Linearly Homomorphic Signature Schemes

A Fair Comparison

D.A. Hondelink

TU Delft

Linearly Homomorphic Signature Schemes

A Fair Comparison

by

D.A. Hondelink

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Wednesday, June 22, 2022 at 3:00 PM.

Student number:	4466683	
Project duration:	March 1, 2021 – June 22, 2022	
Thesis committee:	Dr. Z. Erkin,	TU Delft, supervisor
	Dr. R. Venkatesha Prasad,	TU Delft
	Dr. T. Abeel,	TU Delft

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Abstract

Digital signatures are used everywhere around us. They are well-studied and have been standardized since 1994. In 2002, Johnson et al. introduced the notion of *homomorphic* digital signatures, allowing one to perform computations on signed data. These signatures are especially useful for linear network coding, a technique used to improve throughput and resilience of networks, and in verifiable cloud computing. However, homomorphic signatures are not standardized and less well-studied, which creates a challenge when choosing one of the published schemes. Moreover, most schemes remain unimplemented, and it is insufficient to compare their theoretical performance for real-world applications. Those schemes that are implemented are not directly comparable since they use different instantiations of the same primitives or they are implemented in different programming languages.

In this thesis, we set out to find out how we can assess the performance of homomorphic signature schemes. To this end, we have implemented eleven pairing-based linearly homomorphic signature schemes. All signature schemes have been implemented on the BLS12-381 curve to constitute a fair comparison. We assess the performance of the signature schemes based on their signing, verifying and combining performance, as well as the sizes of their keys and signatures. Furthermore, we analyse the impact that additional features such as supporting a multi-party setting have on the performance of a signature scheme. Based on our experimental results, we make recommendations for three types of applications: For constrained devices, the scheme Li20 is the most suitable due to its compact signing key and efficient signing operation. For network coding, which requires fast verification, fast combining, and small signatures, we also recommend the scheme by Li et al. Finally, for a multi-party, privacy-preserving scheme, we recommend the scheme by Sch18 and Sch19, which preserve input privacy of homomorphically combined signatures. We find that we can assess the performance of a homomorphic signature scheme based on the speed of the signing, verifying and combining operation. Our implementation is publicly available.

Preface

In this thesis, I present a comparison on the performance of pairing-based linearly homomorphic signature schemes. This comparison is intended to aid the process of selecting a signature scheme to be used in a real-world scenario. On the cover of this thesis, we find a homomorphic signature. It is produced by signing the individual words of the title of the thesis, after which we homomorphically combined them.

Working on this thesis has been a long and very educational journey. It is the final chapter of my time as a student in Delft. This journey started back in 2015, together with Jaap, Jip and Rolf. I want to thank them for the great times we had together as computer science students, and as friends. I want to thank Freek for our nice coffee breaks while studying at the library. It is because of friends like them that my time in Delft has been nothing but a joy.

I want to thank my supervisor, Dr Zeki Erkin, for his guidance and support during my thesis, and whose classes piqued my interest in cryptography and privacy. Motivating me to write a research paper during my thesis project was a very valuable experience. The paper is currently awaiting review, but writing and submitting it has already felt like a great accomplishment. I also want to thank Jelle for helping me write the paper and for his valuable tips on this report. I want to thank Dr Thomas Abeel and Dr RangaRao Venkatesha Prasad for being part of my thesis committee.

Finally, I want to thank my family, friends and my girlfriend for their continuous love and support throughout the years.

Dieuwer Hondelink
Rotterdam, June 2022

Glossary

BLS Barreto-Lynn-Scott

BN Barreto-Naehrig

CDH Computational Diffie-Hellman

CH Context Hiding

CSP Cloud Service Provider

CSPRNG cryptographically Secure Pseudo-Random Number Generator

DDH Decisional Diffie-Hellman

DE Designated Entities

DL Discrete Logarithm

ECC Elliptic Curve Cryptography

Homomorphic Signature SHS

HS Homomorphic Signature

HSS Homomorphic Signature Scheme

ID Identity

IETF Internet Engineering Task Force

KGC Key Generation Centre

LHSS Linearly Homomorphic Signature Scheme

MAC Message Authentication Code

MK Multi-Key

NC Network Coding

NCS Network Coding Signature

NIST National Institute of Standards and Technology

PRF Pseudo Random Function

RNG Random Number Generator

ROM Random Oracle Model

MTU Maximum Transfer Unit

List of Figures

List of Tables

Contents

1 Introduction

Digital signatures are used everywhere around us. In most instances, they are hidden from the end-user, but behind the scenes of bank transactions and computer networks, these signatures provide trust and authenticity to digital systems. In some other instances, we explicitly want signatures to be visible, think for example of a signed contract. We want to see that this document is signed, and as such, a digital mark is placed on it.

1.1. Digital Signatures

Digital signatures offer more functionality than a classic signature written with a pen on a piece of paper. A signed paper document can be altered after a signature has been placed, and a signature can be studied and copied, to later make forged signatures to impersonate someone.

Digital signatures, on the other hand, are constructed in such a way that this is not possible (by definition even). A digital signature is created on a message, using a secret signing key. The outcome of signing a message is a list of bytes which are the result of computation involving this signing key and message, instead of a drawing representing someone's name or initials. When someone wants to check the validity of this signature, he needs a verification key that matches the secret signing key, the signature and the message it was produced on. If everything is correct, the verification of the signature succeeds.

If, however, the message was changed after signing, the signature will no longer verify. Further, since the signature is produced using a secret signing key, only the person holding that key can create valid signatures.

The invention of digital signatures can be dated back to 1976 [17] when Diffie and Hellman presented their public key solution. Since then, numerous digital signature schemes have been invented and studied, and the use of digital signatures has been standardized since 1994.

1.2. Homomorphic Signatures

In 1993, Desmedt [16] thought of a concept of signatures that could be combined based on operators. Later in 2002, Johnson et al.[29] formalized this notion as homomorphic signatures.

Homomorphic signatures allow us to compute on signed data. We can thus first sign data,

and then run calculations on the data *and* on the corresponding signatures, to get a signature that verifies on the computed value.

The calculations that can be run on these signatures can be linear, polynomial, or fully homomorphic. As such, there exist linearly homomorphic, polynomial homomorphic and fully homomorphic signature schemes.

Linearly homomorphic signatures are especially useful for *network coding* [2]. This is a technique that through linear combinations [32] of packets provides robustness and resilience to computer networking. When a node wants to transmit a file through a network, it splits this file up into smaller packets, which it then sends to intermediate nodes. Instead of storing and forwarding single packets, in network coding, separate packets are combined (coded) together. This increases the throughput of networks, as fewer packets have to be sent. When a target node has received enough linearly independent packets, it can decode them to retrieve the original file.

