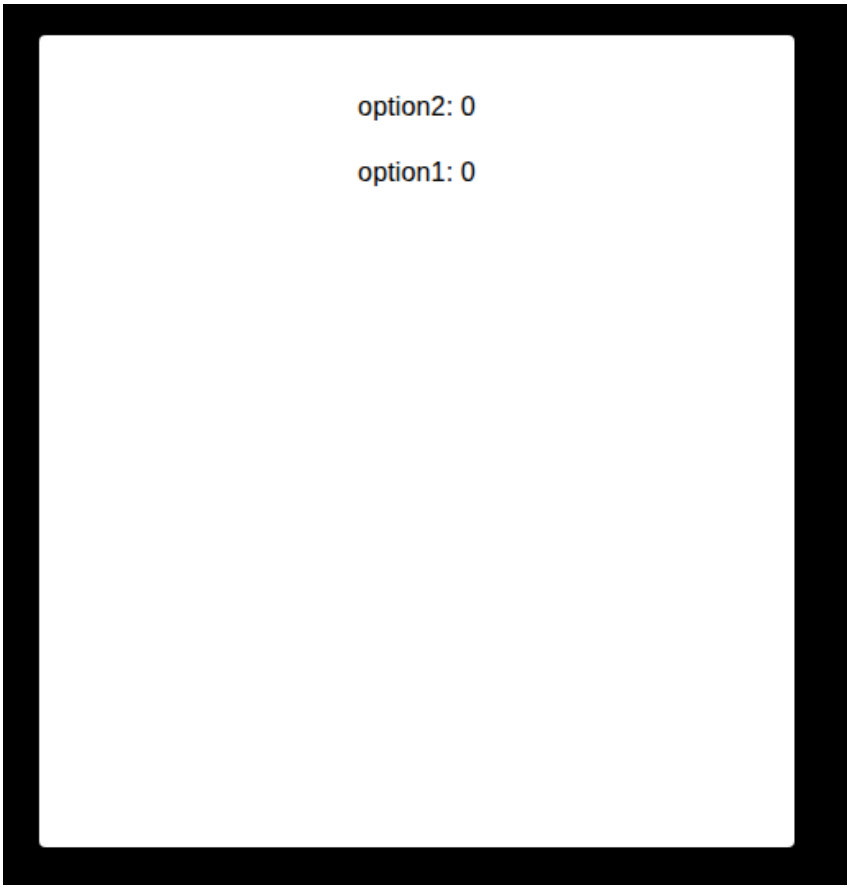
Στη συνέχεια επιλέγει την επιλογή VOTE και ψηφίζει το option1 και τώρα στο μενού της ψηφοφορίας, στην επιλογή DISPLAY VOTE, θα φαίνεται η επιλογή option1.

Σχήμα 4.21: Ο user1 ψηφίζει το option1 στην ψηφοφορία που δημιούργησε



Σχήμα 4.22: Ο user1 έχοντας ψηφίσει option1 στην ψηφοφορία που δημιούργησε

Όπως είπαμε προηγουμένως, επειδή στο ledger καταχωρείται το hash της ψήφου και όχι η ίδια η ψήφος στο στάδιο αυτό, αν ο χρήστης επιλέξει την επιλογή SHOW RESULTS τότε δε θα φαίνεται ότι υπάρχουν καταχωρημένα αποτελέσματα, πράγμα που επιβεβαιώνεται παρακάτω:

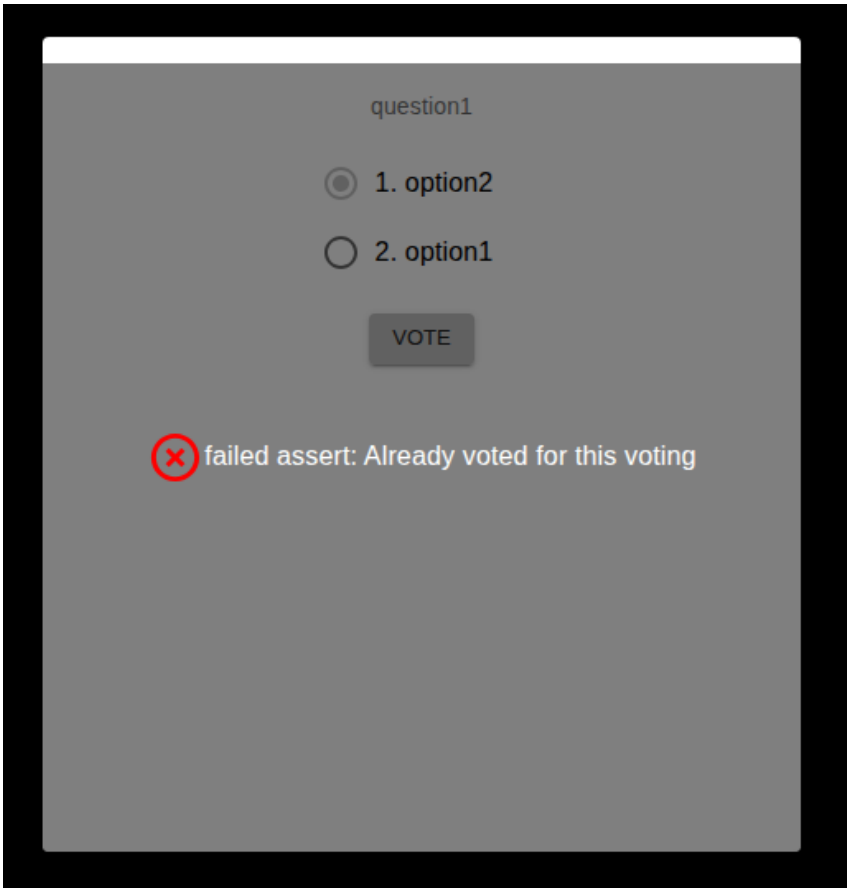


Σχήμα 4.23: Τα αποτελέσματα δε φαίνονται όσο είναι ανοιχτή η ψηφοφορία και δεν έχουν γίνει published οι ψήφοι

