Thesis

Paul van Grol

June 4, 2018

Contents

Chapter 1

Introduction

This thesis has been inspired by a recent paper that introduced OmniLedger [10]: a scalable, decentralized ledger. In a world where more and more uses are found for blockchain technology, the promise of such a technology that scales in performance with the amount of users is incredibly interesting. Considering that current systems might actually perform worse when the amount of users increases, it is interesting to look at OmniLedger to see whether it can help in that regard.

In particular, this thesis focuses on multisignature algorithms that may be used to improve the performance of RandHound [18]. RandHound is used by OmniLedger to ensure that the system remains uncompromised. Evaluation has shown that RandHound takes up more than 70% of the total runtime in the bootstrap process of OmniLedger [10]. Since this process happens periodically, improving the performance of RandHound should lead to performance improvement of OmniLedger.

Therefore the research question of this thesis is as follows: " .. ". To answer this question, some background is needed, before comparisons can be made between various multisignature algorithms.

As such the second chapter, Mathematical Background, introduces algebraic varieties, leading up to an explanation of elliptic curves and some properties that are used to great effect in cryptography. The chapter concludes with an explanation of the Weil pairing on elliptic curves, which is used in one of the multisignature algorithms used.

The third chapter, Technical Background, then introduces the technological background of ledgers, blockchain and multisignatures among other terms and concepts used to clearly show the context of the research.

Chapter four, Related Work, discusses related works and contributions made by those.

In chapter five, BLS Multisignature Scheme and six, Schnorr Multisignature Scheme, two different multisignature algorithms are studied, which leads up to a comparison of the two algorithms and a conclusion regarding the most suitable choice for RandHound in chapter seven.

Chapter 2

Mathematical Background

This chapter provides the mathematical background to elliptic curve cryptography. If the reader is already familiar with elliptic curves and the Weil pairing, skipping to the section on elliptic curve cryptography should not hinder the understanding of that chapter. If the reader is unfamiliar, or wishes to refresh his memory, reading this chapter is advised.

A background in and understanding of fields, rings and Galois groups is assumed in this chapter.

This chapter will start with algebraic varieties and lead up to elliptic curves and the Weil pairing, finishing with an explanation of elliptic curve cryptography.

As source material in this chapter, the excellent work of Joseph H. Silverman is used [17, Chapter 1-3], along with some of his slides [16].

2.1 Algebraic Varieties

2.1.1 Notation

Before explaining algebraic varieties, the following notation is set.

- K is a perfect field.
- $\bar{K}$ is a fixed algebraic closure of K.
- $\bar{K}[X] = \bar{K}[X_1, \ldots, X_n]$ is a polynomial ring in n variables.
- $G_{\bar{K}/K}$ is the Galois group of $\bar{K}/K$.

2.1.2 Affine Varieties

The following definition defines Cartesian n-space and the subsets that are defined by zeros of polynomials.

Definition 2.1.2.1. Cartesian, also known as affine, n-space, which is implied to be over K, is the set of n-tuples.

$$\mathbb{A}^n = \mathbb{A}^n(\bar{K}) = \{P = (x_1, \ldots, x_n) : x_i \in \bar{K}\}$$

This leads to the set of K-rational points of $\mathbb{A}^n$ as the following set.

$$\mathbb{A}^n(K) = \{P = (x_1, \ldots, x_n) \in \mathbb{A}^n : x_i \in K\}$$

From this point on, any space used is, indeed, an affine space. Using the Galois group $G_{\bar{K}/K}$ allows for an alternative description of $\mathbb{A}^n(K)$, because the Galois group fixes elements $P \in \mathbb{A}^n$.

Remark 2.1.2.2. Considering the Galois group $G_{\bar{K}/K}$ leads to an action on $\mathbb{A}^n$ such that for $\sigma \in G_{\bar{K}/K}$ and $P \in \mathbb{A}^n$, it leads to

$$P^\sigma = (x_1^\sigma, \ldots, x_n^\sigma).$$

This then allows to define $\mathbb{A}^n(K)$ as

$$\mathbb{A}^n(K) = \{P \in \mathbb{A}^n : P^\sigma = P, \forall \sigma \in G_{\bar{K}/K}\}.$$

Using ideals in $\bar{K}[X]$ leads to the construction of an algebraic set on $\mathbb{A}^n$.

Definition 2.1.2.3. Taking $I \subset \bar{K}[X]$ as an ideal, each of these can be associated with a subset of $\mathbb{A}^n$ such that

$$V_I = \{P \in \mathbb{A}^n : f(P) = 0, \forall f \in I\}.$$

Any set of this form is an algebraic set.

On an algebraic set, in turn, the ideal may be defined.

Definition 2.1.2.4. Given an algebraic set V, the ideal of V is

$$I(V) = \{f \in \bar{K}[X] : f(P) = 0, \forall P \in V\}.$$

If $I(V)$ can be generated by polynomials in $K[X]$ for an algebraic set V, it is defined over K and noted as V/K.

Using an algebraic set V and the corresponding ideal $I(V)$ leads to the definition of an affine variety.

Definition 2.1.2.5. V is called an affine variety if $I(V)$ is a prime ideal in $\bar{K}[X]$.

2.1.3 Projective Varieties

Making an affine space a projective space is done by adding the so called "points at infinity". This same process can be recreated to define projective varieties.

Definition 2.1.3.1. $\mathbb{P}^n = \mathbb{P}^n(\bar{K})$ denotes the projective n-space over K and is constructed as the set of all $(n+1)$ tuples

$$(x_0, \ldots, x_n) \in \mathbb{A}^{n+1}.$$

These tuples must be such that at least one element is nonzero modulo the following equivalence relation

$$(x_0, \ldots, x_n) \sim (y_0, \ldots, y_n)$$

if there is a $\lambda \in \bar{K}^*$ such that $x_i = \lambda y_i, \forall i$. The resulting equivalence class

$$\{(\lambda x_0, \ldots, \lambda x_n) : \lambda \in \bar{K}^*\}$$

is noted as $[x_0, \ldots, x_n]$, with the elements being called homogeneous coordinates for the point $P \in \mathbb{P}^n$ that corresponds to it. This leads to the set of K-rational points in $\mathbb{P}^n$

$$\mathbb{P}^n(K) = \{[x_0, \ldots, x_n] \in \mathbb{P}^n : x_i \in K\}$$

As with affine varieties, the Galois group $G_{\bar{K}/K}$ allows for an alternate definition of $\mathbb{P}^n(K)$.

