Comparison of multi-signature schemes for performance improvement of OmniLedger

Bachelor thesis for the department of Mathematics of Utrecht University

Student

Paul van Grol

Utrecht University

Primary supervisor

Veelasha Moonsamy

Utrecht University

Secondary supervisor

Damaris Schindler

Utrecht University

June 9, 2018

Utrecht University

Abstract

To be added, do not forget!!

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 multi-signature algorithms that may be used to improve the performance of RandHound [19]. 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.

Since RandHound uses a multi-signature scheme as part of its algorithm, which constitutes a large part of the running time of the algorithm as a whole, this thesis focuses on improving RandHound by improving the multi-signature algorithm used.

Furthermore, OmniLedger also makes use of CoSi [20] in the consensus protocol, which also makes use of multi-signatures, so the same performance improvement can be achieved there by improving the multi-signature algorithm.

Therefore the research question of this thesis is as follows: "Which multi-signature scheme is most suitable for performance improvement of OmniLedger". To answer this question, this thesis researches the performance and security requirements of two multi-signature schemes. MuSig [11] is based on Schnorr signature schemes and BLS multi-signature [3] is based on the BLS [4] sig-

nature scheme.

To properly answer these questions some mathematical and technical background is needed.

The second chapter, Mathematical Background, introduces algebraic varieties, leading up to an explanation of elliptic curves and the Weil pairing on elliptic curves. It further introduces the discrete logarithm problem and the elliptic curve discrete logarithm problem. Elliptic curves, the Weil pairing and the (elliptic curve) discrete logarithm problem are used to great effect in cryptography.

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

Chapter four, Related Work, discusses related works, such as the work on OmniLedger [10], RandHound [19], MuSig [11] and BLS [3] and its multi-signature variant [3]. It further explains the contribution of this thesis to the existing works.

In chapters five, BLS Multi-Signature Scheme, and six, Schnorr Multi-Signature Scheme, the two multi-signature schemes are studied.

This study leads chapter seven, Comparison and Conclusion, in which the two schemes are compared and the research question is answered.

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 [18, Chapter 1-3], along with some of his slides [17].

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 [18, 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*

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

4. *The pairing is Galois invariant*

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

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 [18, page 96] but are not of particular interest in this thesis.

2.3 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 2.3.0.1. 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 2.3.0.2 (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 2.3.0.3 (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 2.3.0.4 (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.

2.4 Schnorr Group

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

Definition 2.4.0.1 (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 2.4.0.2 (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 cryptographic purposes, 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.

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 fulfils 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, hence making 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.

In the protocol used by Bitcoin 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 refuse 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 disincentivesed 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 [21] (DL) can be considered as 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.

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 system 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 (see 3.1) 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 base, 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 focuses 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, i.e. 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 simultaneously, which improves throughput and consequently performance. At the same time, it also means that the amount of resources required remains limited, since it is not the whole system that has to act, but just a part of it.

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.

In contrast to traditional blockchain technologies, sharding in a blockchain means dividing your participants into shards [10]. Transactions will be handled by the shard(s) 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 simultaneously.

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 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.

3.5 Forking Lemma

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

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

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-negligible if acc is non-negligible. In other words, if the underlying problem is hard, then no adversary can forge signatures.

3.6 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 and 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. A signature scheme is a way to present and verify the authenticity of a digital message. Signature schemes work with a pair of private and public key. 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 signature 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 multi-signatures, which are discussed below. A group signature scheme allows a signer to sign on behalf of a group, whereas a multi-signature scheme allows a group to sign a document, with the result begin a joint signature that is more compact than all individual signatures separately.

3.6.1 Group Signature Scheme

As explained just before, a group signature scheme allows a user to sign a message on behalf of a group. There exist various implementations of group signature schemes that each offer different behaviour. It is possible to have a group signature be created in such a way that the signer may be determined, or 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.

3.6.2 Multi-Signature Scheme

A multi-signature scheme is somewhat of a generalisation of a group signature scheme. A multi-signature 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.

3.6.3 Schnorr Signature Scheme

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

Prerequisites

All users agree on a group G of prime order q with generator g. This group is typically a Schnorr group (see 2.4), but is is not required. The users also

agree on a hash function $H : \{0,1\}^* \to \mathbb{Z}_q^*$.

