Bitvoting

Praktikum: Kryptographie im Sommersemester 2015

Betreuerin: Oksana Kulyk
Studenten: Benedikt Hiemenz, Max Kolhagen, Markus Schmidt

TECHNISCHE
UNIVERSITÄT
DARMSTADT

1 Setup

Die im Rahmen dieses Praktikum erstellte Software *Bitvoting* nutzt verschiedene Bibliotheken, die vor der Kompilierung des Codes installiert werden müssen.

Unter Linux/Ubuntu müssen folgende Pakete installiert werden:

- libleveldb-dev (LevelDB)
- libgmp3-dev (wird benötigt von Paillier)
- libssl-dev (OpenSSL)
- libboost-all-dev (Boost)
- qt5-default (Qt)
- qt5-qmake (Qt)

Die Kompilierung lässt sich wie folgt ausführen. Zuerst sollte ein Terminal im Ordner `./build` geöffnet werden, danach sind zwei Befehle auszuführen:

```
> qmake ../bitvoting/bitvoting.pro
> make
```

Die erzeugte Bitvoting kann anschließend gestartet werden. Folgender Befehl gibt eine erste Übersicht:

```
> ./bitvoting --help
```

2 Eingebundene Projekte

LevelDB Für dieses Projekt wurde LevelDB[1] als Datenbank genutzt. LevelDB speichert Daten als Key/Value Paar, unterstützt eine Vielzahl an Betriebssystemen und liefert auch bei einer großen Datenmenge einen schnellen Zugriff.

Paillier Paillier ist ein Kryptosystem, welches eine Homomorphe Verschlüsselung erlaubt. Homomorphe Verschlüsselung ermöglicht Operationen auf Ciphertexten, die sich auf den Plaintextraum widerspiegeln. So kann jede Stimme einer Wahl ausgezählt werden, ohne die Vertraulichkeit der Stimme zu verlieren. Bei jeder Wahl wird der Paillier Public Key veröffentlicht, mit dem jeder Wähler seine Stimme verschlüsseln kann. Zusätzlich wird ein Zero-Knowledge-Proof (ZKP) erstellt, mit welchem sich beweisen lässt, dass die Verschlüsselung tatsächlich aus einem gültigen Plaintextraum erzeugt wurde[2]. Eine weitere Eigenschaft von Paillier ist die Aufteilung des Private Keys. Zu jeder Wahl wird eine Gruppe von Trustees bestimmt, von denen jeder einen Teil des Private Keys besitzt. Während der Entschlüsselung operiert jeder Trustee nur mit seinem Private Key und erzeugt damit eine partielle Entschlüsselung des Ciphertexts sowie einen passenden ZKP, der beweist, dass er tatsächlich seinen Teil des Private Keys zur Entschlüsselung benutzt hat. Zur finalen Entschlüsselung werden alle partiellen Entschlüsselungen kombiniert, sodass sich die Summe der Plaintexte ergibt. Stehen nicht genug partielle Entschlüsselungen zur Verfügung, kann der Plaintext nicht berechnet werden.
Die Implementierung des Paillier Projekt in C++[3] unterstützt derzeit nicht die Aufteilung des Private Keys und die ZKP für die Verifizierung der abgegebenen Stimme. Dieses Features wurde im Rahmen des Projektes in C++ implementiert, basierend auf der Arbeit von Ivan B. Damgård, Mads J. Jurik[4] und der Java Implementierung von Dr. Murat Kantarcioglu und James Garrity[5].

Boost Boost ist eine weit verbreitete Sammlung verschiedener Erweiterungen zu C++.

OpenSSL Stellt Implementierungen für verschiedene Verschlüsselungsysteme und Hashverfahren zur Verfügung. OpenSSL wird genutzt für die Signierung bzw. Verifizierung von Signaturen. Außerdem wird die SHA2 Implementierung von OpenSSL für die Generierung von Hashwerten genutzt.

Bitcoin Bitcoin ist eine Kryptowährung, die dezentral arbeitet und den Austausch der gleichnamigen Währung ermöglicht. Konzepte, wie das Mining oder die Blockchain, wurden aus Bitcoin übernommen.