Remark 2.1.3.2. The Galois group $G_{\bar{K}/K}$ acts on $P \in \mathbb{P}^n$ via

$$[x_0, \ldots, x_n]^\sigma = [x_0^\sigma, \ldots, x_n^\sigma]$$

and as such it leads to

$$\mathbb{P}^n(K) = \{P \in \mathbb{P}^n : P^\sigma = P, \forall \sigma \in G_{\bar{K}/K}\}$$

With the existence homogeneous coordinates, homogeneous polynomials and ideals can be defined.

Definition 2.1.3.3. A polynomial $f \in \bar{K}[X]$ is homogeneous of degree d if

$$f(\lambda X_0, \ldots, \lambda X_n) = \lambda^d f(X_0, \ldots, X_n), \forall \lambda \in \bar{K}$$

An ideal $I \subset \bar{K}[X]$ is homogeneous if it is generated by homogeneous polynomials.

With f a homogeneous polynomial, it is interesting to look at points $P \in \mathbb{P}^n$ where $f(P) = 0$. Using these points, for each homogeneous ideal I a subset can be defined.

Definition 2.1.3.4. Taking f to be a homogeneous polynomial, any set of the following form for a homogeneous ideal I is a projective algebraic set.

$$V_I = \{P \in \mathbb{P}^n : f(P) = 0, \forall f \in I\}$$

Such a set has, of itself, a homogeneous ideal.

Definition 2.1.3.5. $I(V)$, the homogeneous ideal of V is the ideal of $\bar{K}[X]$ that is generated with homogeneous f by

$$\{f \in \bar{K}[X] : f(P) = 0, \forall P \in V\}$$

The homogeneous ideal is used to define a projective variety.

Definition 2.1.3.6. If the homogeneous ideal $I(V)$ of projective algebraic set V_I is a prime ideal in $\bar{K}[X]$, the set V_I is called a projective variety.

2.2 Elliptic Curves

An elliptic curve is an algebraic curve of genus one with a certain base point. This section will begin with elliptic curves given by Weierstrass equations, explain the group action on elliptic curves, then look at arbitrary elliptic curves and show that all have a Weierstrass equation study the Weil pairing on elliptic curves. The final section will explain elliptic curve cryptography and give some uses.

2.2.1 Weierstrass Equations

The Weierstrass equation for elliptic curves is written as as follows

Definition 2.2.1.1. Given a base point O and $a_1, \ldots, a_6 \in \bar{K}$ the Weierstrass equation for an elliptic curve is

$$Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3.$$

This is generally written with non-homogeneous coordinates $x = X/Z$ and $y = Y$

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6.$$

If $a_1, \ldots, a_6 \in K$, E is said to be defined over K.

If $char(\bar{K}) \neq 2$, then this equation an be simplified further by substituting

$$y = \frac{1}{2}(y - a_1x - a_3).$$

This leads to the following equation

$$E : y^2 = 4x^3 + b_2x^2 + 2b_4x + b_6$$

where

$$b_2 = a_1^2 + 4a_2,\ b_4 = 2a_4 + a_1a_3,\ b_6 = a_3^2 + 4a_6.$$

Furthermore define

$$b_8 = a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2,$$

$$c_4 = b_2^2 - 24b_4,$$

$$c_6 = -b_2^3 + 36b_2b_4 - 216b_6,$$

$$\Delta = -b_2^2 - 8b_4^3 - 27b_6^2 + 9b_2b_4b_6,$$

$$j = c_4^3/\Delta,$$

$$\omega = \frac{dx}{2y + a_1x + a_3} = \frac{dy}{3x^2 + 2a_2x + a_4 - a_1y}.$$

Using those, it is easy to see that

$$4b_8 = b_2b_6 - b_4^2 \text{ and } 1728\Delta = c_4^3 - c_6^2.$$

If $char(\bar{K}) \neq 2, 3$, the substitution

$$(x, y) = (\frac{x - 3b_2}{36}, \frac{y}{108})$$

allows to eliminate x^2, leaving the simple equation

$$E : y^2 = x^3 - 27c_4x - 54c_6.$$

Some of the previously defined quantities have specific meanings.

Definition 2.2.1.2. The value Δ is the discriminant of the Weierstrass equation, j is the j-invariant of the elliptic curve and ω is the invariant differential that is associated with the Weierstrass equation.

Because $27c_4$ and $54c_6$ are simple values, the Weierstrass equation of an elliptic curve when the characteristic of K is not 2 or 3 is

$$E : y^2 = x^3 + Ax + B.$$

In this case it follows that

$$\Delta = -16(4A^3 + 27B^2) \text{ and } j = -1728\frac{(4A)^3}{\Delta}.$$

2.2.2 Group Action

In this section the elliptic curve E is given by a Weierstrass equation. As such, $E \subset \mathbb{P}^2$ consists of points $P = (x, y)$ satisfying

$$f(x, y) = y^2 + a_1xy + a_3y - x^3 - a_2x^2 - a_4x - a_6 = 0$$

It is easy to see that the equation has degree three, and that, as such, there are three points of interesection for line $L \subset \mathbb{P}^2$. These points P, Q, R need not be distinct, since L may be a tangent. Using this, the composition law $\oplus$ on E is defined as follows.

Definition 2.2.2.1. Let $P, Q \in E$ and let L be the line through P and Q if the points are distinct, or the tangent to E at P if $P = Q$. Let R be the other point of intersection of L with E. Denote by L' the line through R and O. Now L' intersects E at R, O and a third point denoted by $P \oplus Q$.

Now the usage of the symbol is justified by showing that it makes E into an abelian group with identity O.