Network coding suffers from an issue called *pollution attacks* [30]. When a node sends a corrupt packet instead of a linear combination of a received packet, this prevents the entire file from ever being decoded. This is where homomorphic signatures can provide value. As the source node signs the data packets and sends the packets accompanied by signatures, intermediate nodes can verify the authenticity and correctness of the packets they receive. A corrupted packet is instantly detected and discarded, while coding signatures together is possible due to the homomorphic property. This allows the next node to verify the legitimacy of the combined packet with the combined signature.

Another use case of homomorphic signatures is verifiable cloud computing. In this setting, data owners offload computational work to a cloud service provider (CSP). The reason for outsourcing this task can be due to the limited computational resources a data owner might have, making it infeasible to do the calculations themself. A CSP, on the other hand, does have the means to run a complicated calculation on this data. An issue is, however, that the data owner might not trust the CSP. For all the data owner knows, the CSP might alter the results of the computation, or even just answer with a random result. To be guaranteed of the legitimacy of the operations performed by the CSP, homomorphic signatures can be used. The data owner thus signs his data before sending it to the cloud service provider. The CSP in turn runs the requested calculation on the data, and also combines the signatures in the process. This way the CSP can prove to the data owner that it has indeed done the correct computation.

1.3. Research Questions

Over the years, a multitude of homomorphic signature schemes supporting various levels of homomorphism has been presented. While a lot of work has been put into studying these signature schemes, their real-world application and implementation have lacked behind when compared to digital signatures. Where digital signatures are standardized and extensively tested, no such standards are available for homomorphic signature schemes. Furthermore, some homomorphic signature schemes offer additional features, such as supporting multi-party computation or identity-based key generation. These extra features make it less straightforward to find out which scheme has a better performance. This makes choosing a homomorphic signature scheme for an application, not a trivial task. To combat this issue, we want to compare signature schemes and find out which scheme has the best performance.

Formally, in this thesis, we set out to answer the following research question:

How can we assess pairing-based, linearly homomorphic signature schemes to be used in practice, based on their performance, and the size of their signatures and keys?

To help answer this question, we formulate the following sub-research questions:

- RQ_1 What is the fastest pairing-based, linearly homomorphic signature scheme, when run on modern hardware?
- RQ_2 Which scheme is the best to use in one of the following scenarios?
  - Network Coding
  - Multi-party computation
  - Privacy preserving
- RQ_3 What is the performance cost of supporting one of the following features?
  - Network Coding
  - Multi-party computation
  - Context hiding
  - Identity-based key generation
  - Designated entities

To answer these questions, we implement and study eleven homomorphic signature schemes in this work. We focus on pairing-based signature schemes supporting linear operations. To facilitate a fair comparison, all schemes have been implemented under the same constraints.

1.4. Contributions

To list our contributions, we:

1. Implement and open-source eleven pairing-based, linearly homomorphic signature schemes[1].
2. Analyse and compare the performance of these signature schemes.
3. Make recommendations on which scheme to use in which scenario.

1.5. Overview of document

The rest of this document is structured as follows: In chapter 2 we present the preliminary knowledge required to discuss pairing-based linearly homomorphic signatures. In chapter 3, we discuss related works on homomorphic signatures. In chapter 4 we explain in detail what a homomorphic signature is, and how one is created. In chapter 5, we present the details of our implementation and discuss our measuring method. Then in chapter 6, we present our results. After that, we apply our findings to three real-world scenarios and explain which choices lead to which decision in chapter 7. Finally, we reflect on our work, make recommendations for future works and conclude in chapter 8.

[1] Our implementation can be found at `https://gitlab.com/dieuwerh/lhss`

2

Preliminaries

In this section, we go through all the building blocks that are required in the construction of pairing-based linearly homomorphic signature schemes. We discuss the cryptographic primitives using which the signature schemes are created, as well as the assumptions on which the security of signature schemes relies.

2.1. Cryptosystems

A distinction between two kinds of cryptosystems can be made. They regard which person holds which keys of the signing or encrypting algorithm. We briefly explain the two cryptosystems and show in which way they differ.

2.1.1. Symmetric Cryptography

In symmetric-key cryptography, every entity has a single key. This same key is used to both encrypt and decrypt messages or to sign and verify a message.

For a key k and a message m:

$$\begin{aligned} c &= Enc(m, k) \\ m &= Dec(c, k) \end{aligned} \tag{2.1}$$

In symmetric cryptography, signatures are called message authentication codes (MACs). A MAC on a message is also known as a tag, instead of a signature. To produce a MAC, the MACing algorithm is used on a message m together with the symmetric key k:

$$\tau = MAC(m, k) \tag{2.2}$$

2.1.2. Asymmetric Cryptography

In asymmetric cryptography (also known as public-key cryptography, on the other hand, each entity (person) possesses a pair of keys. A *key-pair* consists of a secret key *sk* only known by that entity, and a public key *pk* which everyone knows belongs to that person. In the context of encryption, a public key can be used by anyone who has access to it to encrypt a message for someone. Anyone can create an encrypted message destined for someone, but only the person holding the corresponding secret key can decrypt those cipher texts.

For a key-pair (sk, pk) and a message m:

$$\begin{aligned} c &= Enc(m, pk) \\ m &= Dec(c, sk) \end{aligned} \tag{2.3}$$

For signatures, this works just the other way around. Only the person with the signing key can create a signature, while anyone with the public verification key can verify its legitimacy:

$$\begin{aligned} \sigma &= Sign(m, sk) \\ 1 &= Ver(\sigma, vk) \end{aligned} \tag{2.4}$$

2.2. Digital Signatures

To discuss signatures, we must first have a clear understanding of what they are and what they should offer. We will start with an intuition which we will explain by an example. Suppose Bob has promised to pay Alice 50 euros. To make it official, Bob writes down on a piece of paper ”I, Bob, promise to pay Alice 50 euros”. Bob then signs the message with his signature and gives it to Alice as proof. In this example, Bob’s signature is used as a mark of proof that Bob agrees to the terms stated on the document.

After Bob has signed the message, he should not be able to deny ever having claimed he said it. Otherwise, a signature would not be worth anything and it would not offer any value to Alice.

After Bob has given the signed document to Alice, we do not want her to be able to change the signed message. If this was possible, she could add a zero after the stated amount and can claim that Bob owes her 500 euros. So another requirement for a signature is that it is only valid for the original message. When the message changes, the signature becomes invalid.

Lastly, we do not want anyone but Bob to be able to create this signature. If anyone was able to impersonate Bob, he would go bankrupt shortly.

To capture this intuition in a more formal sense, we require that a signature offers:

Authentication A signature for entity A can only be created by using entity A’s signing key.

Integrity After a signature σ on message m has been created, σ will not verify after changes are made to m.

Non-repudiation After entity A has signed a message, he should not be able to deny ever signing it.

Digital signatures are built upon asymmetric cryptography. Every participating party possesses a key pair consisting of a signing and a verification key.

A digital signature scheme consists of three algorithms:

$(sk, vk) \leftarrow$**KeyGen**(λ) On input of a security parameter λ, the key generation algorithm produces a key pair consisting of a private signing key sk and a public verification key vk.