---

1 http://leveldb.org/

2 http://people.csail.mit.edu/rivest/voting/papers/BaudronFouquePointchevalPoupardStern-PracticalMultiCandidateElectionSystem.pdf

3 http://acsc.cs.utexas.edu/libpaillier/

4 http://www.brics.dk/RS/00/45/

5 http://cs.utdallas.edu/dspl/cgi-bin/pailliertoolbox/

3 Transaktionen - Wahlablauf

Bitvoting kennt vier verschiedene Arten von Transaktionen. Jede Transaktion muss von seinem Erzeuger signiert werden. Außerdem wird jede Transaktion erst dann als gültig angesehen, wenn sie als Teil eines Blockes in der Blockchain existiert. Als Resultat wird auch eine selbst erzeugte Transaktion (z.B. eine Election) erst dann angezeigt (siehe Kapitel GUI), wenn sie vom Netzwerk akzeptiert wurde (siehe Kapitel Mining).

Election Diese Transaktion stellt eine Wahl dar. Sie enthält zum einen alle Informationen zur Wahl, wie den Namen, die gestellten Fragen und erlaubten Antworten, die Liste aller berechtigter Wähler und Trustees und den Paillier Public Key zur Verschlüsselung einer Stimme. Zum anderen ist die Wahl eröffnet, sobald die Transaktion gemint und damit als Teil eines gültigen Blockes akzeptiert wurde. Der Erzeuger kann zusätzlich den Zeitpunkt angeben, ab dem keine neuen Stimmen akzeptiert sollen (Ende der Wahl).

Vote Jeder Client kann (sobald die Election Transaktion in der Blockchain existiert), seine Stimme zur Wahl abgeben. Damit die Stimme später ausgezählt wird, muss diese Transaktion mit einem Key signiert werden, der in der Election Transaktion als berechtigter Wähler gekennzeichnet wurde. Eine Vote Transaktion enthält neben Informationen zur Identifizierung der dazugehörigen Wahl die verschlüsselten (Paillier) Antworten des Wählers, sowie einen ZKP. Mit diesem lässt sich beweisen, dass die Verschlüsselung aus einem gültigen Plaintext (Ja/Nein) erzeugt wurde.

Tally Die Tally Transaktion wird vom Wahl Ersteller für diese Wahl generiert, wenn eine Auszählung der Stimmen gewünscht wird. Dabei veröffentlicht die Transaktion den Hash des Blockes, bis zu welchem die Stimmen der Wahl ausgezählt werden sollen (beginnend mit dem Block, der die Election Transaktion enthält). Ein zusätzliches Flag signalisiert das endgültige Ende einer Wahl.

Trustee Tally Sobald ein Client eine Tally Transaktion registriert für eine Wahl, bei welcher er als Trustee teilnimmt, beginnt er mit seinem Teil der Entschlüsselung. Dazu sammelt er in der Blockchain alle Stimmen zur Wahl die zwischen der Election Transaktion und dem in der Tally Transaktion angegebenen Block liegen. Für diese nutzt er zur Entschlüsselung seinen Teil des Pallier Private Key. Anschließend veröffentlicht er eine Trustee Tally Transaktion, welche später zur Auszählung genutzt wird. Die Transaktion wird mit dem Key signiert, der ihn als Trustee der Wahl identifiziert.

Ende einer Wahl (Auszählung): Nachdem genug Trustees ihre Entschlüsselung veröffentlicht haben, kann jeder Client mit der Auszählung beginnen. Dazu sammelt er alle zu einer Wahl gehörigen Trustee Tallies und vereinigt deren Entschlüsselung, um das Ergebnis zu erhalten. Dabei werden keine Informationen zu einzelnen Stimmabgaben enthüllt, lediglich das Endergebnis wird berechnet (Anzahl an JA Stimmen).