Συμμετρικά τώρα, ο χρήστης user2 με ίδια βήματα ψηφίζει την επιλογή option2 στην ψηφοφορία του user1(question1). Επίσης, ο user1 ψηφίζει την επιλογή option11 στην ψηφοφορία του user2(question2) και ο user2 ψηφίζει την επιλογή option22. Για χάριν απλότητας παραλείπουμε τα ενδιάμεσα βήματα.

4.3.2 Κακόβουλη χρήση - double-voting

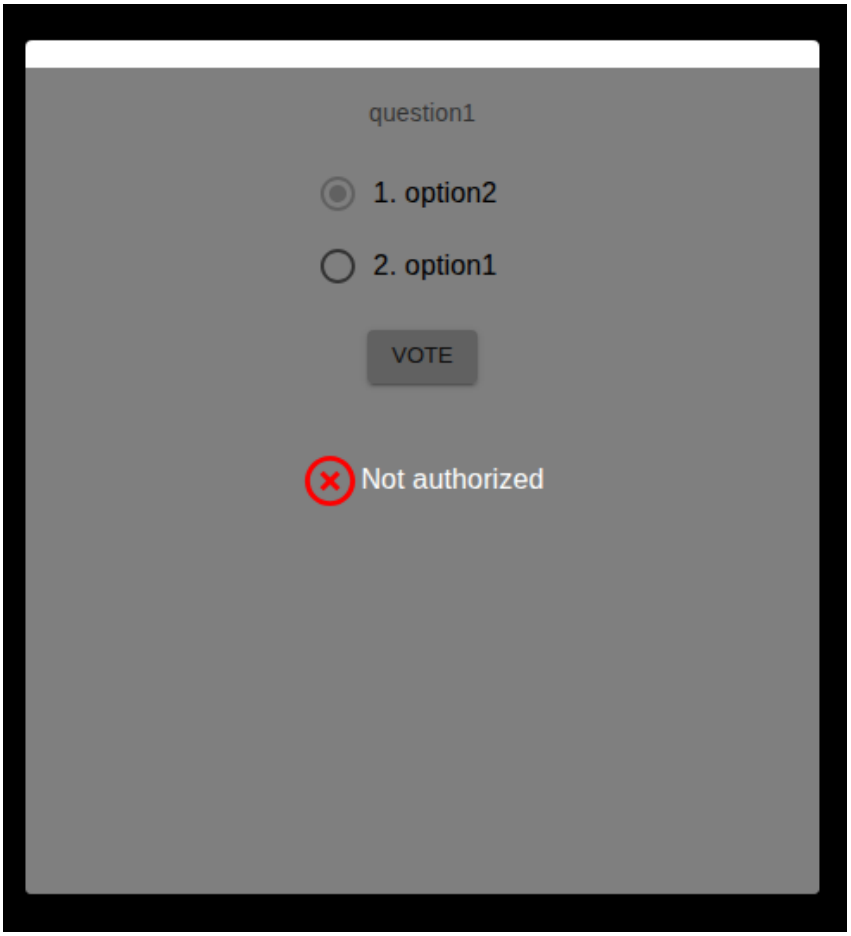
Έστω ότο ο χρήστης user1 επιλέγει να ξαναψηφίσει στην ίδια ψηφοφορία. Αν το πραγματοποιήσει αυτό, τότε ευμφανίζεται κατάλληλο μήνυμα λάθους που προέρχεται από το assert statement του cast_vote circuit στο smart contract, το οποίο έχει αναλυτικά περιγραφεί παραπάνω.