Key Generation

Each user now chooses a private key $x \in \mathbb{Z}_q^*$ and determines the corresponding public key $y = g^x$.

Message Signing

A message $M \in \{0,1\}^*$ is signed as follows. The signer takes $k \in \mathbb{Z}_q^*$ and calculates $r = g^k \in \{0,1\}^*$. Here M and r are represented as bit strings.

Then r and M are concatenated (represented by $\|$) and subsequently hashed, so that $e = H(r \parallel M)$.

Finally e is used to determine $s = k - xe$ and the signature $(s, e) \in \mathbb{Z}_p$ is produced.

Signature Verification

The signature is verified by first calculating $r_v = g^s y^e$ and $e_v = H(r_v \parallel M)$ and then checking that $e_v = e$.

Proof of Signature Verification

To see that signature verification holds true, consider that $s = k - xe$. It follows that

$$r_v = g^s y^e = g^{k-xe} g^{xe} = g^k = r.$$

Therefore

$$e_v = H(r_v \parallel M) = H(r \parallel M) = e$$

and the verification check holds.

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 produce the signature.

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 modelled as a random oracle.

Seuring has shown [16] 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.

3.7 Public Verifiable Secret Sharing

A Public Verifiable Secret Sharing (PVSS) [15] algorithm is an algorithm such that any participating party can verify the validity of all shares.

In other words, in a PVSS scheme each participant generates a secret, and shows a share to others to prove that they have generated the secret. When the actual secrets are shared, anybody can verify that the secret has not been altered.

3.8 RandHound

RandHound [19] is a protocol to create public, verifiable and unbiased randomness. The protocol is modelled 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 generates its own secret value and creates shares for the other participants in the set via a PVSS scheme (see 3.7). The shares generated by the servers are sent to the client, who picks a subset of the servers to use, which fixes the output.

Having picked the subset, the client asks the servers to co-sign his choice, which is done with the help of a multi-signature scheme (see 3.6.2).

Each server, after acknowledging and verifying the commitment of the client, sends a Schnorr response (see 3.6.3) to the client.

The client creates the aggregate Schnorr response from all received responses and presents it to the servers to receive the actual secret from each server.

Each server verifies the aggregate Schnorr response and then returns the secret.

Finally the client can now compute the random value using the received secrets. The resulting value is verifiable and unbiased.

3.9 OmniLedger

OmniLedger [10] is presented as a scalable distributed ledger system that remains decentralised and secure. Furthermore it promises performance on par with centralised payment processors, such as Visa.

OmniLedger achieves scalability by using sharding (see 3.3) to divide the workload over participants. To ensure that the chance that any shard is or becomes compromised remains negligible, RandHound (see 3.8) is used to divide the participants over shards. Moreover, the shards are periodically

reformed, again with use of RandHound, to not compromise the long-term stability of a shard.

For its consensus protocol, OmniLedger uses CoSi [20], a scalable witness cosigning protocol that makes use of multi-signatures. Although many such schemes can be used for CoSi, the creators of CoSi have chosen to use a Schnorr-based multi-signature scheme, which is therefore also used in OmniLedger.

The level of performance is achieved by processing all non-conflicting transactions in parallel and to let all transactions only be handled by the shard(s) that it affects. Finally OmniLedger boasts an architecture to quickly verify low-value transactions according to a "trust-but-verify" mechanism, in which a transaction is at first quickly, and afterwards thoroughly checked. If the subsequent check shows dishonest behaviour of the first validator, this validator can be punished and the defrauded customers can be compensated. Since this mechanism is only used on low-value transactions and punishment will occur for dishonest behaviour, acting dishonestly does not offer any benefit for the first validator.

Chapter 4

Related Work

In this chapter various related works regarding multi-signature algorithms and their contributions are discussed.

The work on OmniLedger [10] by Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta and Bryan Ford uses CoSi [20] in the consensus protocol. CoSi uses a Schnorr-based multi-signature scheme, and OmniLedger has used CoSi as such, not looking for improvement on the algorithm.

Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J. Fischer and Bryan Ford in [19] state that they rely on Schnorr-based multi-signature schemes, but do not give options or arguments for other multi-signature schemes. As such their work simply shows that Schnorr-based multi-signature schemes are used in other applications, but it gives no indication towards the performance in comparison to other multi-signature schemes.

Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi and Bryan Ford [20] in their work on CoSi have stated that, although they chose for a Schnorr-based multi-signature scheme, any multi-signature scheme with efficient public key and signature aggregation could be used. The choice for a Schnorr-based scheme was made since such a scheme is simple and well-understood. They did note that a BLS [4] based scheme might be more desirable in a unstable or asynchronous situations, but they did not focus on the performance of either scheme.

In their work on the BLS multi-signature scheme [3], Dan Boneh, Manu Drijvers and Gregory Neven briefly compare their scheme to Schnorr-based multi-signature schemes, but their comparison does not focus on performance. It is mentioned that their scheme, as opposed to Schnorr-based schemes, does not require multiple rounds of communication, but that is the

only comparison regarding performance. They mention that their scheme allows for public aggregation via simple multiplication long after signatures have been generated, as opposed to the Schnorr-based multi-signatures that can only be aggregated at the time of signing. Furthermore they give applications of their scheme in crypto currencies, but with a focus on shrinking the transaction size by limiting the size of the signature, not regarding performance.

Gregory Maxwell, Andrew Poelstra, Yannick Seurin and Pieter Wuille in [11] describe a Schnorr-based multi-signature scheme that is both simple and efficient, and allows for key aggregation to create a joint signature. They compare their work to existing Schnorr-based multi-signature algorithms to show that their algorithm is an improvement in regards to performance and simplicity.

The contribution of this thesis, is that it compares two existing multi-signature algorithms, BLS multi-signature [3] and MuSig [11] with a focus on performance. This contributes to any future works where the performance of a multi-signature scheme is an important aspect.

Chapter 5

Boneh-Lynn-Shacham Multi-Signature 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 bilinear 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 [3] noted that the intuitive multi-signature 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 multi-signature scheme that is resistant against this type of attack works slightly different, but uses the same principles. As added prerequisite, the BLS multi-signature scheme requires a second hash function $H_1 : \mathbb{G}_1^n \to R^n$ with $R^n = \{2^0, \ldots, x^1 28\}$.

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 multi-signature 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) [1]. 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 multi-signature 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.

Furthermore, the BLS multi-signature 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 Multi-Signature Scheme

MuSig [11] is a Schnorr-based multi-signature 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 successful 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.

MuSig does also not allow for any values to be calculated in advance. After the first round-trip, each step in the scheme requires new information that must be acquired via network communication, 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 BLS (see 5) and MuSig (see 6) multi-signature algorithms and propose the best solution to improve OmniLedger. The performance will briefly touch upon the security requirements of both schemes, but focuses mostly on the performance of the schemes. The focus of this thesis is, in the end, on improving the performance of OmniLedger to ensure even better scalability.

7.1 Comparison

Both of the algorithms use the discrete logarithm problem, or a related problem, to prove the security of the scheme. Both proofs of security show that, in order for the attacker to successfully attack the system, he must have found an algorithm that can be used to solve the discrete logarithm problem in the case of MuSig, or one that can be used to solve the computational Diffie-Hellman problem in the case of the BLS multi-signature scheme. Because both schemes use, at the basis, the same proof of security, the differences in security are marginal at best.

Considering the computational requirements of both schemes, it is easy to see that they operate in a similar manner. Key generation, signing and verification require basic computational actions that require very little in terms of computational power and the computational requirements do therefore not significantly impact the performance of either scheme. It must be noted that the BLS multi-signature algorithm allows for aggregation of the public key even before the message that must be signed is known, which allows for some performance optimisation. Furthermore, the BLS multi-signature scheme also allows for batches of multi-signature algorithms to be checked at the same time, allowing for even more optimisation.

However, in the part of the algorithm where a message is signed, a rather significant performance discrepancy is found. MuSig requires three round-trips of communication during this part of the scheme, whereas BLS only uses one. Moreover, MuSig requires that all participants are on-line when creating the multi-signature, since the three round-trips each require in- and output from all participants before the algorithm can continue. This means that each round of communication not only impacts the performance by the way of network latency, but can also be the cause for critical failure if the network for one or more participants fails at any point during the message signing. Contrary, BLS can operate even under unstable networks, since it allows all steps barring the aggregation of the final signature to be undertaken without communication. This means that all participants in the BLS multi-signature scheme can produce their part of the signature when they are able to do so, and that the aggregated signature can be computed as soon as every participant has published his part. This means that the network latency only impacts the algorithm only once, and that network blackout does not cause critical failure.