4 Mining

Jeder Client hat die Möglichkeit Transaktion zu minen. Mining vereint mehrere Transaktionen zu einem Block. Dabei werden die Transaktionen verifiziert, indem ihr Inhalt mit Informationen aus bereits gültigen Transaktionen (gespeichert in früheren Blöcken, siehe Kapitel Blockchain) abgeglichen wird. Anschließend muss der Miner einen Proof of Work ausarbeiten. Dabei wird versucht einen Hash zu erzeugen, der kleiner ist als ein vom Netzwerk festgelegter Wert. Als feste Eingabe der Hashfunktion werden die Hashwerte der Transaktionen genutzt, die Teil des Blockes werden sollen und der Hash des Vorgängers des Blockes. Als variable Eingabe wird eine Nonce genutzt. Sobald der Miner eine Nonce gefunden hat, mit der die Anforderung erfüllt ist, gilt der Block als gültig. Der Miner signiert den Block und schickt ihn ins Netzwerk. Alle Teilnehmer prüfen den Block und fügen ihn an als letzten Block in ihre Blockchain ein. Das Mining System ist modular aufgebaut, sodass weitere Strategien, wie die Nonce möglichst schnell gefunden werden kann, hinzugefügt werden können.

5 Blockchain

Die Blockchain speichert alle Blöcke, welche vom Netzwerk jemals als gültig akzeptiert wurden. Dabei enthält jeder Block den Hash eines anderen Blocks, wodurch sich eine strikte Hierachie von Blöcken ergibt (Chain). Als gemeinsamen Ursprung teilen alle Clients den Genesis Block, dessen Hash in den Settings gespeichert wird. Ein einmal akzeptierter Block kann weder gelöscht, noch verändert werden. Die Blockchain wird genutzt um die Gültigkeit neuer Transaktionen verifizieren zu können. Außerdem ermöglicht die Blockchain maximale Transparenz einer Wahl bei gleichzeitiger Geheimhaltung der einzelnen Stimmen. Jeder Client speichert lokal die gesamte Blockchain und kann sich so selbst von

der Gültigkeit jeder Transaktion überzeugen. Meldet sich ein Client neu im Netzwerk an, bekommt er von den anderen Teilnehmern die bisherigen Blockchain. Zusätzlich vergleicht jeder Client regelmäßig seine Blockchain mit anderen um Unstimmigkeiten erkennen und reagieren zu können. Blöcke entstehen als Ergebnis des Mining Prozesses (siehe Kapitel Mining).

6 Datenbanken

ElectionDB Die ElectionDB wird genutzt um Informationen über alle Wahlen zu speichern, die für den Client von Interesse sind. Dies betrifft z.B. Wahlen, in denen der Client als Voter oder Trustee teilnimmt oder die er selbst erstellt hat. Zu den gespeicherten Informationen gehört der aktuelle Status der Wahl (beendet/laufend), seine abgegebene Stimme und gegebenenfalls das Ergebnis der Auszählung.

SignKeyDB Jeder Client kann eine unbegrenzte Anzahl von Signatur Keys besitzen. Um erfolgreich an einer Wahl teilnehmen zu können, muss der Client seine Vote Transaktion mit einem Key signieren, welcher zuvor in der Liste aller autorisierten Wähler als Teil der Election Transaktion veröffentlicht wurde. Der SignKeyStore dient als Cache für die Datenbank, er hält alle Keys für die Zeit der Programmausführung bereit und stellt verschiedene Operationen zur Verfügung, wie die Erzeugung neuer Signatur Keys. Jeder Signatur Key kann vom Client mit einer Rolle gekennzeichnet werden. Diese hilft ihm zu unterscheiden, welche Keys er für welche Operation vorgesehen hat.

PaillierDB In dieser Datenbank werden die Paillier Private Keys gespeichert, die ein Trustee benötigt, um seinen Teil während der Entschlüsselung erfüllen zu können. Zu jedem Private Key wird sowohl der Hash abgespeichert, welcher die zugehörige Wahl identifiziert als auch der Signatur Key, mit welchem der Trustee seine Tally Transaktion signieren muss.