Σχήμα 4.24: Περίπτωση double-voting

4.3.3 Κακόβουλη χρήση - Μη έγκυρος χρήστης

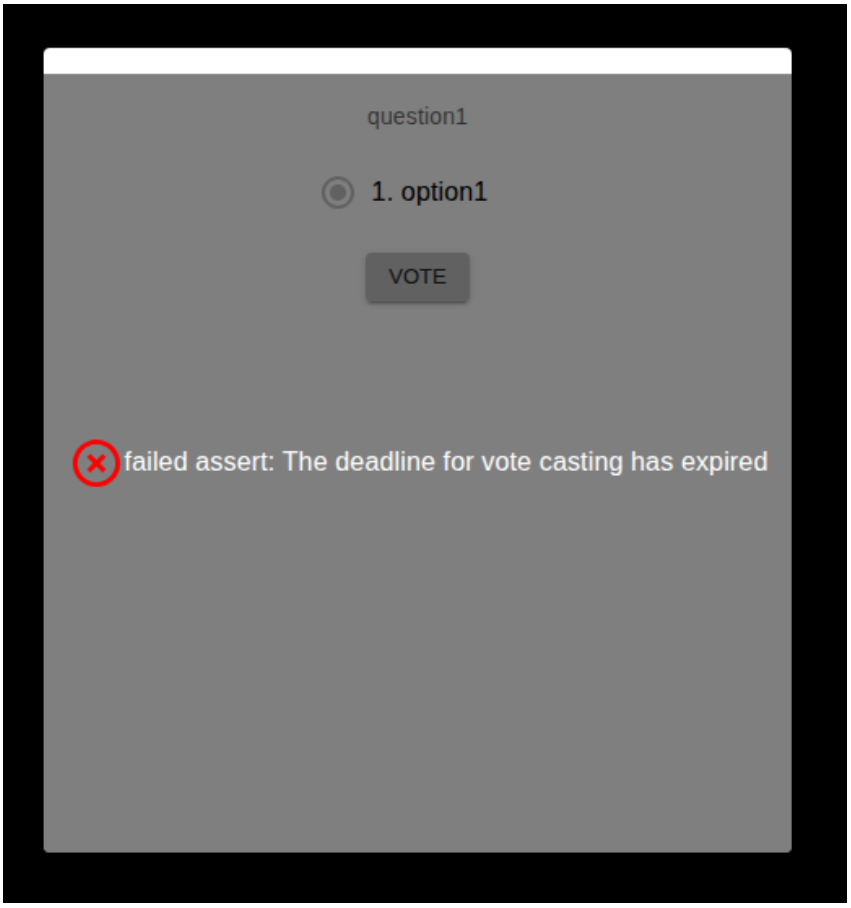
Ας υποθέσουμε ότι έχουμε έναν χρήστη ο του οποίου το public key δεν ανήκει στο Merkle Tree Με τους έγκυρους ψηφοφόρους. Αν αυτός ο χρήστης προσπαθήσει να ψηφίσει σε οποιαδήποτε ψηφοφορία, ακριβώς επειδή θα αποτύχει να αποδείξει ότι γνωρίζει έγκυρο μονοπάτι προς τη ρίζα του δέντρου, δεν έχει δικαίωμα να ψηφίσει και εμφανίζεται κατάλληλο μήνυμα λάθους από την εφαρμογή.



Σχήμα 4.25: Μη έγκυρος χρήστης προσπαθεί να ψηφίσει

4.3.4 Κακόβουλη χρήση - Υποβολή ψήφου μετά την ημερομηνία λήξης

Σε περίπτωση που ο χρήστης υποβάλλει την ψήφο μετά την ημερομηνία λήξης υποβολής ψήφου για τη συγκεκριμένη ψηφοφορία τότε εμφανίζεται κατάλληλο μήνυμα λάθους το οποίο προέρχεται από το smart contract όπως έχουμε δει.



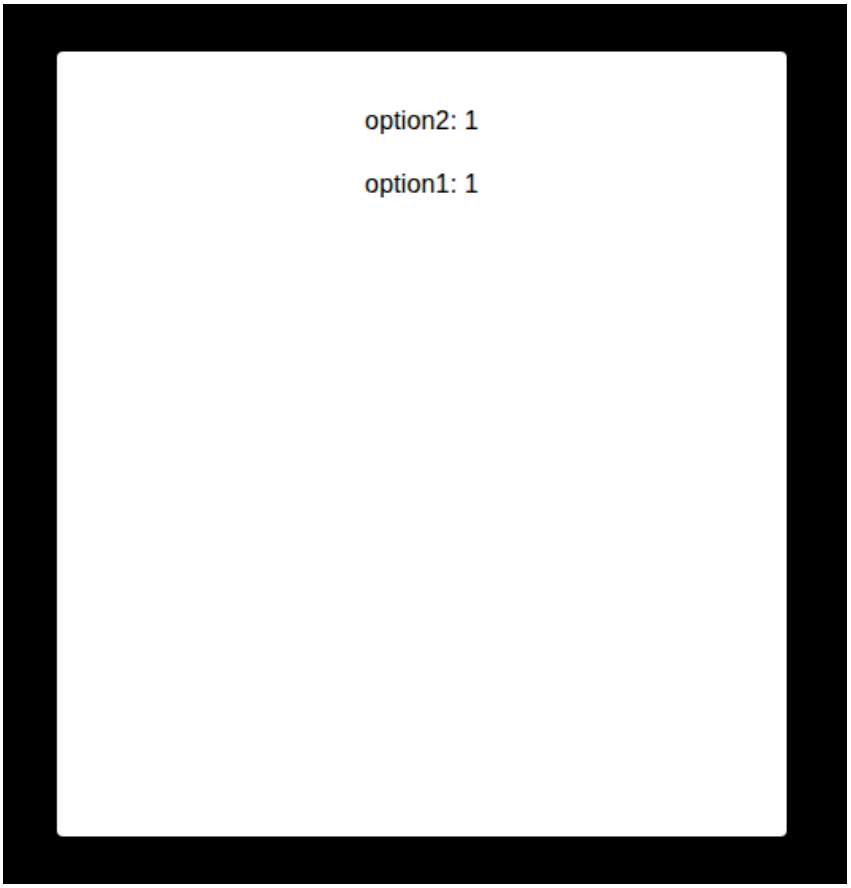
Σχήμα 4.26: Χρήστης ψηφίζει μετά το χρονικό deadline

4.4 Δημοσίευση ψήφου

Σε αυτή την ενότητα θα συνεχίσουμε το παράδειγμα που έχουμε ήδη ξεκινήσει ώστε να ολοκληρώσουμε έναν πλήρη κύκλο χρήστης της εφαρμογής, αξιοποιώντας όλες τις δυνατότητές της.