7.2 Conclusion

Because both schemes use a very similar proof of security and remain similar in terms of computational requirements, it suffices to compare the performance directly to pick the most suitable multi-signature scheme.

The BLS multi-signature scheme requires only one round-trip of communication, minimising the performance loss because of network latency, and does not suffer critical failure upon network blackout during the signing process. Since the scheme also offers performance optimisation, it is the superior choice for a multi-signature algorithm to improve the performance of RandHound and CoSi, thereby improving the performance of OmniLedger.

Bibliography

[1] Feng Bao, Robert H. Deng, and Huafei Zhu. Variations of diffie-hellman problem, 2003. `https://static.aminer.org/pdf/PDF/000/314/734/variations_of_diffie_hellman_problem.pdf`, visited 08-06-2018.

[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*, pages 390–399, 2006.

[3] Dan Boneh, Manu Drijvers, and Gregory Neven. Bls multi-signatures with public-key aggregation, 2018. `https://crypto.stanford.edu/~dabo/pubs/papers/BLSmultisig.html`, visited 01-05-2018.

[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*, pages 514–532, 2001.

[5] Christian Cachin. Resilient consensus protocols for blockchains, 2017. `https://www.ibm.com/blogs/research/2017/10/resilient-consensus-protocols-blockchains/`, visited 09-05-2018.

[6] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. Spanner: Google's globally-distributed database. *ACM Trans. Comput. Syst.*, 31(3):8:1–8:22, August 2013.

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

[8] Misha Kapovich. The riemann-roch theorem, 2007. `https://www.math.ucdavis.edu/~kapovich/RS/RiemannRoch.pdf`, visited 25-04-2018.

[9] Craig Kerstiens. Database sharding explained in plain english, 2018. `https://www.citusdata.com/blog/2018/01/10/sharding-in-plain-english/`, visited 16-05-2018.

[10] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding, 2017. `https://eprint.iacr.org/2017/406`, visited 20-04-2018.

[11] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple schnorr multi-signatures with applications to bitcoin, 2018. `https://eprint.iacr.org/2018/068`, visited 30-04-2018.

[12] David Mazières. On worldwide consensus, 2015. `https://www.stellar.org/papers/stellar-consensus-protocol.pdf`, visited 16-05-2018.

[13] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. `https://bitcoin.org/bitcoin.pdf`, visited 16-05-2018.

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

[15] B. Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In *IACR International Cryptology Conference (CRYPTO)*, page 784, 1999.

[16] Yannick Seurin. On the exact security of schnorr-type signatures in the random oracle model, 2012. `https://eprint.iacr.org/2012/029`, visited 02-05-2018.

[17] Joseph H. Silverman. An introduction to the theory of elliptic curves, 2006. `https://www.math.brown.edu/~jhs/Presentations/WyomingEllipticCurve.pdf`, visited 27-04-2018.

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

[19] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J. Fischer, and Bryan Ford. Scalable bias-resistant distributed randomness, 2016. `https://eprint.iacr.org/2016/1067`, visited 20-04-2018.

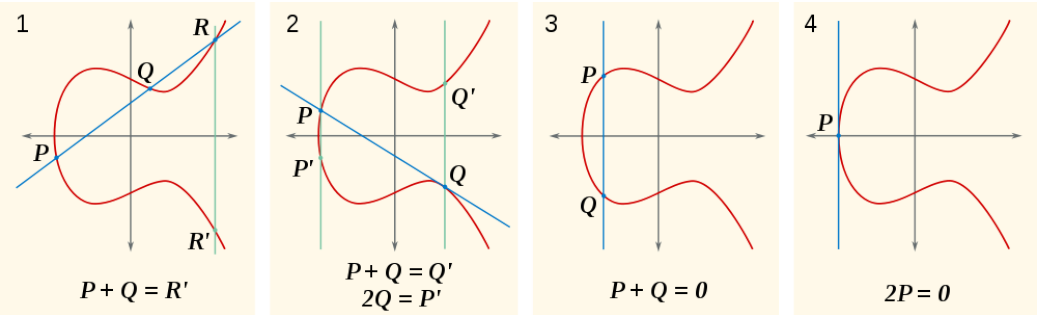
[20] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. Keeping authorities "honest or bust" with decentralized witness cosigning. In *37th IEEE Symposium on Security and Privacy*, 2016.