BlockChainDB Aufgrund der Größe/Länge der Blockchain ist es ineffizient, jeden Block komplett in einer Datenbank zu speichern. Der schnelle Zugriff auf alle Blöcke gleichzeitig ist auch nicht nötig, weil der Client für eine Operation in der Regel nur bestimmte Blöcke benötigt. Aus diesem Grund werden die Blöcke auf der Festplatte gespeichert in sogenannten Blockfiles. Die Größe eines Blockfiles ist beschränkt um den Zugriff auf ein einzelnes Blockfile performant zu halten. Ein Blockfile kann über seine ID identifiziert werden (Beispiel: blockfile_0000000032.bin für das 32te Blockfile). In der Datenbank werden zu jedem Block (und jeder Transaktion) nun die ID des Blockfile und die Position innerhalb des Blockfiles gespeichert, an dem der Block später wieder gefunden werden kann. Zudem wird immer der Hash des neusten Blockes gespeichert.

7 Netzwerk

Das Wahlsystem ist auf einem P2P Netzwerk aufgebaut, dadurch ist jeder Peer ein gleichwertiger Teilnehmer im Netzwerk. Dieses wurde als separates Projekt implementiert. Es unterstützt alle Funktionalitäten, die notwendig sind, damit Peers sich untereinander verbinden und Nachrichten austauschen können. Neue Peers benötigen Informationen über einen bereits mit dem Netzwerk verbundenen Peer (Bootstrapping). Über diesen werden weitere Peers gefunden und Verbindungen hergestellt (Ping/Pong). Regelmäßige Heartbeat Nachrichten überprüfen die Verfügbarkeit anderer Peers. Wurde eine neue Transaktionen vom Client erstellt, wird sie über einen Flooding Mechanismus innerhalb des Netzwerkes verteilt, wobei jeder Client in der Lage ist bereits weitergereichte Transaktionen zu erkennen, um die Bildung von Zyklen zu verhindert. Neben Transaktionen werden auch Informationen über die Blockchain regelmäßig zwischen den Peers ausgetauscht.

8 GUI

Für die GUI wurde das Framework QT verwendet, welches auch in Bitcoin genutzt wird. Wird der Client gestartet, präsentiert sich die GUI wie folgt:

Main window Im Main Window werden alle Wahlen aufgelistet, in denen der Client teilnimmt (als Wähler, Trustee oder Creator). Es werden zusätzliche Informationen bereit gestellt, z.B. ob schon gewählt wurde, wann die Wahl beendet sein wird und gegebenenfalls das Ergebnis der Wahl.

Manage Keys Ein Klick öffnet ein neues Fenster, indem die eigenen Signatur Keys verwaltet werden können. Es werden bestehende Keys mit ihrer jeweiligen Rolle angezeigt. Die Rolle eines Keys spiegelt sich nicht im Key selbst wider (sie hat keine Auswirkung auf die erzeugte Signatur). Sie bietet dem Nutzer aber eine Möglichkeit, eine bessere Übersicht über seine eigenen Keys zu haben, indem er angeben kann, welchen Key er für welchen Zweck erzeugt hat. Keys können zusätzlich importiert und exportiert werden.

Create Election In diesem Fenster können neue Wahlen erstellt werden. Dazu muss der Nutzer zuerst auswählen, mit welchem Signatur Key er die Transaktion später signieren möchte. Anschließend wird das Verzeichnis gewählt, indem die Pallier Private Keys abgespeichert werden sollen, um sie an die Trustees verteilen zu können. Im letzten Schritt kann der Nutzer alle Angaben zur Wahl eingeben.

Vote Möchte der Nutzer wählen, kann er mit diesem Button den Wizard zu einer Wahl starten. Alle Fragen der Wahl werden ihm nacheinander angezeigt, er wählt zu jeder die gewünschte Antwort aus. Möchte er für eine Wahl erneut wählen, so wird seine alte Stimmabgabe ungültig.

Tally Mit diesem Button kann der Wahl Ersteller die Tally Transaktion in das Netzwerk schicken. Er wählt den Block aus, bis zu welchem die Stimmen ausgezählt werden sollen. Zusätzlich kann er entscheiden, ob die Wahl endgültig beendet werden soll.

See Results Ist eine Wahl beendet, so kann jeder Teilnehmer durch einen Klick auf diesen Button die Auszählung starten. Das Ergebnis der Wahl (wie viele JA Stimmen hat es gegeben) werden für jede Frage angezeigt.