$\sigma \leftarrow$**Sign**(m, sk) The signing algorithm produces a signature σ on a message m using a supplied signing key sk.

$b \leftarrow$ **Verify**(m, σ, vk) The verify operation checks the validity of a supplied signature σ, message m and verification key vk. It outputs a bit b indicating success (1) or failure (0).

2.3. Number theory

Constructions of cryptographic protocols often work on 'groups' and 'fields'. These are different from regular numbers. We describe below what each of these items is. The reason for using groups and fields is because there are certain problems which are known to be hard for these structures. By applying the hardness of solving these problems, we can construct cryptographic protocols which we can prove are secure.

2.3.1. Groups

A group is a set of numbers for which one binary operation is defined, for example, addition (resulting in additive groups), or multiplication (resulting in multiplicative).

A group must satisfy the following properties:

Associativity Rearranging the order of operations has no impact on the result of calculation: $a \times (b \times c) \Leftrightarrow (a \times b) \times c$.

Identity There exists an identity element I, such that when the group operation is applied to a number with this element, the number is returned: $a \times I = a$.

Inverse For every element in the group, there exists an inverse element such that when the group operation is applied to a number and its inverse, the identity element is returned: $a \times a^{-1} = I$.

If the group additionally has the following property, $a \times b = b \times a$, the group is called **commutative**.

2.3.2. Fields

A field F is a set of numbers for which two binary operations $+$ and $\times$ are defined. A field must satisfy the following properties:

- $(F, +)$ is a commutative group with identity 0.
- $(F*, \times)$ is a commutative group with identity 1.
- The distributive law holds for all elements in F, that is, $a \times (b + c) = ab + ac \forall a, b, c \in F$

2.3.3. Order

The order of a group or a field indicates the amount of numbers that exist in that group. The group of integers mod 5, for example, has an order of 5 and contains the numbers [0,1,2,3,4]

2.3.4. Finite Fields

A field with a finite number of elements is called a finite field. A finite field always has an order of a prime or a prime power. Finite fields are widely used in cryptography and number theory.

2.4. Security Assumption

Security of cryptographic protocols is guaranteed by proofing mathematical equations. A common building block for these security proofs is the use of *security assumptions*. These assumptions are about problems we assume to be hard.

In this section, we list and define the security assumptions used by the signature schemes implemented in this work.

2.4.1. Discrete Logarithm (DL)
The discrete logarithm problem in a group $\mathbb{G}$ of order q, is to produce, given g, g^a for a random generator $g \in \mathbb{G}$ and random $a \in \mathbb{Z}_q$, the value a.

2.4.2. Computational Diffie-Hellman (CDH)
The computational Diffie-Hellman problem states that given elements $g, g^a, g^b \in \mathbb{G}$ and $a, b \in_R Z_q^*$ it is intractable to compute g^{ab}. The CDH problem implies the hardness of the DL problem. If we could solve the DH problem, we could compute from g^a the value a, and as such we could compute $(g^b)^a = g^{ab}$, which would break the CDH assumption.

2.4.3. Computational Bilinear Diffie-Hellman (CBDH)
The computational bilinear Diffie-Hellman problem in groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ is to compute g_1^{ab} given points $g_1, g_1^a \in \mathbb{G}_1$, $g_2, g_2^b \in \mathbb{G}_2$, and a bilinear mapping $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, $a, b \in \mathbb{Z}_q^*$

2.4.4. Decisional Bilinear Diffie-Hellman (DBDH)
The decisional bilinear Diffie-Hellman problem is: given $g_1 \in \mathbb{G}_1$ and $g_2, g_2^a, g_2^b \in \mathbb{G}_2$ with $a, b \in_R \mathbb{Z}_q^*, \omega \in \mathbb{G}_T$, decide whether $\omega = e(g_1, g_2)^{ab}$

2.4.5. co-Computational Diffie-Hellman
The co-computational Diffie-Hellman problem in groups $(\mathbb{G}_1, \mathbb{G}_2)$ is to compute $g^x \in \mathbb{G}_1$ given $g \in \mathbb{G}_1 \setminus \{1\}$ and $h, h^x \in \mathbb{G}_2 \setminus \{1\}$ and $x \in_R \mathbb{Z}_q*$

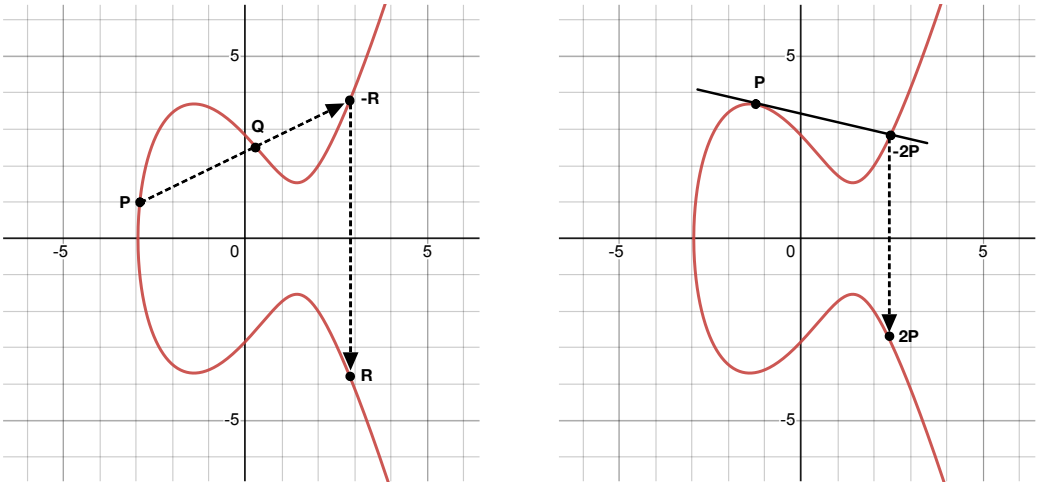
2.4.6. co-Bilinear Diffie-Hellman
The co-Bilinear Diffie Hellman problem for groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ is to compute $e(g_1, g_2)^{ab} \in \mathbb{G}_T$ given $g_1, \in \mathbb{G}_1$ and $g_2, g_2^a, g_2^b \in \mathbb{G}_2$ for unknown values $a, b \in \mathbb{Z}_q^*$

2.4.7. q-Strong Diffie-Hellman
The q-Strong Diffie-Hellman problem is, given a bilinear group ($\mathbb{G}_1$,$\mathbb{G}_2$,$\mathbb{G}_T$, $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$), with groups of equal order $p > 2^k$, and with generator g of $\mathbb{G}_1$ and h of $\mathbb{G}_2$, $q = poly(k)$, elements $g^x, g^{x^2}, \ldots, g^{x^q}, h^x$, it is hard to compute $(c, g^{\frac{1}{x+c}})$.