4.4.1 Φυσιολογική ροή

Έχοντας λοιπόν και οι δύο χρήστες user1 και user2 ψηφίσει και έχοντας περάσει το deadline για υποβολή ψήφου, ήρθε η ώρα για τη δημοσίευση ψήφων. Ο user1 εκτελεί το transaction PUBLISH VOTE. Αντίστοιχα ο user2 εκτελεί και αυτός PUBLISH VOTE, συνεπώς η εικόνα στην επιλογή SHOW RESULTS είναι αυτή που παρουσιάζεται στο σχήμα 4.27:



Σχήμα 4.27: Τα αποτελέσματα στην ψηφοφορία μετά την δημοσίευση και των δύο ψήφων

Αντίστοιχη είναι και η εικόνα στην ψηφοφορία που δημιούργησε ο user2 όταν και οι δύο χρήστες δημοσιεύσουν τις ψήφους τους,

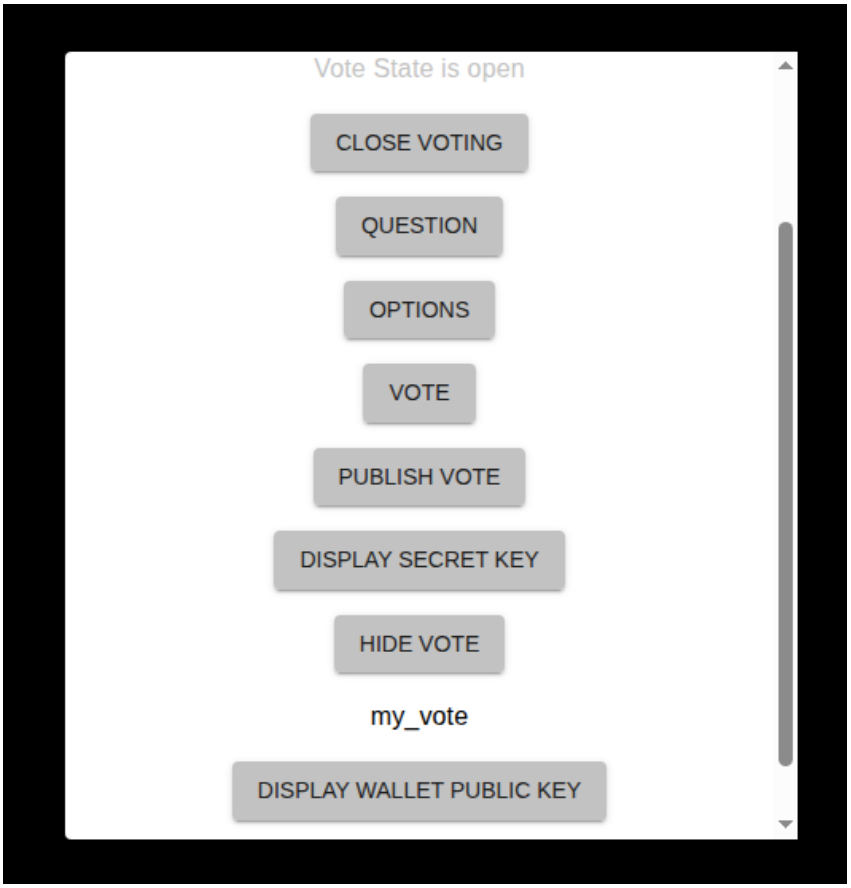
4.4.2 Κακόβουλη χρήση - Δεύτερη δημοσίευση ψήφου

Αν ο χρήστης user1 επιλέξει να εκτελέσει δεύτερη φορά το transcation publish_vote τότε θα εμφανιστεί το αντίστοιχο μήνυμα λάθους που υπάρχει στο circuit. 4.28:

Σχήμα 4.28: Ο χρήστης user1 προσπαθεί δεύτερη φορά να δημοσιεύσει την ψήφο του

4.4.3 Κακόβουλη χρήση - Μη έγκυρη ψήφος

Έστω ότι έχουμε έναν χρήστη, ο οποίος αλλάζει την υλοποίηση της εφαρμογής, και θέτει στην private state του, τη μεταβλητή secret_vote(voting_id) να είναι μια ψήφος της επιλογής του, έστω my_vote. Όπως έχουμε ήδη αναφέρει, η private state ανήκει στον χρήστη και δεν εκτίθεται ποτέ έξω από το τοπικό του περιβάλλον, συνεπώς μπορεί να την τροποποιήσει όπως θέλει.