Theorem 2.2.2.2. *The composition law $\oplus$ has the following properties:*

1. *If line L intersects E at points P, Q, R, not necessarily distinct, then*

$$(P \oplus Q) \oplus R = O$$

2. $P \oplus O = P \ \forall P \in E$

3. $P \oplus Q = Q \oplus P \ \forall P, Q \in E$

4. *Given $P \in E$ there is a point of E, denoted $\ominus P$, such that*

$$P \oplus (\ominus P) = O$$

5. *Given $P, Q, R \in E$*

$$(P \oplus Q) \oplus R = P \oplus (Q \oplus R)$$

6. *Suppose that E is defined over K, then the following is a subgroup of E*

$$E(K) = \{(x, y) \in K^2 : y^2 + a_1xy + a_3x^2 + a_4x + a_6\} \cup \{O\}.$$

Proof 2.2.2.3. 1. From the definition it follows that the line L' used to construct $P \oplus Q$ intersects E at R, O and $P \oplus Q$. Therefore the intersection point is O, and the line through O and O ends up in O. In other words, the tangent line to E at O intersects E at O with multiplicity 3.

2. Note that in this case, the lines L and L' are the same, and that as such the intersection of the line through O and $P \oplus O$ intersects E at precisely P.

3. It is easy to see that the construction of $P \oplus Q$ is symmetric in both P and Q. The line through two points that remain fixed does not depend on the order in which the points are picked.

4. Take a line through P and O and call the intersection point R. Using 1. and 2. it is easy to see that $O = (P \oplus O) \oplus R = P \oplus R$.

5. This proof will be handled later with explicit formulas.

6. If both P and Q have coordinates in K, then the coefficients of the equation of the line L connecting P and Q are in K. If, furthermore, E is defined over K then the third point will be a ratinal combination of coefficients of the line and of E, and will thus be in K.

Remark 2.2.2.4. Since it has been proven that $\oplus$ is a group operation, this thesis will from now on use $+$ instead, and $-$ for $\ominus$. Furthermore, define $mP = \overbrace{P + \cdots + P}^{m}$ if $m > 0$, $mP = \overbrace{-P - \cdots - P}^{m}$ if $m < 0$ and $0P = O$.

2.2.3 Explicit Formulas

The various group operations can also be expressed in explicit formulas, which will be derived in this section.

Theorem 2.2.3.1. *Let E be an elliptic curve given by the Weierstrass equation*

$$f(x, y) = y^2 + a_1xy + a_3y - x^3 - a_2x^2 - a_4x - a_6 = 0,$$

1. *Let $P_0 = (x_0, y_0) \in E$. Then*

$$-P_0 = (x_0, -y_0 - a_1x_0 - a_3).$$

Now let $P_1 + P_2 = P_3$ with $P_i = (x_i, y_i) \in E$.

2. *If* $x_1 = x_2$ *and* $y_1 + y_2 + a_1x_2 + a_3 = 0$ *then*

$$P_1 + P_2 = 0.$$

If this is not the case, define for $x_1 \neq x_2$

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1} \text{ and } \nu = \frac{y_1x_2 - y_2x_1}{x_2 - x_1}$$

and for $x_1 = x_2$

$$\lambda = \frac{3x_1^2 + 2a_2x_1 + a_4 - a_1y_1}{2y_1 + a_1x_1 + a_3} \text{ and } \nu = \frac{-x_1^3 + a_4x_1 + 2a_6 - a_3y_1}{2y_1 + a_1x_1 + a_3}$$

Then $y = \lambda x + \nu$ *is the line through* P_1 *and* P_2.

3. *Using the same notation as* 2., *the coordinates for* $P_3 = P_1 + P_2$ *are*

$$(x_3 = \lambda^2 + a_1\lambda - a_2 - x_1 - x_2, y_3 = -(\lambda + a_1)x_3 - \nu - a_3)$$

Proof 2.2.3.2. tba

2.2.4 Elliptic Curves

This part will focus on linking Weierstrass equations to generic elliptic curves, to show that the results achieved for Weierstrass equations apply generally.

Definition 2.2.4.1. An elliptic curve is a pair (E, O), with E a nonsingular curve of genus one, and $O \in E$. The elliptic curve E is defined over K, written as E/K, if E is defined over K and $O \in E(K)$.

To show that elliptic curves and Weierstrass equations are linked the Riemann-Roch theorem is used. See [8] for a proof.

Theorem 2.2.4.2 (Riemann-Roch)**.** *Let* C *be a smooth curve and let* K_C *be a canonical divisor on* C. *There is an integer* $g \geq 0$, *called the genus of* C, *such that for every divisor* $D \in Div(C)$,

$$\ell(D) - \ell(K_C - D) = deg(D) - g + 1$$

Theorem 2.2.4.3. *Let* E *be an elliptic curve defined over* K, *then there exist functions* $x, y \in K(E)$ *such that the map*

$$\phi : e \to \mathbb{P}^2 \ , \ \phi = [x, y, 1]$$

gives an isomorphism of E/K *onto a curve given by the Weierstrass equation*

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

satisfying $\phi(O) = [0, 1, 0]$. *The functions* x *and* y *are called the Weierstrass coordinates for the elliptic curve* E

This proof will show that the image of the map is indeed in C as described by the Weierstrass equation. The rest of the proof may be found in [17, page 60]

Proof 2.2.4.4. tba

2.2.5 Weil Pairing

The Weil pairing will be the last of the mathematical background discussed. This pairing uses E_m, the group of m-torsion points and μ_m, the group of m-th roots of unity.