2.4.8. Gap Bilinear Diffie-Hellman
The gap bilinear Diffie-Hellman problem in groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ is to solve the co-BDH problem, while assuming the DBDH problem is solvable in these groups. In other words, there exists a polynomial-time solution for the DBDH problem, but there is no solution for the co-BDH problem for these groups.

2.4.9. Flexible Diffie-Hellman Inversion
The Flexible Diffie-Hellman inversion problem is to find an element W and W' such that $W' = W^{\frac{1}{z}}$ given $g_1, g_1^{\frac{z}{v}}, g_1^r, g_1^{\frac{r}{v}} \in \mathbb{G}_1, g_2, g_2^z, g_2^v$ with $z, r, v \in_R \mathbb{Z}_p$

2.5. Elliptic Curves
Elliptic-curve cryptography (ECC) is a form of public-key cryptography, based on the algebraic structures of elliptic curves over finite fields. They are widely used on the modern internet, for example for key agreement when visiting an SSL/TLS secured website, as well as for digital signatures using the standardized Elliptic-Curve Digital Signature Algorithm (ECDSA)[28].



(a) Point addition on elliptic curves

(b) Point doubling on elliptic curves

Figure 2.1: Point operations are visualized on an elliptic curve with equation $y^2 = x^3 - 6x + 8$

An important reason why ECC is used is that smaller keys can be used than in factoring-based cryptosystems while providing a similar level of security. A 256-bit ECC encryption key provides the same level of security as a 3072-bit RSA encryption key [7]. The size of these keys is an important factor for mobile devices which have less powerful processors. Performing operations using these small keys is more efficient than calculations on large prime numbers.

Elliptic curve cryptography is based on the hardness of finding the discrete logarithm (subsection 2.4.1) of an elliptic curve point with respect to a publicly known base point.

An elliptic curve is defined over a finite field, with points on the curve satisfying the following formula:

$$y^2 = x^3 + ax + b \tag{2.5}$$

2.5.1. Operations on EC

A binary operation is defined over points on the curve, called the group operation. This operation is often written as additive, and as such is called point addition. The operation takes two points on the curve, P and Q, and returns a point $R = P + Q$. To compute the coordinates of point R, a line is drawn from point P to point Q. The inverse of point R lies at the point where the line crosses the curve again. The coordinates of R lie at the vertical opposite side of the curve. This process is visualized in Figure 2.1a.

To double a point, a line is drawn tangent to the point. The double of the point lies at the vertical opposite side of where the line crosses the curve again. We illustrate this in Figure 2.1b.

When the line through a point does not intersect with the curve again, we say that applying the operation results in the point at infinity. This point is defined as the identity element of the curve.

2.6. Pairing

A pairing function (also known as a bilinear map) is defined over three elliptic curve groups; two input groups (subgroups) $\mathbb{G}_1$, $\mathbb{G}_2$ and a target group $\mathbb{G}_T$. This function maps inputs from

$\mathbb{G}_1$ and $\mathbb{G}_2$ to the target group:

$$e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T \tag{2.6}$$

We call a bilinear group a tuple of the groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$, together with the pairing function e, for which following properties hold:

Bilinearity $e(g_1^a, g_2^b) = e(g_1^b, g_2^a) = e(g_1, g_2)^{ab}$

Non-Degeneracy $e(g, h)$ generates $\mathbb{G}_T$ for any generators $g \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$

Computability $e(g, h)$ is efficiently computable for any $g \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$

We say that a pairing is of Type-1 when $\mathbb{G}_1 = \mathbb{G}_2$. This means that we pair points of the same group to a target group, instead of pairing points from two different input groups together.

For pairings of Type-1, we can break the DDH assumption by checking whether $e(g^a, g^b) = e(g^{ab}, g)$.

We say that the pairing is of Type-2 when $\mathbb{G}_1 \neq \mathbb{G}_2$ and there is also an efficiently computable mapping from points in $\mathbb{G}_2$ to $\mathbb{G}_1$: $\phi : \mathbb{G}_2 \to \mathbb{G}_1$ is an efficiently computable isomorphism.

Using this isomorphism, we can break the DDH assumption in $\mathbb{G}_2$. Namely, by checking whether $e(\phi(h^a), h^b) = e(\phi(h), h^c)$ for $h, h^a, h^b, h^c \in \mathbb{G}_2$.

Finally, a Type-3 pairing has distinct groups, $\mathbb{G}_1 \neq \mathbb{G}_2$, but no isomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$ exists. In this setting, the DDH problem is assumed to be hard in both $\mathbb{G}_1$ and $\mathbb{G}_2$.

Without going into details, examples of pairing functions are the Weil pairing [38], the Tate pairing [21] and the optimal Ate pairing [46], the last of which is considered the most efficient to compute.

2.6.1. Pairing friendly curves

Not every elliptic curve supports pairing functions. Curves that do support them are called *pairing friendly curves*. Examples of families of such curves are Barreto-Naehrig (BN) [9] Curves and Barreto-Lynn-Scott (BLS) [8] curves. Curves in these families construct optimal Ate pairings.

2.6.2. BLS 12 381

The chosen curve in this work is a Curve called BLS12-381. This is a Barreto-Lynn-Scott curve. The equation for this curve is $y^2 = x^3 + 4$. The field modulus is prime and has 383 bits. This makes it efficient to do 64-bit or 32-bit arithmetic on it. The order r of the subgroups is also prime and has 255 bits or fewer. This curve is widely used, allows for fast computations, and is recommended by the Internet Engineering Task Force (IETF) [40] for a curve that provides 128-bit security.

2.7. Hashing

A hash operation maps an arbitrary input to an element of a certain domain. This means that no matter how big the input is, the output of this function is always the same length. Hash functions are used for checksums and data access algorithms, but also for cryptographic protocols.

2.7.1. Cryptographic hash functions

Cryptographic hash functions differ slightly from regular hash functions, as they require some additional properties. For a hash function to be considered cryptographically secure, it must

provide the following characteristics:

Pre-image resistance From a hashed value h, it should be difficult to find a message m such that $m \leftarrow H(m)$

Second pre-image resistance For a message m, it should be difficult to find a different message $m'! = m$, such that $H(m) == H(m')$

Collision resistance It should be difficult to find any two messages that hash to the same value, $H(m_1) == H(m_2)$

While the second and the third properties are quite similar, the restriction to a provide message in the second pre-image resistance makes this a stricter task.

An example of a cryptographically secure hash function is SHA2 (Secure Hash Algorithm 2) [39]. This is a family of cryptographic hash functions with hash digests (output sizes) varying in length between 224 bits and 512 bits. It is recommended by the National Institute of Standards and Technology (NIST) as a standard hashing algorithm [3]

2.8. Pseudo-Random Functions

True randomness is something that is hard to acquire for computers. There are ways of sampling randomness from the world around us, but this is a difficult task. A solution to this problem is pseudo-random functions. These are functions that based on an input value produce a string of bits. Their output is very hard to distinguish from actual randomness.