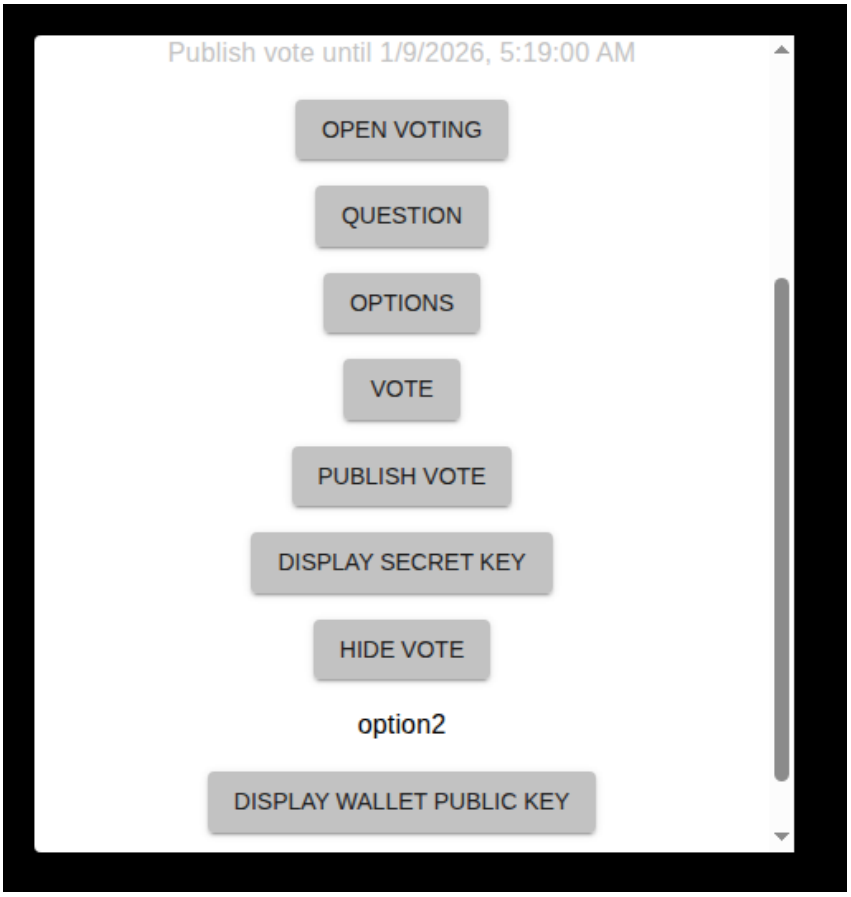
Σχήμα 4.29: Ο χρήστης ρίχνει μια δική του ψήφο που δεν υπάρχει στις διαθέσιμες επιλογές

Το transaction cast_vote θα πετύχει κανονικά και στη ledger μεταβλητή hashed_votes Θα καταχωρηθεί το hash της συμβολοσειράς my_vote. Όταν όμως ο χρήστης εκτελέσει το transaction publish_vote, θα ελεγθεί ότι η ψήφος αυτή δεν ανήκει στο σύνολο voting_options οπότε και θα εμφανιστεί μήνυμα λάθους που προέρχεται από το circuit publish_vote. Αξίζει να σημειωθεί εδώ, ότι δεν είναι δυνατή η συσχέτιση της ψήφου με τον ψηφοφόρο καθώς η χρήση Merkle Trees αποτρέπει τη σύνδεση του public key με το transaction που εκτελείται.

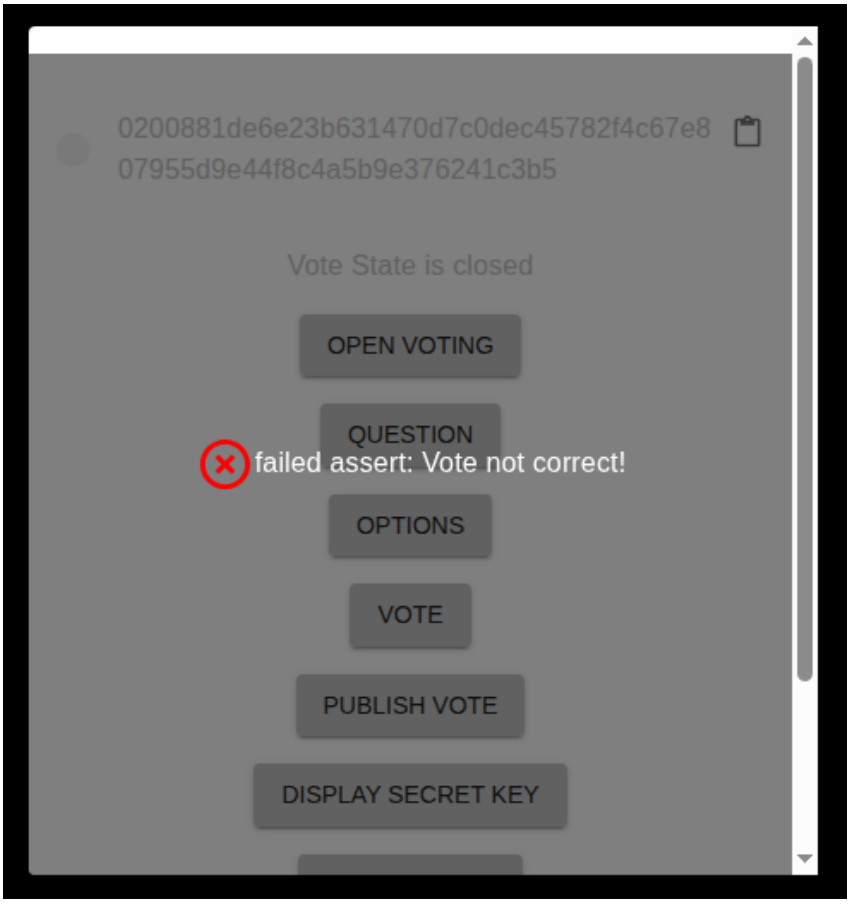
Σχήμα 4.30: Ο χρήστης προσπαθεί να δημοσιεύσει μη έγκυρη ψήφο

4.4.4 Κακόβουλη χρήση - Αλλαγή ψήφου

Παρόμοια με το προηγούμενο παράδειγμα, έστω ότι έχουμε ένα χρήστη ο οποίος αρχικά επιλέγει κανονικά μία από τις διαθέσιμες επιλογές και εκτελεί το cast_vote transaction. Το hash της έγκυρης ψήφου θα καταχωρηθεί ως γνωστόν στο hashed_votes. Έστω τώρα ότι ο κακόβουλος χρήστης αλλάζει το private state του και θέτει στο secret_vote κάποια άλλη επιλογή, για παράδειγμα την option2. Αν εκτελέσει το transaction publish_vote για να δημοσιεύσει την ψήφο του, τότε τα δύο hashes δε θα ταιριάζουν (δηλαδή το hash της αρχικής ψήφου και το hash της αλλαγμένης ψήφου), συνεπώς θα εμφανιστεί μήνυμα λάθους που έρχεται από το αντίστοιχο circuit.