9 Related Work - Helios

Helios ist ein Voting System von Ben Adida[6], welches ähnliche Ziele verfolgt wie Bitvoting. Es soll für jeden Wähler möglich sein, die Auszählung der Stimmen verifizieren zu können. Zudem soll die Wahl möglichst transparent verlaufen, die (verschlüsselten) Stimmen also für jeden Teilnehmer sichtbar sein. Einige Unterschiede sollen hier dargestellt werden:

Helios basiert auf einem Client-Server System. Obwohl der Server nicht in der Lage ist eine Wahl unbemerkt zu manipulieren, ist er zwingende Voraussetzung für jede Wahl. Damit ergibt sich ein Single-Point-Of-Failure. Das Bitvoting Projekt implementiert ein Peer-to-Peer (P2P) Netzwerk, in dem es keine zentrale Instanz gibt. Es ist robuster gegenüber Ausfällen. Allerdings sind manche Funktionalitäten in einem P2P Netzwerk schwerer umzusetzen, was einige Einschränkungen zur Folge hat. So ist es schwer zu bestimmen, zu welchem Zeitpunkt genau ein Wähler seine Stimme abgegeben hat, da nicht alle Teilnehmer innerhalb des Netzwerkes die Nachricht über seine Stimmabgabe zum gleichen Zeitpunkt erhalten und es auch keine zentrale Instanz gibt, die den Zeitpunkt festlegen/verifizieren könnte. Bitvoting löst dieses Problem, indem der Zeitraum der Gültigkeit einer Wahl an der Blockchain festgemacht wird. Stimmen werden als gültig angesehen, wenn sie innerhalb der Chain hinter der Election Transaktion liegen und vor der dazugehörigen Tally Transaktion. Der Wahl Erzeuger kann in der Wahl angeben, zu welchem Zeitpunkt er plant, die Tally Transaktion ins Netzwerk zu schicken. Eine weitere Einschränkung betrifft die Wahl selbst. Eine Transaktion, welche einmal Teil der Blockchain wurde (Mining), kann nicht mehr geändert werden. Wahländerungen, wie das Hinzufügen weiterer Wähler, sind daher nur möglich, wenn die Wahl (inklusive Änderungen) erneut vom Erzeuger in das Netzwerk geschickt wird. Wahlen können also nur neu erstellt und nicht direkt verändert werden. Im Helios System sind beide Features durch den Server leichter umsetzbar.
Zur homomorphen Ver- und Entschlüsselung nutzt Helios ein ElGamal Kryptosystem. Im Gegensatz dazu wird in Bitvoting das Paillier System verwendet (siehe Kapitel Eingebundene Projekte). Beide Kryptosysteme bieten ähnliche Funktionalitäten und erfüllen die Anforderungen eines E-Voting System (Verifizierung der Gültigkeit einer Stimme, Verifizierung der Entschlüsselung/Auszählung, beides durch ZKPs).

10 Future Work

- Paillier
  - Aktuell werden nur Ja/Nein Fragen unterstützt. Um Wahlen mit mehr Antwortmöglichkeiten zu erstellen (z.B einer Liste von zu wählenden Personen) kann das Paillier Projekt erweitert werden. Es befindet sich in `./paillier/paillier.h`. Anschließend muss die Wahl (`./election.h`) und Auszählung (`./electionmanager.h`) entsprechend angepasst werden.
  - Zur Erstellung einer Wahl gehört die Generierung der Paillier Public/Private Keys. Die einzelnen Teile des Private Keys müssen anschließend an die Trustees geschickt werden. Die Implementierung eines sicheren Kanals zur Übertragung ist eine Aufgabe für zukünftige Projekte.