2.9. Network Coding

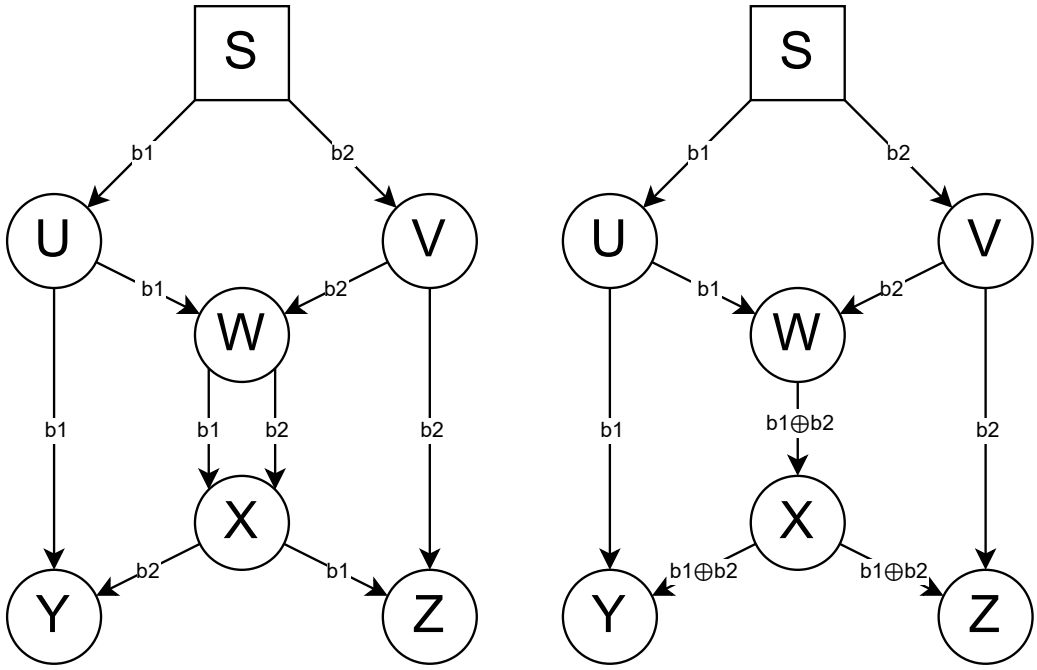
Network coding is a technique used to increase the throughput and resilience of networks. Instead of storing and forwarding separate packets, network nodes combine (code) packets together.

A simple example of the usefulness of network coding is demonstrated through the butterfly network. As we see in Figure 2.2, a source node sends a bit b_1 to intermediate node U and another bit b_2 to node V. The goal is for target nodes Y and Z to receive both of these bytes. If we want each node to only send a message once, we require that node W has two links to node X. Otherwise, after sending one of the bits, it would have to use its link again. Using network coding, on the other hand, we see that node W can combine the two bits. In this example, the bits are combined using the XOR operation. node W then forwards the XOR of b_1 with b_2 to node X, which then forwards it to target node Y. node Y ends with receiving b_1 from node U and $b1 \oplus b2$ from node X. By XORing the bit received from node X, the target node Y can retrieve bit b_2 again. We see that this change to the network has increased its efficiency.

When network coding is applied to send a file from a source node to a target node, the file is first spilt into parts called packets. The number of packets depends on the maximum transfer unit of the network medium.

Each packet is further augmented with a unit vector, based on which part of the file this packet is. For example, packet three of a file consisting of 5 packets gets augmented with the unit vector [0,0,1,0,0].

When an intermediate node has received multiple packets, it linearly combines them using random coefficients, after which it sends the combined packet to the next node. When the target node has received enough linearly independent packets, it can decode them to retrieve



(a) A butterfly network with two links between node W and X

(b) A butterfly network which codes packets together

Figure 2.2: Two butterfly networks in which a source node sends bit b_1 to intermediate node U and bit b_2 to intermediate node V

the original file.

To illustrate this concept, in Figure 2.3 we see how a file is prepared for transmission. It is first split into six packets, which are then appended with a unit vector corresponding to their place in the file. Then in Figure 2.4 we see how these packets are transmitted through the network. Packets are sent to intermediate nodes U, V and W, which combine their received packets before transmitting them to the next nodes. Finally, node Z receives a linear combination of all packets which make up the file.

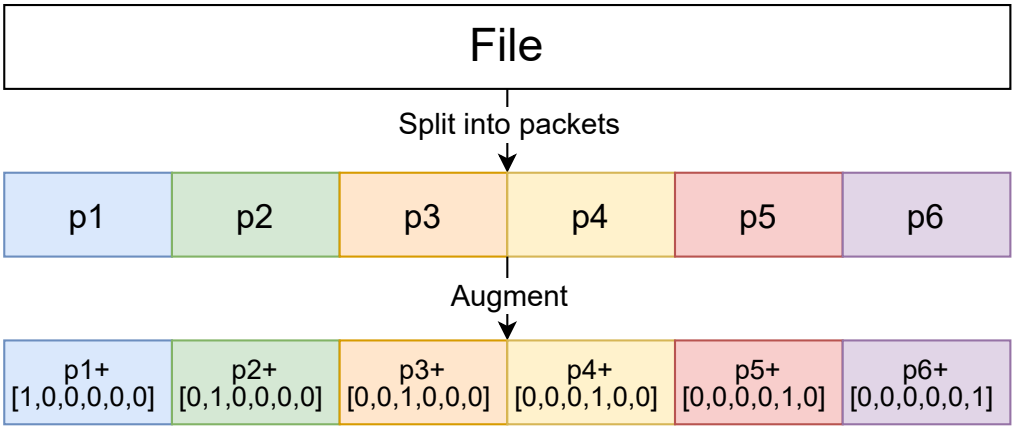


Figure 2.3: Preparation of a file for transmission in a network coding scenario. The file is split into six packets, which are then augmented with a unit vector corresponding to their placement in the file.