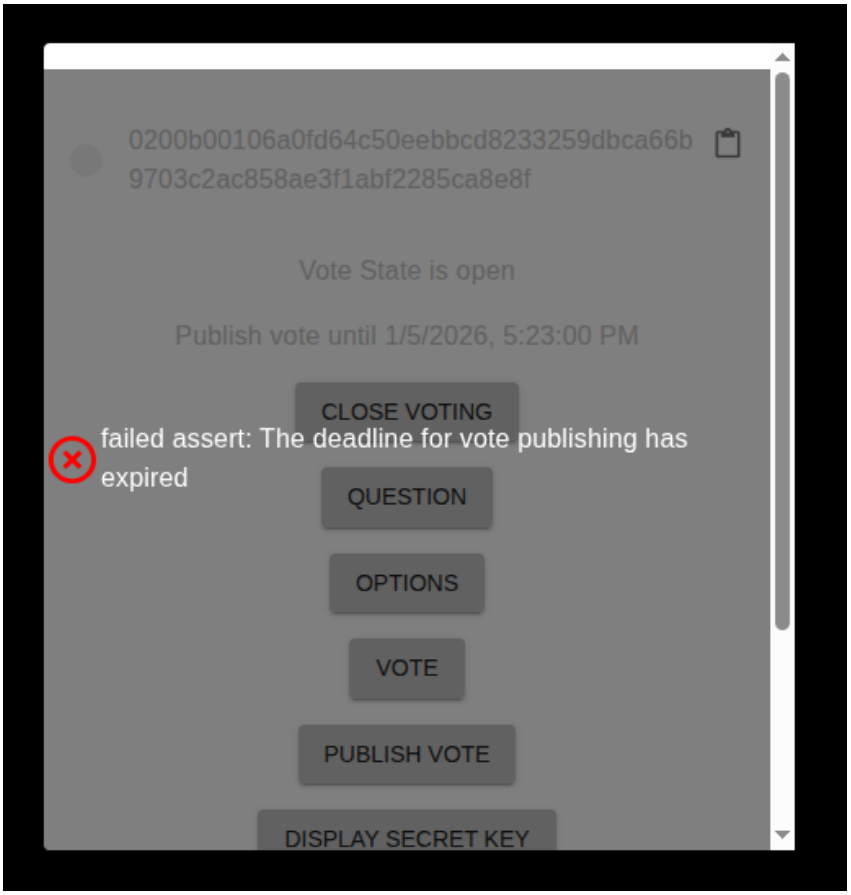
Σχήμα 4.31: Ο χρήστης άλλαξε την ψήφο του μετά την υποβολή του cast_vote



Σχήμα 4.32: Ο χρήστης άλλαξε το private state της ψήφου και προσπάθησε να δημοσιεύσει

4.4.5 Κακόβουλη χρήση - Δημοσίευση ψήφου μετά την ημερομηνία λήξης

Σε περίπτωση που ο χρήστης δημοσιεύσει την ψήφο μετά την ημερομηνία λήξης δημοσίευσης ψήφου για τη συγκεκριμένη ψηφοφορία τότε εμφανίζεται κατάλληλο μήνυμα λάθους το οποίο προέρχεται από το smart contract όπως έχουμε δει.