[6] `http://documentation.heliosvoting.org/verification-specs/helios-v4`

- Mining
  - Aktuell gibt es eine Technik, wie versucht wird eine passende Nonce für den Proof-of-Work im Mining Prozess zu finden. Dies kann durch weitere Stretegien/Techniken ergänzt werden, damit das Mining z.B. besser an die Möglichkeiten des Client angepasst werden kann (Hardware etc). Im Code findet sich der Mining Vorgang in der `./miner.h` Datei.
  - Ergänzend zum ersten Punkt könnten neue Mining Techniken weitere Features enthalten. Kommen Transaktionen aus dem Netzwerk bei dem Client an, muss z.B. ein zusätzlicher Mining Prozess gestartet werden, es können keine Transaktionen in den laufenden Mining Prozess eingebunden werden. Zukünftige Techniken können das unterstützen. Auch hier ist der Code in der Datei `./miner.h` zu finden.
- Signatur Keys
  - Eine Art Adressbuch kann vom Client zur Verfügung gestellt werden, um zusätzlich zu den eigenen auch andere bekannte Public Keys aufzulisten und zu bearbeiten. Dazu kann eine weitere Datenbank hilfreich sein. Als Orientierung kann der Code für die Datenbank der eigenen Keys genutzt werden, er befindet sich in der Datei `./database/signkeydb.h`. Der dazugehörige Store liegt unter `./store.h`.
  - Eine PKI ermöglicht Zertifikate, die wiederum den Public Key mit der Identität einer Person verbinden. Damit könnten Wähler/Trustees besser identifiziert bzw. der Public Key verifiziert werden. Damit verbunden ist auch die Revokation eines Signatur Key ein denkbares Feature für zukünftige Projekte.
  - Die verwendeten Signatur Keys basieren auf den Klassen des Bitcoin Projektes, die wiederum OpenSSL 's EC_KEY nutzen. Der Import von extern erzeugten Signatur Keys ist damit nicht ausgeschlossen und eine mögliche Erweiterung für das Projekt. Ein Beispiel sind PGP Keys, welche als zusätzlichen Vorteil bereits ein bestehendes Vertrauensmodell haben und damit den oben genannten Punkt der PKI abdecken. Die Signatur Keys werden in der Datei `./bitcoin/key.h` erstellt, dort finden sich auch die Implementierungen aller Key Operationen (signieren, verifizieren etc).
  - Der Export der Signature Keys kann erweitert werden, sodass diese beim Export mit einem ausgesuchten Passwort verschlüsselt werden. Damit können diese auf der Festplatte gespeichert werden und sind gleichzeitig vor unbefugtem Zugriff geschützt. Der Export eines Signature Keys wird geregelt in der Datei `./gui/dialogmanagekeys.h`, genauer in der Methode `on_btnExport_clicked()`. Entsprechend muss auch der Import erweitert werden, er befindet sich in der Methode `on_btnImport_clicked()`.
- Nach einer Wahl sind alle Informationen in der Blockchain gespeichert und daher für jeden Client verfügbar. Zukünftige Projekte können eine Export Funktion bereit stellen, welche die relevanten Informationen aus der Blockchain sammelt und gebündelt zur Verfügung stellt. Dazu gehören die Transaktionen selbst, die dazugehörige Auszählung der Stimmen, alle ZKP etc. Die Blockchain wird implementiert in der Datei `./database/blockchaindb.h`, eine Übersicht über die verschiedenen Transaktionen wird gegeben in den Dateien des Ordners `./transactions`. Die Auszählung befindet sich in der Datei `./electionmanager.h`.
- Das P2P Netzwerk unterstützt derzeit nur IPv4 zur Adressierung von Peers. Als mögliche Erweiterung könnte in Zukunft IPv6 als Adressierung genutzt werden. Die verantwortlichen Dateien für das P2P Netzwerk befinden sich im Ordner `./network`.
- Aufgrund der Nutzung von IPv4 ergeben sich die bekannten Probleme der NAT-Traversierung. In Bitvoting wurden hierfür bisher keine Mechanismen implementiert, die dieses Problem zu umgehen versuchen (Hole Punching, ICE, TURN etc.).
- Bitvoting wurde unter Linux programmiert und auch ausschließlich unter Linux/Ubuntu getestet. Alle verwendeten Bibliotheken sind allerdings auch für andere Betriebssysteme verfügbar (Windows & MacOS). Als zukünftige Aufgabe kann das Projekt auf andere Plattformen migriert werden.