Definition 2.2.5.1. Setting, for $X \in E$ and $g \in \bar{K}(E)$

$$E_m(S,T) = \frac{g(X+S)}{g(X)}$$

the Weil pairing is a non-degenerate alternating bilineair form

$$e_m = E_m \times E_m \to \mu_m$$

This pairing has the following properties

Theorem 2.2.5.2. *The Weil e_m pairing has these properties*

1. *The pairing is bilinear*

$$e_m(S_1 + S_2, T) = e_m(S_1, T)e_m(S_2, T),$$

$$e_m(S, T_1 + T_2) = e_m(S, T_1)e_m(S, T_2),$$

2. *The pairing is alternating*

$$e_m(T, T) = 1$$

which means in particular that

$$e_m(S, T) = e_m(T, S)^{-1}$$

3. *The pairing is nondegenerate*

$$\text{If } e_m(S, T) = 1 \text{ for all } S \in E_m, \text{ then } T = O$$

4. *The pairing is Galois invariant*

$$e_m(S, T)^\sigma = e_m(S^\sigma, T^\sigma) \text{ for all } \sigma \in G_{\bar{K}/K}$$

5. *The pairing is compatible*

$$e_{mm'}(S,T) = e_m(m'S,T) \text{ for all } S \in E_{mm'} \text{ and } T \in E_m$$

Proof 2.2.5.3. 1. tba

2. tba

3. tba

The resulting proofs may be found in [17, page 96] but are not of particular interest in this thesis.

Chapter 3

Technical Background

This chapter provides the technical background to the various terms and concepts used in this thesis. If the reader is already familiar with anything explained in a section in relation to blockchain, skipping it should not hinder in further reading. If the reader is unfamiliar, or wishes to refresh his memory, reading this chapter is advised.

A bottom-up approach is used in this chapter, which means that sections may use or require terms and concepts explained in the previous section(s). Each section aims to give an intuitive understanding via an example and offers a detailed explanation. A visual representation of how the various sections are linked can be found in appendix A.

The chapter first leads up to explaining blockchain, followed by an introduction to signature schemes. To end the chapter, RandHound and finally OmniLedger are explained.

3.1 Concensus protocol

By definition consensus is an opinion that everyone in a group agrees with or accepts. An agreement, in other words. A protocol, in the technical definition, is an established method for connecting computers so that they can exchange information. A consensus protocol is therefore a method to reach agreement between connected computers.

In relation to blockchain technology, a consensus protocol [5] is a protocol used to reach agreement over input to add to the ledger, and the order in which it should be added. Blockchain technology recognises various such protocols, operating in various ways under different assumptions and requirements.

Below brief explanations will be given of the consensus protocols used by

Bitcoin and Stellar. The former uses a concept known as proof-of-work, the latter introduces a notion of trust.

3.1.1 Bitcoin

Conversely, the consensus protocol used by Bitcoin [13] does not require any trust in other participants. Instead it relies on a mechanism known as proof-of-work. In Bitcoin a reward goes to the participant that manages to create the next agreement. This agreement is a set of transactions, called a block, that fulfills the requirements set for the block. Once a participant has found such a block, he sends it to as many others as he can, so that his result will be on the chain, which makes the reward his. Other participant can easily verify that a block meets those requirements, after which they accept the result and forward it to others.

The protocol functions with the help of a one-way hash function. A one-way function is a function that makes it easy to compute $f(x)$ given x, but makes calculating x from $f(x)$ practically impossible. A one-way hash function has the added property of producing output of a fixed length.

What happens in the protocol used by Bitcoin is that each participant will create a block. This block then serves as input for the one-way hash function. The goal is to have a block that results in a hash with a certain amount of leading zero's. Given the block, others can easily verify that hashing the block results in a hash with a sufficient amount of leading zero's.

Producing a wrong block, therefore, is not in the best interest of the participant. Others will quickly refure the block, and all the effort put into creating the block will be wasted.

Therefore the notion of proof-of-work for the Bitcoin consensus protocol is apt. Arriving at a sufficient result requires the participant to try various input blocks, with no way to compute one from the desired result.

3.1.2 Stellar

To understand the Stellar Consensus Protocol [12] (SCP) some definitions are needed. SCP introduces a quorum, a set of participants that is sufficient to reach agreement, and a quorum slice, a subset of a quorum that can convince one participant of the agreement.

The basis of SCP is that participant trust others in their quorum slice to behave honestly, hence the notion of trust in SCP. Assuming all participant behave honestly and under the same set of input and rules, each participant should arrive at the same conclusion. After a participant reaches his conclusion, he publishes it to the others in his quorum slice. In publishing his

conclusion he votes for it, as do all others in his slice. The final conclusion is the one most voted for.

As long as there are sufficient reputable participants, the network can never be corrupted. Behaving dishonestly is further disincentivised by the protocol, since those participants are not trusted to be part of a quorum slice, and the influence of their opinion will then quickly be reduced. Conversely, reputable participants will be part of multiple quorum slices, which gives them a large influence over the decisions made.

To ensure that the whole network is indeed connected and reaches network wide consensus, SCP requires that there are no disjunct quorums. Since disjunct quorum can reach their opinion separate from the network, a different agreement undermines the network wide consensus. Participants wishing to join the network have to make sure that they join a quorum in such a way that no disjunct quorums are created.

3.2 Distributed Ledger

By definition, a distributed ledger [19] (DL) would be shared records. This remains true in the context of blockchain technology, but that is not all that it is. Although in use it acts as if there is one ledger that everybody uses, so that everybody agrees what has happened, the technique behind it works differently.

The most important property of a DL in the context of blockchain is the irrefutable state of truth it represents. Anything on the distributed ledger has happened exactly as described there, and no-one can alter those existing records. A DL may therefore be of interest to any group of parties looking to abolish ledger conflicts, to be able to trust that anything in the ledger has indeed happened, nothing more, nothing less, nothing else.

In a DL system, each participant holds and keeps track of his own copy of the ledger. This happens independently, so ledger states are never directly compared. The mechanism used by the sytem ensures that each participants holds the exact same conclusion on his ledger, thereby assuring the irrefutable state of truth.

What happens in the system, is that each participant starts at the same basis: an empty ledger. Input is then presented to all participants, who will handle the input as the system describes. Having handled the input and produced the result, a consensus protocol is used to ensure network wide agreement. Once agreement has been reached, each participant adds input to his ledger as agreed upon, ensuring that each participant reaches the same next ledger state.

Since it may happen that participants join at a later time, mechanisms exist to ensure that the participant can catch-up to the current ledger. This is not done by presenting the ledger, but instead by presenting the input for each ledger state, and corresponding consensus. The new participant then uses that information to construct his ledger from the basis state up, eventually reaching the same ledger state as all other participants.