Σχήμα 4.33: Χρήστης δημοσιεύει την ψήφο μετά το χρονικό deadline

Κεφάλαιο 5

Επίλογος

Στο πλαίσιο της παρούσας διπλωματικής εργασίας καταδείξαμε πώς η τεχνολογία blockchain μπορεί να αξιοποιηθεί για την υλοποίηση μιας πραγματικά αποκεντρωμένης εφαρμογής ηλεκτρονικών ψηφοφοριών για φοιτητές. Παράλληλα, αναδείξαμε τα ουσιαστικά πλεονεκτήματα που προσφέρει το Midnight σε σχέση με τα παραδοσιακά blockchain συστήματα, εστιάζοντας ιδιαίτερα στους μηχανισμούς προστασίας των προσωπικών δεδομένων. Με τον τρόπο αυτό αναπτύξαμε μια εφαρμογή στην οποία διασφαλίζεται τόσο η ιδιωτικότητα της ψήφου όσο και η ανωνυμία και ακεραιότητα της ταυτότητας κάθε χρήστη, προσφέροντας ένα περιβάλλον που συνδυάζει ασφάλεια, διαφάνεια και εμπιστοσύνη — στοιχεία απαραίτητα για μια σύγχρονη και αξιόπιστη ψηφιακή διαδικασία ψηφοφορίας.
Ωστόσο, δεν εξαλείψαμε πλήρως την ανάγκη ύπαρξης μιας κεντρικής αρχής, καθώς το πανεπιστήμιο εξακολουθεί να αποτελεί τον φορέα που αναλαμβάνει την αυθεντικοποίηση των φοιτητών και τη δημοσίευση του smart contract στο δίκτυο. Με άλλα λόγια, παρότι το σύστημα βασίζεται σε αποκεντρωμένες τεχνολογίες, απαιτείται ένας έμπιστος οργανισμός για τη διασφάλιση της εγκυρότητας και της αρχικής εγκαθίδρυσης των διαδικασιών. Ωστόσο, εξαιτίας της χρήστης δένδρων Merkle καταφέρνουμε να απορρίψουμε τη σύνδεση μεταξύ της ταυτότητας ενός ψηφοφόρου και της ψήφου του. Ακόμα κι αν διαρρεύσουν στοιχεία από τη βάση δεδομένων του πανεπιστημίου και γίνουν φανερά τα public keys, δεν είναι δυνατή η σύνδεσή τους με τις ατομικές ψήφους. Επίσης ένα ακόμα πλεονέκτημα της υλοποίησής μας είναι πως έχοντας δύο περιόδους ψηφοφορίας, την περίοδο υποβολής ψήφων και την περίοδο δημοσίευσης ψήφων, αποκρύπτουμε τα αποτελέσματα και αυτά γίνονται ορατά μόνο κατά το στάδιο δημοσίευσης. Έτσι, όταν ένας φοιτητής δοκιμάσει να ψηφίσει, η μόνη δημόσια μεταβλητή είναι ένα σύνολο από hashes το οποίο δεν δίνει καμία πληροφορία για τα αποτελέσματα. Έτσι, η ψήφος των φοιτητών δεν επηρρεάζεται από την πορεία της ψηφοφορίας.
Κατά τη διάρκεια της μελέτης και της υλοποίησης της διπλωματικής εργασίας αντιμετωπίσαμε αρκετές προκλήσεις και δυσκολίες, ενώ σε ορισμένα σημεία χρειάστηκε να προβούμε σε συγκεκριμένες παραδοχές για να προχωρήσει ο σχεδιασμός και η ανάπτυξη του συστήματος. Παρ' όλα αυτά, το τελικό αποτέλεσμα κρίνεται ιδιαίτερα ικανοποιητικό, καθώς ανταποκρίνεται στους στόχους που είχαν τεθεί αρχικά και προσφέρει μια ολοκληρωμένη και λειτουργική λύση.

5.1 Τεχνικές Προκλήσεις

Ένα από τα μεγαλύτερα ζητήματα ήταν η αρχική έλλειψη γνώσης σχετικά με τη λειτουργία των αποκεντρωμένων εφαρμογών και του Midnight. Συγκεκριμένα, έπρεπε να κατανοήσουμε πώς λειτουργεί η γλώσσα Compact, πώς συνδέεται με την TypeScript, αλλά και το μοντέλο διαχείρισης κατάστασης του Midnight, το οποίο διαχωρίζει ξεκάθαρα την private state από την public state. Ήταν απαραίτητο να μελετήσουμε σε βάθος την τεκμηρίωση και να συζητήσουμε με ανθρώπους που εργάζονται στον χώρο, ώστε να κατανοήσουμε καλύτερα τον τρόπο με τον

οποίο λειτουργεί ολόκληρο το οικοσύστημα. Μέσα από αυτή τη διαδικασία αποκτήσαμε σαφή κατανόηση των μηχανισμών που διέπουν τη λειτουργία του Midnight και καταφέραμε να προχωρήσουμε στην ανάπτυξη των απαιτούμενων λειτουργιών της εφαρμογής.

5.2 Περιορισμοί - Παραδοχές

Όπως αναφέραμε, λόγω της δυσκολίας υλοποίησης αλλά και των ιδιαιτεροτήτων της τεχνολογίας του Midnight, αναγκαστήκαμε να προβούμε σε μερικές παραδοχές για τη λειτουργία - χρήση της εφαρμογής.

- Όλοι οι φοιτητές μπορούν να συμμετέχουν σε όλες τις διαθέσιμες ψηφοφορίες.
- Το πανεπιστήμιο θεωρείται έμπιστη αρχή και στη βάση δεδομένων του καταχωρείται για κάθε φοιτητή το public key του, δηλαδή το Hash του secret key του. Η αποθήκευση στοιχείων στη βάση δεδομένων αυξάνει το βαθμό κεντρικοποίησης του συστήματος. Παρ' όλα αυτά, όπως αναλύσαμε προηγουμένως, ακόμα κι αν αποθηκεύονται τα public keys στη βάση δεδομένων, επειδή η ψηφοφορία γίνεται χρησιμοποιώντας merkle trees, ποτέ δεν είναι δυνατή η σύνδεση ενός ψηφοφόρου με την ψήφο του.
- Υπάρχει ένα smart contract πάνω στο οποίο έχει χτιστεί η εφαρμογή και όπως έχουμε περιγράψει, οι έγκυροι χρήστες καθορίζονται πριν την δημιουργία του smart contract. Συνεπώς, με την ένταξη νέων φοιτητών στο πανεπιστήμιο, θα πρέπει να δημιουργείται πάλι το smart contract από το πανεπιστήμιο ώστε να συμπεριληφθούν και οι νεοεισαχθέντες φοιτητές.
- Αφού ψηφίσει ένας φοιτητές και αφού ο organizer κλείσει μια ψηφοφορία, θεωρούμε ότι ο φοιτητής θα πρέπει να δημοσιεύσει την ψήφο του πριν το πέρας του χρονικού deadline, διαφορετικά αυτή δε θα μετρήσει.
- Αν κάποιος φοιτητής χάσει το secret key του, δεν υπάρχει μηχανισμός ανάκτησης ή χορήγησης κάποιου νέου κλειδιού.