[21] Mark Walport. Distributed ledger technology: beyond block chain, 2016. `https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/492972/gs-16-1-distributed-ledger-technology.pdf`, visited 02-05-2018.

Appendix A

Elliptic Curves

This appendix contains geometrical explanations and proofs for the group law on elliptic curves.

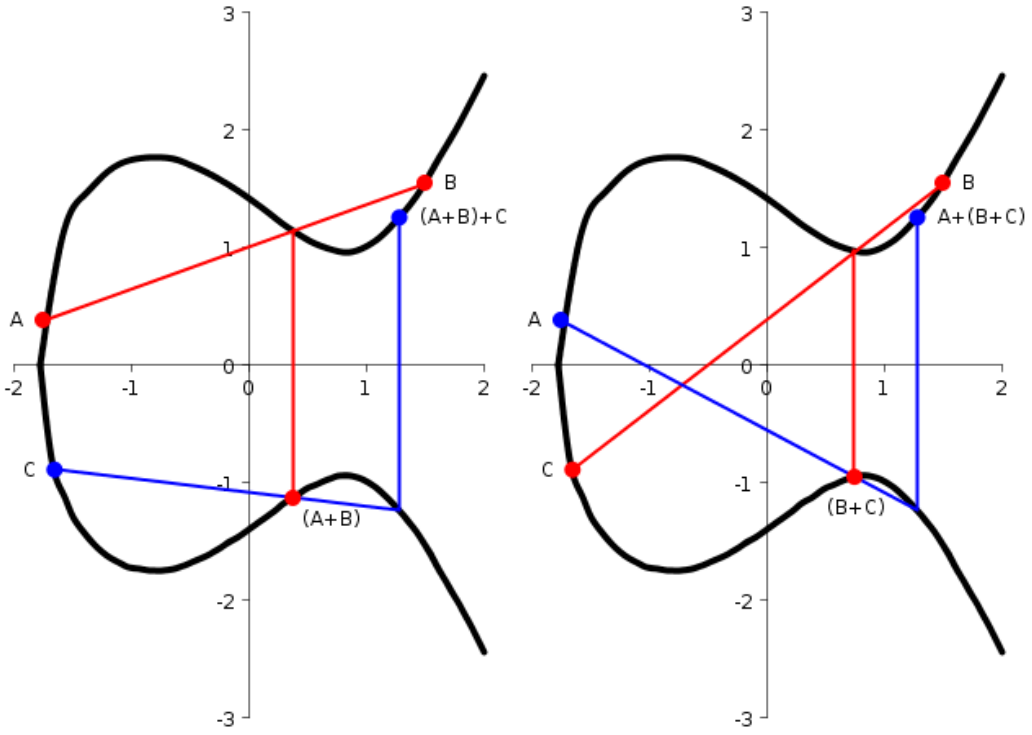


The leftmost picture, labeled with 1 shows that $P + Q = Q + P$, for it is easy to see that the line through P and Q remains the same.

The second picture (2) shows an expansion on the addition, wherein $Q+Q$ is calculated by taking the tangent to Q, arriving at P which results in $Q + Q = P'$.

The third picture (3) shows both the existence of an inverse element, and the existence of a neutral element. Consider that any vertical line ends in the point O, the point at infinity. It is easy to see that the line through O and P ends up having a third point of intersection in Q, and that therefore $P + O = P$. Since it's already been shown that $P + O = P = O + P$ and therefore O is the neutral element. Furthermore it is easy to see that Q is the inverse element of P, since $P + Q = O$. Therefore it may be stated that $Q = -P$ and the existence of the inverse element has been proven.

The fourth picture (4) shows an interesting case, where the point on the curve at $x = 0$ results in a point that is its own inverse, since $2P = O$ and $P - P = O$ so therefore in this case $P = -P$.



Finally associativity is shown by the two figures above. The left picture shows $(A+B)+C$, and the right picture shows $A+(B+C)$. It is easy to see that the points A, B, C remained the same, and that, indeed, $(A+B)+C = A+(B+C)$.