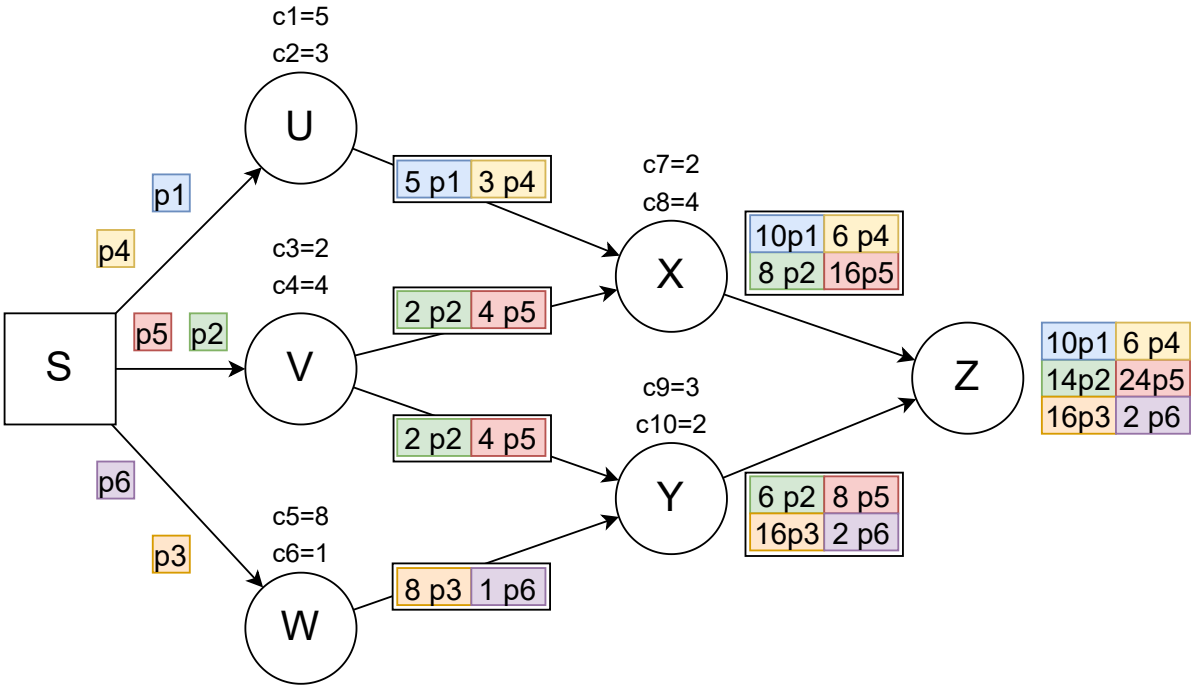


Figure 2.4: Transmission of packets through a network using network coding. Different packets are sent to three nodes, who combine the packets as they receive them by using randomly chosen codes. Combined packets are then forwarded to the next layer of intermediate nodes, which repeat this process. Target node Z finally receives a linear combination of all packets in the file.

3

Related Works

3.1. Digital Signatures

With the invention of public-key cryptography by Diffie and Hellman in 1976 [17], the first works on what is now known as a digital signature were introduced. They describe how to create a message-dependent signature, which could replace written contracts. They described a technique to produce a signature that is recognizable by anyone, but producible only by the legitimate signer. They named this technique one-way authentication.

In 1988, Goldwasser et al. [25] were the first to discuss what the security requirements of a digital signature should be. They discuss what a digital signature scheme is, what attacks on the schemes can happen and what it means to break a signature scheme. Attacks are described based on the knowledge an adversary has, namely only knowing the public key of a signer, or also having knowledge of previously produced signatures (of which the underlying message is either known or unknown). Breaking a signature scheme is defined by the following scenarios, in decreasing order of severity. A setting in which an attacker can recover a signing key, one where forgeries on any message can be produced, one where forgeries on specific messages can be produced, or lastly where one specific forgery can be made.

3.2. Homomorphic signature schemes

The concept of homomorphic signatures (HS) was first discussed by Desmet in 1993 [16]. He envisioned a mechanism to authenticate text as it was being entered into a computer, by authenticating each keystroke. As one types and corrects the text entered to a computer, signatures for each keystroke are produced. The usefulness of the 'operator-oriented signatures' is illustrated by the delete character. By signing the delete operator together with the cursor location, a corrected string can be verified. Desmet only theoretically described this concept of operator-oriented signatures and questioned the possibility of their construction.

Later, in 2002, Johnson et al. [29] formalized the notion of homomorphic signatures. In their work, they apply the already studied principles of 'privacy homomorphisms', now better known as homomorphic encryption, to signature schemes. In their work, Johnson et al. present the first formal definition of a homomorphic signature, and describe an application in the form of redactable signatures. Signatures are produced on an entire document, and to create a signature for a redacted version of the document, the homomorphic property is used.

3.3. Linearly homomorphic signature schemes

Katz and Waters [31] proposed a homomorphic signature scheme (HSS) to be used for network coding. This was one of the first works to suggest using homomorphic signatures to combat pollution attacks. Earlier works only provided ways for honest nodes to verify individual packets but did not support verifying coded packets, without re-signing them. There were two other homomorphic signature schemes proposed for network coding, but one of them [15] had the drawback of having to regenerate keys after each file that was sent, while the other scheme [48] had public keys with a size that depends on the size of the file. The solution proposed by Katz en Waters works with constant size signatures and public keys. Their construction works over symmetric pairings (Type I).

Agrawal et al. [1] presented the first homomorphic signature scheme for multi-source network coding. New definitions are presented to account for the attack model of multi-source networks. The construction is based on what is called a *vector hash*. This is a construction that captures the properties of homomorphic hashes.

In 2010, Gennaro et al. [23] presented the first linearly homomorphic signature scheme based on the RSA assumption in the random oracle model. The construction works over integers instead of over fields. This allows the use of small coefficients, which improves the computational overhead at intermediate nodes.

Attrapadung and Libert presented the first homomorphic signature scheme proven secure in the standard model [5]. Existing network coding signatures schemes proven in the standard model had to sign all the base vectors of a subspace at the same time, so the signer has to know the contents of the entire file before being able to sign it. This prevents the possibility of data streaming. This solution does allow this on the other hand. To facilitate their construction, dual encryption is used, as well as groups of composite order. Signatures are randomized. Type 1 pairing.

In 2012, Freeman [20] presented a generic framework which converts regular signature schemes to linearly homomorphic signature schemes. Their construction works as long as the signature schemes have certain properties. Namely, it works on ”hash-and-sign” signatures which raise a generator g of a cyclic group $\mathbb{G}$ to a computed value depending on a randomness r and a message: $f(m,r)$. The construction allows for a stronger adversary, who can adaptively query messages, one at a time.

Attrapadung et al. [6] presented a completely context-hiding LHS scheme. The security of the scheme is proven in the standard model. The application for this scheme was quoting substrings of a signed message. The context hiding property is created using Groth-Sahai proofs. Components of signatures that could previously not be randomized are replaced by perfectly hiding commitments. The construction works over type 1 pairings.

Libert et al.[35] present a linearly homomorphic, structure-preserving signature scheme and discuss applications in verifiable computation and trapdoor commitments. Their construction works over symmetric pairings

3.3.1. Asymmetric-pairing based

In this work, we consider signature schemes that are based on the pairings of asymmetric bilinear groups. The signature schemes listed in this section will be implemented and compared. The workings of these schemes will be discussed later on in this thesis.

The first signature scheme in this section is the one by Boneh et al. from 2009 [11]. This is the first scheme to use pairings over asymmetric bilinear groups for homomorphic signatures, and

it has set the standard for the schemes to come. In their paper, two homomorphic signature schemes are proposed: NCS_0, a generic solution for signing messages, and NCS_1 a solution optimized for network coding. Both schemes are proven secure in the random oracle model. In this paper, scheme NCS_1 will be considered, as other works are also optimized for network coding.