Blockchain is a form of a DL, using blocks of transactions to shape the ledger and various consensus protocols to agree upon blocks to be added. Since this thesis focusses on techniques used in blockchain, any references to a chain, ledger or anything similar will mean a blockchain, a specific form of a DL.

3.3 Sharding

Sharding is a concept mostly encountered in relation to databases. [9] In that context, it refers to splitting up a large set of data in smaller pieces, shards. These shards can then be spread across distinct containers, be it a table, scheme or even different physical databases. Sharding databases can drastically improve performance and is a well-proven technique for large data sets. Google, for example, uses it for their own globally distributed database, Spanner [6].

In a more generic sense sharding refers to a partitioning of workload. By dividing a system into various parts, that each handle a particular subset of the workload, more work can be done simultaniously, which improves throughput and consequently performance. At the same time, it also means that the amount of resources requires remains limited, since it is not the whole system that has to act, but just a part of it.

Looking at sharding in blockchain [10], it is important to note that in traditional blockchain technologies, each participant handles each input. That means that in a particularly large network, each participant will have to handle a large set of transactions, which uses time and resources.

Sharding in a blockchain means dividing your participants into shards [10]. Transactions will be handled by the shard or shards to which the participants in that particular transaction belong. A transaction that takes place between participants in different shards is known as a cross-shard transaction and requires a different method to handle than transactions happening whithin a shard. Assuming that there are not too many cross-shard transactions, sharding results in a higher throughput of transactions, since each group handles a subset of the transactions simultaniously.

Since blockchain requires that malicious participants cannot influence the

result, is is important to note that a shard is somewhat of a mini blockchain, which means that sharding must happen in such a way that shards cannot be compromised.

3.4 Discrete Logarithm Problem

On the set of real numbers ($\mathbb{R}$) the $\log_b$ function has been defined as the solution to the following problem.

Definition 3.4.0.1. Given $a, b, n \in \mathbb{R}$, base b and power a of b, what is n such that $a = b^n$?

This same problem can also be defined over modulo p, which is known as the discrete logarithm problem over $\mathbb{Z}/p\mathbb{Z}$.

Definition 3.4.0.2 (Discrete Logarithm Problem over $\mathbb{Z}/p\mathbb{Z}$)**.** Given $a, b \in \mathbb{Z}/p\mathbb{Z}$, base b and power a of b, what is n such that $a = b^n \mod p$?

In a more generic sense, this problem can be defined over a group G.

Definition 3.4.0.3 (Discrete Logarithm Problem over group G)**.** Given $a, b \in G$, base b and power a of b, what is n such that $a = b^n$?

In particular, since an elliptic curve E is a group, the problem holds over elliptic curves. Taking elliptic curve E as group G, it follows that.

Definition 3.4.0.4 (Elliptic Curve Discrete Logarithm Problem)**.** Given elliptic curve E and points $P, Q \in E$, what is n such that $nP = Q$?

Although no actual proof exists, the assumption is that the discrete logarithm problem in a well chosen group G is a hard problem. The group must be well chosen, for there are groups that have a structure that allows for an algorithm to solve the problem, but there is a sufficiently large group of groups left for which no such algorithm exists.

In the context of computer science, this means that there is no known efficient algorithm to solve the problem, other than trying various solutions, quite like the mechanic used in the Bitcoin consensus protocol (section 1). More specifically, the problem being hard means that the runtime of the solution finding algorithm grows linearly to the group size. Or in other words, exponentially in the amount of digits of the group size.

Problems that are hard to solve in this sense of the word, lead to applications in cryptography. Since we can ensure a group large enough that trying all solutions becomes infeasible, it allows for cryptographic security.

3.5 Random Oracle

The random oracle model [7] was introduced by Fiat and Shamir. It is a theoretical black box that, for each unique input, presents a random output from its output domain. It is important to note that the oracle is deterministic; each unique input will produce the same output every time.

3.6 Forking Lemma

The forking lemma [2] gives a relation between the chance of a fork and the chance of success for the attacker. Bellare and Neven state that.

$$frk \geq acc \cdot (\frac{acc}{q} - \frac{1}{h}) \tag{3.1}$$

Here frk represents the chance of obtaining two good forgeries, and acc the probability of success of an adversary on a random input. The lemma shows that frk is non-neglible if acc is non-neglible. In other words, if the underlying problem is hard, then no adversary can forge signatures.

3.7 Signature Scheme

A signature is used in all manner of situations. An artist might add his signature to his creation to show that he was in fact the one to create it. People sign documents to show that they have created them, or to signify that they agree with the content. Because signatures are unique to a person and therefore difficult to recreate by others, a signature gives truth, validity to the signed piece. It can also be used to verify that someone is who they say they are. If they can produce the same signature, they are likely the person in the flesh.

Digital signatures serve the exact same purpose. It is a proof of identity, genuineness. A signature scheme is a way to present and verify the authenticity of a digital message. Signature schemes work with a pair of a private and public. The private key is some piece of information that is only known to the signer, the public key is made public and used in signature verification. The signer uses this private key to sign a message, which produces a signature. This signature can be verified by others using the public key and the signare algorithm.

In further reading, a signature will refer to a digital signature produced by a signature scheme and any digital message with such a signature has been signed.

A classical signature is one that proves the identity of just one person. This is, however, not the only kind. Among others, there are group- and multisignatures, which are discussed below. A group signature scheme allows a signer to sign on behalf of a group, whereas a multisignature scheme allows a group to sign a document, with the result begin a joint signature that is more compact than all individual signatures separately.

3.7.1 Group Signature Scheme

As explained just before, a group signature scheme allows a user to sign a message on behalf of a group. Various implementations of group signature schemes each offer different behaviour. It is possible to have a group signature be created in such a way that the signer may be determined, but also so that the signer remains obfuscated. A scheme may offer restrictions on the signature, so that it may only be produced if sufficient signers participate in the signing. In any case, a group signature implies that the group, as a whole, agrees with the content of the message signed.

3.7.2 Multisignature Scheme

A multisignature scheme is somewhat of a generalisation of a group signature scheme. A multisignature scheme will produce a joint signature for the group of signers. It finds particular use in blockchain technology because it limits the size of the signature, and therefore the size of the input that has to be sent over the network.