5.3 Μελλοντικές Κατευθύνσεις - Βελτιώσεις

Λόγω των παραδοχών που αναφέρθηκαν προηγουμένως, προτείνουμε λύσεις και επεκτάσεις οι οποίες θα έκανα την εφαρμογή πιο ρεαλιστική και θα της προσέφεραν περισσότερες δυνατότητες.

- Για να υπάρχει μεγαλύτερη ευελιξία, μπορούν οι φοιτητές να λαμβάνουν πέρα από ένα απλό secret key, ένα πιστοποιητικό πως ανήκουν σε κάποιο συγκεκριμένο εξάμηνο ή σε κάποια φοιτητική ομάδα. Με αυτό τον τρόπο, μπορεί να δημιουργούνται ψηφοφορίες μόνο για συγκεκριμένες φοιτητικές ομάδες ή για συγκεκριμένα εξάμηνα.
- Επειδή ακριβώς οι έγκυροι χρήστες καθορίζονται πριν την δημιουργία του smart contract και δεν υπάρχει ευελιξία αν πρέπει να προστεθεί κάποιος νέος χρήστης ή αν κάποιος χρήστης χάσει το secret key του είναι κάθε ψηφορορία να αποτελεί ένα ξεχωριστό smart contract και ο κάθε χρήστης να μπορεί να αυθεντικοποεί τον εαυτό του για αυτή την ψηφοφορία. Δοκιμάσαμε μια τέτοια υλοποίηση, όπου ο κάθε χρήστης όταν δημιουργεί μια ψηφοφορία, ουσιαστικά δημοσιεύει ένα νέο contract, και υπάρχει ένα transaction add_voter το οποίο προσθέτει ένα public key στο merkle tree με τους έγκυρους ψηφοφόρους. Ωστόσο, εγκαταλείψαμε αυτή τη λύση, διότι το add_voter πραγματοποιούταν από το πανεπιστήμιο, κάτι το οποίο του έδινε περισσότερη ισχύ και ένα malicious πανεπιστήμιο θα μπορούσε να προσθέτει όσους και όποιους χρήστες θέλει, παραβιάζοντας

τη διαφάνεια των ψηφοφοριών. Παρ'όλα αυτά μπορούν να εξερευνηθούν μέθοδοι που θα επέτρεπαν μια τέτοια πιο ευέλικτη υλοποίηση και ταυτόχρονα δε θα ενίσχυαν τόσο πολύ τη δύναμη της κεντρικής αρχής.

Αξίζει να σημειωθεί ότι η λογική του συστήματός μας δεν περιορίζεται μόνο στο πανεπιστήμιο. Η ίδια τεχνολογία μπορεί να εφαρμοστεί με την ίδια επιτυχία και σε άλλους τομείς, όπως οι εθνικές εκλογές, οι ψηφοφορίες σε μεγάλες εταιρείες ή ακόμα και οι αποφάσεις σε τοπικούς συλλόγους. Το βασικό πλεονέκτημα παραμένει το ίδιο: όλοι μπορούν να ελέγξουν ότι το αποτέλεσμα είναι σωστό, χωρίς κανείς να μάθει τι ψήφισε ο κάθε πολίτης.

5.4 Github Link

Ο κώδικας της εφαρμογής μπορεί να βρεθεί στον ακόλουθο σύνδεσμο: `https://github.com/JimV4/VoteGuardian2`

Βιβλιογραφία

[1] Ledger Academy. *Proof of Authority (PoA) Meaning*. URL: https://www.ledger.com/academy/glossary/proof-of-authority.

[2] Cardano Docs. *Cardano Docs*. URL: https://docs.cardano.org/stake-pool-operators/about-stake-pools.

[3] Google. *leveldb*. URL: https://github.com/google/leveldb.

[4] Xingjie Yu Huaqun Guo. *A survey on blockchain technology and its security*. 2022. URL: https://www.sciencedirect.com/science/article/pii/S2096720922000070?via%3Dihub.

[5] Midnight. *Midnight Docs*. URL: https://docs.midnight.network/.

[6] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. URL: https://bitcoin.org/bitcoin.pdf.

[7] Midnight Network. *NIGHTPAPER A litepaper introducing Midnight*. URL: https://45047878.fs1.hubspotusercontent-na1.net/hubfs/45047878/Midnight%20litepaper.pdf.

[8] CSLAB NTUA. *Κατανεμημένα Συστήματα, Bitcoin and Blockchain*. 2023. URL: https://helios.ntua.gr/2023-24/pluginfile.php/56269/mod_page/content/30/distributed_10_2023.pdf.

[9] OpenEthereum. *Proof-of-Authority Chains - Wiki*. URL: https://openethereum.github.io/Proof-of-Authority-Chains.

[10] Nick Szabo. *Smart Contracts*. 1994. URL: https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html.

[11] Aggelos Kiayias Thomas Kerber, Markulf Kohlweiss The University of Edinburgh και IOHK. *Kachina – Foundations of Private Smart Contracts*. URL: https://eprint.iacr.org/2020/543.pdf.

[12] ΠΑΝΑΓΙΩΤΗΣ ΓΡΟΝΤΑΣ ΕΥΣΤΑΘΙΟΣ ΖΑΧΟΣ ΑΡΙΣΤΕΙΔΗΣ ΠΑΓΟΥΡΤΖΗΣ. *ΥΠΟΛΟΓΙΣΤΙΚΗ ΚΡΥΠΤΟΓΡΑΦΙΑ*. URL: https://repository.kallipos.gr/bitstream/11419/5439/25/main-KOY.pdf.