In 2012, Catalano et al. [13] introduce two signature schemes that are provably secure in the standard model. The schemes are proven secure using standard assumptions, namely the q-Strong Diffie Hellman assumption and the Strong RSA assumption. In their work, Catalano et al. present two homomorphic NCS schemes, which are more efficient than previous solutions proven in the standard model. The construction based on type 2 pairings is adopted in this work.

In 2017, Lin et al. [36] introduced a scheme with designated entities based on the co-BDH assumption proven secure in the random oracle model. It is the first HSS to implement a designated entity feature. The construction of this signature scheme is based on that of Boneh et al. [11].

In 2017, Schabhüser et al. [41] introduced a signature scheme that is secure against an adaptive adversary under the computational Diffie-Hellman assumption. Furthermore, this is the first context hiding construction.

In 2018, Zhang et al. [47] created an identity-based homomorphic signature scheme. Public keys are constructed as hashed values of the identity and therefore have a constant size. The scheme is further based on the construction of Boneh et al.[11].

In 2018, Schabhüser et al. [42] presented the first work on their context hiding, multi-key homomorphic signature scheme. They define the notion of context hiding in the multi-key setting and create a perfectly context hiding multi-key signature scheme. Later, in 2019, Schabhüser et al. [43] slightly changed the construction of their CH LHSS, making the signing operation more efficient.

In 2018, Li et al. [33] introduced a network coding signature scheme for multiple sources for IoT systems. Its security is proven in de standard model.

Aranha and Pagnin [4] presented a simple multi-key linearly homomorphic signature scheme in 2019, with the intention to ease the introduction to the field of homomorphic signatures. It generalizes the signature scheme by Boneh et al. [11] to the multi-key setting. The scheme is proven secure under standard assumptions in the random oracle model.

In 2020, Li et al. [34] introduced an identity-based linearly homomorphic signature scheme for network coding in IoT. Their construction allows for signing and verifying operations that are not dependent on the size of the data packets. The scheme is proven secure under adaptive identity and adaptive message space attacks in the random oracle model.

Finally, in 2021, Lin et al. [37] presented a new signature scheme with a designated combiner. In contrast to the scheme in [36], this scheme has signatures that are publicly verifiable. It is proven secure in the random oracle model.

The signature schemes in this category will be further discussed in chapter 5.

3.4. Polynomial

In 2011, Boneh and Freeman [10] constructed the first HSS which supports the evaluation of multivariate polynomial functions. To facilitate this feature, ideal lattices are used in an

analogue way to Gentry's fully homomorphic encryption[24].

Hiromasa et al. [26] present an HSS for polynomial functions, whose signatures are shorter than those in [10]. They use different algorithms during signing, which is more efficient in the sense that it produces smaller signatures. The length of their signature is reduced from $O(n^{4.5})$ to $O(n^3)$.

Catalano et al. [14] present a homomorphic signature scheme for polynomial functions, which is proven secure in the standard model instead of the random oracle model. It is proven secure against an adaptive attacker, who can query messages one at a time instead of having to query all messages at once.

Fiore et al. [19] present a formal definition of multi-key homomorphic authenticators and propose a solution based on standard lattices, which supports the evaluation of circuits of polynomial depth.

3.5. Fully homomorphic

Gennaro and Wichs [22] present a fully homomorphic message authentication scheme. Their construction relies on fully homomorphic encryption[24].

3.6. Surveys

In 2016, Traverso et al. [45] presented an extensive survey on homomorphic signatures. In that survey, the authors discuss various types of homomorphic signature schemes, one of which is pairing-based linearly homomorphic signature schemes. The schemes are compared based on their hardness assumptions, security model and the number of pairings amongst other things. However, no actual results of implementations are discussed.

In 2018, Emmanuel et al. [18] also released a survey on homomorphic signature schemes. In that work, the authors also perform a theoretical performance analysis, based on the evaluation correctness, succinctness of signatures and context hiding of signature schemes.

3.7. Homomorphic signatures in general

Catalano et al. [12] analyse a stronger notion of security for homomorphic signatures. They note that no existing homomorphic signature schemes satisfy this notion, but propose generic compilers that turn an HSS secure under weak definitions into one secure in the newly proposed stronger definition.

4

Homomorphic Signature Schemes

In this section, we explain in detail what a homomorphic signature scheme is, what it should do, and what variants exist.

4.1. Homomorphic Signatures

We start by explaining what a homomorphic signature scheme is. The intuition of homomorphic signatures is that after signing a message, operations can be run on the message as well as on the signature, after which the resulting pair verifies successfully. This is a useful property for computing on authenticated data. For example, we have a message, $m = 2$, which we sign to obtain $\sigma = Sign(sk, m)$. We can then double the message by running the function $f(x) = 2x$ on it. We also run this function on the signature: $\sigma_{double} = Combine(f, \sigma))$. Finally, verification of this signature results in $accept \leftarrow Verify(vk, \sigma_{double}, m = 4)$. Furthermore, we can also combine separate signatures. If we have $\sigma_5 \leftarrow Sign(sk, m = 5)$ and $\sigma_8 \leftarrow Sign(sk, m = 8)$, we can add these signatures together with the function $f(x, y) = x + y$. When we then combine the signatures, we get $\sigma_{13} \leftarrow Combine(f, \{\sigma_5, \sigma_8\})$. Instead of just signing numbers without a context, signatures are produced for a specific file id. This provides a context to the signatures, and only signatures signed under the same file id can be successfully combined.

4.1.1. Formal Definition

More formally, we present the definition of a homomorphic signature scheme:

Definition 4.1.1. We define a homomorphic signature scheme to consist of 5 probabilistic, polynomial time algorithms: HSS = (Setup, KeyGen, Sign, Combine, Verify):

$pp \leftarrow$ **Setup(**1^λ**)** On input the security parameter λ, the setup algorithm outputs common parameters of the signature scheme.

$(sk, vk) \leftarrow$ **KeyGen(pp)** On input the public parameters, the algorithm outputs a key-pair consisting of a signing key *sk* and a verification key *vk*.

$\sigma \leftarrow$ **Sign(**sk, m, fid**)** On input a signing key sk, a message m and a file id fid, the algorithm outputs a signature σ on the message.

$\sigma \leftarrow$ **Combine(**$f, \boldsymbol{\sigma}$ **)** On input a function $f : M^n \rightarrow M$, and a list of signatures $\sigma_1, \ldots \sigma_n \in \boldsymbol{\sigma}$, the algorithm outputs a homomorphically combined signature σ.

$accept \leftarrow$ **Verify(**vk, σ, m, fid**)** On input a verification key vk, a signature σ, a message m and a file id fid, the algorithm outputs a bit *accept*, indicating whether the verification was successful, 1, or unsuccessful, 0.

4.1.2. Correctness

Further, a homomorphic signature scheme should offer authentication correctness, evaluation correctness, as well as succinctness.