3.7.3 Schnorr Group

A Schnorr group [14], proposed by Claus P. Schnorr, inventor of the Schorr Signature Scheme, is defined as follows.

Definition 3.7.3.1 (Schnorr Group)**.** Generate p, q, r such that $p = qr + 1$ with p, q prime. Then pick any $1 < h < p$ such that $h^r \not\equiv 1 \mod p$. Then $g = h^r \mod p$ is the generator of the Schnorr group, which is a subgroup of $\mathbb{Z}_p^*$ of order q

Proof 3.7.3.2 (Schnorr Group)**.** It is trivial to see that the Schnorr group is indeed a group. Note that the order of $\mathbb{Z}_p^*$ is $p - 1 = qr$. Because $\mathbb{Z}_p^*$ is cyclic, for each divisor d of qr there is one subgroup of order d, generated by $a^{n/d}$, with $a \in \mathbb{Z}_p^*$. As such there is a subgroup of order q generated by $a^{qr/q}$, which is precisely the h picked. As such the order of the Schnorr group is indeed q.

For puruposes in cryptography, p is typically 1024 to 3072 bits and q 160 to 256 bits, which means that the discrete logarithm problem is sufficiently hard to solve for both.

3.7.4 Schnorr Signature Scheme

The Schnorr signature scheme is considered a simple and efficient signature scheme that sees widespread use. The algorithm works as follows.

First all users agree on a group G of prime order q with generator g. This group is typically a Schnorr group, but it is not required. They also agree on a hash function $H : \{0,1\}^* \to \mathbb{Z}_q^*$.

With these prerequisites each user can choose a private key $x \in \mathbb{Z}_q^*$ which leads to the corresponding public key $y = g^x$.

Signing a message M goes as follows. $r = g^k, k \in \mathbb{Z}_q^*$. Then $e = H(r \parallel M)$, with $\parallel$ representing concatenation and $r, M \in \{0,1\}^*$, the bitstring representations. Finally $s = k - xe$ leading to the signature (s, e) with $(s, e) \in \mathbb{Z}_p$.

Verification of the message is done by determining $r_v = g^s y^e$ and $e_v = H(r_v \parallel M)$. The signature is genuine if $e_v = e$.

To prove that the signature is indeed genuine if $e_v = e$, consider that $s = k - xe$. It follows that $r_v = g^s y^e = g^{k-xe} g^{xe} = g^k = r$. As such $e_v = H(r_v \parallel M) = H(r \parallel M) = e$.

Because G, g, q, y, s, e, r are all public and k, x private, it is easy to see that anyone can verify, but only the owner of k and x can have signed the message.

It is important to note that the security of the signature scheme relies on the security of the hash algorithm used. Fiat and Shamir have argued [7] that the algorithm is secure if H is modeled as a random oracle.

Seuring went on to show [15] that proof of security with the Forking lemma is the best possible result for the Schnorr signature scheme (among others).

The actual securiy proof is not discussed in this thesis, for it would broaden the scope too much.

3.8 RandHound

RandHound [18] is a protocol to create public, verifiable and unbiasable randomness. The protocol is modeled as a client/server model, wherein the

client requests a random value, that is generated by a set of RandHound servers. Each server in the set will generate his own secret, which is then shared with the client via a share that proves that the server has generated a secret according to the rules set, but that does not share the secret itself. After a certain time, the client commits to a certain set of secrets to use to create the random value. Having committed to the set, the client asks the servers to co-sign his choice. The servers then acknowledge the commitment, verify that it is genuine and respond with a Schnorr response. The client receives these responses and creates the aggregate Schnorr response, which he then presents to the servers in order to receive the actual secret. Each server then responds with his actual secret value, which the client finally combines to create the complete random value.

3.9 OmniLedger

OmniLedger [10] is presented as a scalable distributed ledger. It uses sharding to divide the workload over participants. OmniLedger uses RandHound to divide the participants over shards, so that the chance of any shard being compromised is negligible. Moreover, shards are periodically reformed, to further reduce the chance that they become compromised.

Chapter 4

Related Work

Boneh, Drijvers and Neven in [3] have devised a multisignature scheme using the signature scheme described by Boneh, Lynn and Sacham in [4]. The work in [3] desribes an efficient multisignature scheme and links it to use in Bitcoin, but is at its core an efficient algorithm.

Gregory Maxwell, Andrew Poelstra, Yannick Seurin and Pieter Wuille in [11] describe another multisignature scheme to be applied in Bitcoin, based on the well known Schnorr signature schemes. Their work descirbes a scheme that is simple and efficient and allows key aggregation to create a joint signature.

This thesis focuses on comparing the two algorithms, in particular regarding performance, and is as such a link between the two works, allowing consideration over the choice of one scheme over the other.

Chapter 5

Boneh-Lynn-Shacham Multisignature Scheme

5.1 Boneh-Lynn-Shacham Signature Scheme

The Boneh-Lynn-Shacham [4] (BLS) signature scheme, is a signature scheme utilising the Weil pairing to create signatures. The scheme uses the following.

- A bilineair pairing $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_T$ which is efficiently computable and non-degenerate. The three groups have prime order q and the generators of $\mathbb{G}_0$ and $\mathbb{G}_1$ are g_0 and g_1 respectively.
- A hash function $H_0 : M \to \mathbb{G}_0$ to hash the message M.

The scheme then has three phases. In the first phase the key is generated, in the second the message is signed and in the final phase the signature is verified.

- To generate the key, $\alpha \in \mathbb{Z}_q$ is picked randomly and then used to create $h = g_1^\alpha \in \mathbb{G}_1$. Here α is the private key, and h the public key.
- The message m is then signed by creating $\sigma = H_0(m)^\alpha \in \mathbb{G}_0$. The result is therefore a single group element.
- The signature is verified if $e(g_1, \sigma) = e(h, H_0(m))$ holds, otherwise the signature is a forgery.

To see that the verification holds, note that e is bilinear, and as such $e(g^x, g^y) = e(g, g)^{xy}$. Therefore it follows that $e(g_1, \sigma) = (g_1, H_0(m)^\alpha) = (g_1, H_0(m))^\alpha = e(h, H_0(m))$ and the signature is verified.

5.2 Algorithm

Boneh, Drijvers and Neven noted [3] that the intuitive multisignature algorithm using BLS, which aggregates signatures $\sigma_1, \ldots, \sigma_n$ by computing $\sigma = \sigma_1 \ldots \sigma_n$ can be easily attacked. An attacker can register his public key h_2 by using h_1 of another user, called Alice such that $h_2 = g_1^{\beta} \cdot h_1^{-1}$, where the attacker picks $\beta \in \mathbb{Z}_q$ himself.

The attacker can now present the signature $\sigma = H_0(m)^{\beta}$ for some message m and claim that it was signed by both him and Alice since it follows that $e(g_1, \sigma) = e(g_1, H_0(m)^{\beta}) = e(g_1^{\beta}, H_0(m)) = e(h_1 \cdot h_2, H_0(m))$, and that therefore the signature is genuine.

The BLS multisignature scheme that is resistant against this type of attack works slightly different, but uses the same principles. As added prerequisite, the BLS multisignature scheme requires a second hash function $H_1 : \mathbb{G}_1^n \to R^n$ with $R^n = \{2^0, \ldots, x^128\}$.

Key generation and message signing remain the same as in single use BLS, but the aggregation scheme works differently.

- The function H_1 is used to calculate $(t_1, \ldots, t_n) = H_1(h_1, \ldots, h_n) \in R^n$. These values are used to create the aggregate signature $\sigma = \sigma_1^{t_1} \ldots \sigma_n^{t_n} \in \mathbb{G}_0$.

- The signature σ is verified by computing $(t_1, \ldots, t_n) = H_1(h_1, \ldots, h_n) \in R^n$ and $h = h_1^{t_1} \ldots h_n^{t_n} \in \mathbb{G}_1$. The signature is genuine if $e(g_1, \sigma) = e(h, H_0(m))$.

Because the multisignature now requires genuine input from each participant, the attack on the previous scheme is no longer possible.

5.3 Security

The security of this scheme is based on the computational Diffie-Hellman assumption (co-CDH). This assumption, closely related to the discrete logarithm problem, is as follows.

Definition 5.3.0.1. Take group $\mathbb{G}$ of order q. Given (g, g^a, g^b) with g a random generator and $a, b \in \{0, \ldots, q-1\}$, it is infeasible to compute g^{ab}

The security proof shows that any attacker that manages to forge a message in the BLS multisignature scheme, can use that very same algorithm to break co-CDH, which contradicts the statement that it is infeasible to compute.

5.4 Performance

Since aggregation of the signatures only requires the individual signatures and the value $H_1(h_i)$ for signer i, the aggregation can happen as soon as all needed signatures are published. Specifically, this means that it is not needed for all signers to participate in the process at the exact same time, completely skipping any multi-round protocol between signers.

Futhermore, the BLS multisignature algorithm is set up in such a way that the aggregated public key can be computed without knowing the message m, since it only uses public values to calculate the key. This, of course, means that, if it is known who will be the signers, any party wishing to verify the signature can calculate the aggregate public key beforehand, allowing for greater performance.

Adding to this, the scheme also allows to check a set of aggregate signatures, to allow batch verification over different messages. Considering distinct messages $m_1, \ldots, m_b$ with corresponding signatures $\sigma_1, \ldots, \sigma_b$, calculating $\sigma = \sigma_1 \cdot \sigma_b$ allows to verify that $e(g_1, \sigma) = e(h_1, H_0(m_1)) \cdots e(h_b, H_0(m_b))$.

Chapter 6

Schnorr Multisignature Scheme

MuSig [11] is a Schnorr based multisignature scheme allowing for key aggregation with a resulting aggregated public key that functions just as in standard Schnorr schemes.

6.1 Algorithm

MuSig uses the following algorithm.

- A cyclic group $\mathbb{G}$ of order p and generator g.
- Hash functions $H_1, H_2, H_3 : \{0,1\}^* \to \{0,1\}^\lambda$, with λ representing the length of the output.

The scheme has the same three phases as BLS, key generation, message signing and signature verification.

- To generate the key, $\alpha \in \mathbb{Z}_p$ is sampled uniformly and randomly and used to create $h = g^\alpha$. Here α is the private key and h the public key.
- Signing a message m requires a set $L = \{h_1, \ldots, h_n\}$ representing all public keys involved in signing, with h_1 the current signer. The signer then calculates, for $i \in \{1, \ldots, n\}$, $\beta_i = H_2(L, h_i)$. The aggregated public key is defined as $H = \prod_{i=1}^{n} \alpha_i^{\beta_i}$.

  Afterwards, the signer samples $r_1 \in \mathbb{Z}_p$ uniformly and calculates $R_1 = g^{r_1}$ and $t_1 = H_1(R_1)$. He then sends t_1 to all other signers. Once all commitments $t_2, \ldots, t_n$ have been received, the signer sends R_1. Upon receiving $R_2, \ldots, R_n$ the signer verifies $t_i = H_1(R_i)$ before continuing.

  The signer can now compute $R = \prod_{i=1}^{n} R_i$, $c = H_3(H, R, m)$ and $s_1 = r_1 + c\beta_i\alpha_i \mod p$ and subsequently send s_1 to all cosigners. Upon

receiving $s_2, \ldots, s_n$, the signer can create $s = \sum_{i=1}^{n} s_i \mod p$. The resulting signature is $\sigma = (R, s)$.

- To verify signature σ on message m for the set of public keys $L = \{h_1, \ldots, h_n\}$, the verifier needs $\beta_i = H_1(L, h_i)$, $H = \prod_{i=1}^{n} \alpha_i^{\beta_i}$ and $c = H_3(H, R, m)$. The signature is verified if $g^s = r \prod_{i=1}^{n} h_1^{\beta_i c} = RH^c$.

6.2 Security

The security of MuSig is proven by extracting the discrete logarithm problem from the challenge of the public key. Hence, if someone would be able to get a private key from a public key to forge a key in such a manner, they would have found an algorithm to solve the discrete logarithm problem, which is considered infeasible.

6.3 Performance

The various hashing, group and other operations require very little computational power and time, and are as such not of interest when regarding the performance of the algorithm.

It must be noted that MuSig requires three rounds of communication, each requiring a connection to all participating parties. Hence the performance of the scheme is heavily gated by the network rate and availability.

Each of the three rounds requires connections to all other participants, which means that a succesful run of the scheme requires a constant connection between the participants in order to achieve the best possible performance. Assuming a maximum network latency γ and minimum latency δ, it is easy to see that MuSig has an additional time requirement of at least 6δ and at most 6γ.

Other than the round trips needed, there is not optimisation to be found in being able to calculate values in advance. Each step in the scheme after the first roundtrip requires new information and as such the performance of MuSig is heavily gated by the network it is operating in.

Chapter 7

Comparison and Conclusion

This chapter will compare the two discussed multisignature algorithms, and propose the best solution to use in RandHound. The comparison will focus on the performance of the algorithms, since a greater performance leads to an even better scalable OmniLedger.

7.1 Comparison

Considering the two algorithms, it is easy to see that the computational requirements barely differ. Key generation works similar and both signing and verifying require basic computational actions, mostly addition and multiplication and some use of hash functions.

A performance discrepency is found when regarding the signing part of the algorithm. MuSig requires a roundtrip between all participants thrice during each iteration of the algorithm. Each roundtrip suffers from network latency and network blackouts for one or more participants lead to failure of the algorithm. In this regard, BLS is clearly the superior choice. All steps of the algorithm, barring the aggregation, can be performed without a connection, only the publicly available details are needed. Even better, the algorithm doesn't fail if someone fails to respond, but simply waits until all responses have been gathered. This also gives BLS a significant edge in stability. It ensures that algorithms that have been started by a group can be completed at a later point. Therefore a network blackout does not require work to be done again or lead to a loss of data.

The security proof for both algorithms work similar, using the infeasibility of the discrete logarithm problem, or a related problem, to prove that any succesful attacker has found an algorithm to solve the aforementioned problem. Therefore any differences in security remain small, and should not

significantly impact the choice for either algorithm.

7.2 Conclusion

All in all the BLS multisignature scheme is the most suitable choice for RandHound. It boasts an algorithm that remains stable even under unstable network conditions, and offers the possibility for performance improvement by calculating data ahead of time and allows to check batches of multisignatures at the same time.

Bibliography

[1] Alex Edward Aftuck. *The Weil Pairing on Elliptic Curves and Its Cryptographic Applications*. 2011. URL: https://digitalcommons.unf.edu/etd/139/ (visited on 04/28/2018).

[2] Mihir Bellare and Gregory Neven. "Multi-signatures in the Plain public-Key Model and a General Forking Lemma". In: *Proceedings of the 13th ACM Conference on Computer and Communications Security*. CCS '06. ACM, 2006, pp. 390–399.

[3] Dan Boneh, Manu Drijvers, and Gregory Neven. 2018. URL: https://crypto.stanford.edu/~dabo/pubs/papers/BLSmultisig.html (visited on 05/01/2018).

[4] Dan Boneh, Ben Lynn, and Hovav Shacham. "Short Signatures from the Weil Pairing". In: *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*. 2001, pp. 514–532.

[5] Christian Cachin. *Resilient Consensus Protocols for Blockchains*. 2017. URL: https://www.ibm.com/blogs/research/2017/10/resilient-consensus-protocols-blockchains/ (visited on 05/09/2018).

[6] James C. Corbett et al. "Spanner: Google's Globally-Distributed Database". In: *ACM Trans. Comput. Syst.* 31.3 (Aug. 2013), 8:1–8:22.

[7] A. Fiat and A. Shamir. "How to prove yourself: practical solutions to identification and signature problems". In: *Advances in Cryptology: CRYPTO '86*. 1987, pp. 186–194.

[8] Misha Kapovich. *The Riemann-Roch Theorem*. 2007. URL: https://www.math.ucdavis.edu/~kapovich/RS/RiemannRoch.pdf (visited on 04/25/2018).

[9] Craig Kerstiens. *Database sharding explained in plain English*. 2018. URL: https://www.citusdata.com/blog/2018/01/10/sharding-in-plain-english/ (visited on 05/16/2018).

[10] Eleftherios Kokoris-Kogias et al. *OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding*. 2017. URL: https://eprint.iacr.org/2017/406 (visited on 04/20/2018).

[11] Gregory Maxwell et al. *Simple Schnorr Multi-Signatures with Applications to Bitcoin*. 2018. URL: https://eprint.iacr.org/2018/068 (visited on 04/30/2018).

[12] David Mazières. *On Worldwide Consensus*. 2015. URL: https://www.stellar.org/papers/stellar-consensus-protocol.pdf (visited on 05/16/2018).

[13] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2008. URL: https://bitcoin.org/bitcoin.pdf (visited on 05/16/2018).

[14] Claus P. Schnorr. *Method for identifying subscribers and for generating and verifying electronic signatures in a data exchange system*. 1989. URL: https://patents.google.com/patent/US4995082 (visited on 05/02/2018).

[15] Yannick Seurin. *On the Exact Security of Schnorr-Type Signatures in the Random Oracle Model*. 2012. URL: https://eprint.iacr.org/2012/029 (visited on 05/02/2018).

[16] Joseph H. Silverman. *An Introduction to the Theory of Elliptic Curves*. 2006. URL: https://www.math.brown.edu/~jhs/Presentations/WyomingEllipticCurve.pdf (visited on 04/27/2018).

[17] Joseph H. Silverman. *The Arithmetic of Elliptic Curves*. Springer, New York, NY, 2009.

[18] Ewa Syta et al. *Scalable Bias-Resistant Distributed Randomness*. 2016. URL: https://eprint.iacr.org/2016/1067 (visited on 04/20/2018).

[19] Mark Walport. *Distributed Ledger Technology: beyond block chain*. 2016. URL: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/492972/gs-16-1-distributed-ledger-technology.pdf (visited on 05/02/2019).