Authentication correctness means that a signature that is a result of the signing algorithm, with as input any message m from the message space $\mathcal{M}$, and any signing key sk produced by the key generation algorithm, should verify correctly with a very large probability. More formally:

Definition 4.1.2 (Authentication Correctness)**.** We define a homomorphic signature scheme to be correct with respect to authentication if:
Verify(vk, σ, m) = accept, for any $pp \leftarrow Setup(1^\lambda), (sk, vk) \leftarrow KeyGen(pp), m \in \mathcal{M}$, $\sigma \leftarrow Sign(sk, m)$

Evaluation correctness states that signatures that satisfy authentication correctness, when provided to the combine algorithm should produce a signature that verifies to the correspondingly combined message with a very large probability. More formally:

Definition 4.1.3 (Evaluation Correctness)**.** A homomorphic signature scheme is defined as correct with respect to evaluation if:
Verify(vk, σ', m') = accept, for any $pp \leftarrow Setup(1^\lambda), (sk, vk) \leftarrow KeyGen(pp)$, $m_1, ..., m_n \in \mathcal{M}^n, \sigma_1 \leftarrow Sign(sk, m_1), ..., \sigma_n \leftarrow Sign(sk, m_n)$, $accept \leftarrow Verify(\sigma_i, vk) \forall i \in \{1, ..., n\}$, $f : \mathcal{M}^n \rightarrow \mathcal{M}, m' \leftarrow f(m_1, ..., m_n), \sigma' \leftarrow Combine(f, \sigma_1, ..., \sigma_n)$.

Succinctness means that the length of the homomorphically combined signature should be shorter than the length of m individual signatures.

4.1.3. Forgeries

The unforgeability requirement of digital signatures has to change for homomorphic signatures to account for the *intended* possibility of creating new signatures by combinations. For regular digital signatures, this would strictly break the unforgeability definition. To adapt to this characteristic, the following types of forgeries are described, against which an HSS should be protected.

Type I A signature is produced on a dataset for which the file identifier has never before been queried.

Type II A homomorphic signature authenticates to a value that is not the correct output of a function: $m* != f * (m_1, ..., m_n) \wedge 1 \leftarrow Verify(\sigma, m*, fid)$.

4.2. Construction of a pairing-based LHSS

To give an intuition of how a pairing-based linearly homomorphic signature scheme is constructed, we explain how the scheme by Boneh et al. [11] is defined. This was the first scheme to employ pairing of asymmetric bilinear groups and the schemes that came after it are similar in construction. This makes it a good choice as an example.

Setup To setup the scheme, based on the security parameter a bilinear group is chosen. The amount of items that can be combined together is fixed to a number. Corresponding to the length of a packet n, generators $g_1, ..., g_n \in \mathbb{G}_1$ are chosen, as well as a generator g'

Algorithm 1 NCS2 by Boneh et al.

function Setup(λ)
  $\mathcal{G} \leftarrow (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e, \phi)$ ▯ Select a bilinear group based on the security parameter λ
  $(g_1, ..., g_n) \leftarrow_r \mathbb{G}_1^n$ ▯ Select generators of $\mathbb{G}_1$ based on the size of a packet
  h $\leftarrow_r \mathbb{G}_2$
  $\mathcal{H} = (0,1)^* \times (0,1)^* \rightarrow \mathbb{G}_1$ ▯ Select a hash function $\mathcal{H}$
  $pp \leftarrow (\mathcal{G}, g_1, ..., g_n, h, \mathcal{H})$
  return pp
end function
function KeyGen(pp)
  $sk \leftarrow_r Z_q^*$
  $vk = h^{sk}$
  return (sk, vk)
end function
function Sign($pp, \vec{m}, sk, fid$)
  parse $\vec{m} = (u_1, ..., u_m, v_1, ..., v_n)$ ▯ The message consists of data $\vec{v}$ and unit vector $\vec{u}$
  $\sigma \leftarrow 1$
  for $1 \le i \le m$ **do**
    $\sigma \leftarrow \sigma \cdot \mathbb{H}(fid, i)^{u_i}$ ▯ Bind to the file id
  end for
  for $1 \le j \le n$ **do**
    $\sigma \leftarrow \sigma \cdot g_j^{v_j}$ ▯ Involve each part of the packet data in the signature
  end for
  $\sigma \leftarrow \sigma^{sk}$ ▯ Bind the signature to the secret key
  return σ
end function
function Combine($pp, \vec{\sigma}, \vec{f}$)
Require: $|\vec{\sigma}| = |\vec{f}|$
  $\sigma' \leftarrow 1$
  for $1 \le i \le |\vec{\sigma}|$ **do**
    $\sigma' \leftarrow \sigma' \cdot \sigma_i^{f_i}$
  end for
  return σ'
end function
function Verify($pp, \vec{m}, vk, \sigma, fid$)
  $p_1 \leftarrow e(\sigma, h)$
  parse $\vec{m} = (u_1, ..., u_m, v_1, ..., v_n)$
  s $\leftarrow 1$ ▯ Reconstruct the signature
  for $1 \le i \le m$ **do**
    s $\leftarrow s \cdot \mathbb{H}(fid, i)^{u_i}$
  end for
  for $1 \le j \le n$ **do**
    s $\leftarrow s \cdot g_j^{v_j}$
  end for
  $p_2 \leftarrow e(s, vk)$
  return p_1 equals p_2
end function

of $\mathbb{G}_2$ is chosen at random.

KeyGen To generate a key pair, a random field scalar is chosen as signing key: $sk \in_R {Z_q}^*$. Next, the verification key is calculated as $vk = g'^{sk}$. The verification key is an element of $\mathbb{G}_2$. The generated key-pair is then provided to the user.

Sign To sign a message $\overrightarrow{m} = (v_1, \ldots v_n, u_1, \ldots, u_m)$, a signature is calculated as follows:

$$\sigma = \left(\prod_{i=1}^{m} (\mathbb{H}(fid, i)^{u_i}) \prod_{j=1}^{n} (g_j^{v_j}) \right)^{sk}$$

Combine To combine signature based on a given function, we calculate $\sigma' = \prod_{i=1}^{m} \sigma_i^{f_i}$

Verify To verify a signature, we first compute a pairing over the supplied signature:

$$p_1 = pairing(\sigma, g') \tag{4.1}$$

Then, using the message $\overrightarrow{m} = (v_1, .., v_n, u_1, \ldots, u_m)$ which we want to verify, we compute $s = \prod_{i=1}^{m} (\mathbb{H}(fid, i)^{u_i}) \prod_{j=1}^{n} (g_j^{v_j})$, and compute the second pairing:

$$p_2 = pairing(s, vk) \tag{4.2}$$

Finally, we check whether $p_1 \stackrel{?}{=} p_2$. If so we return 1, and if not we return 0.

The pseudo code of this signature scheme is presented in Algorithm 1.

4.2.1. Verifying

To see how the verification works, we go through an example. We sign a message $\overrightarrow{m} = (4, 1, 0)$:

$$\sigma = (h_1^1 \cdot h_2^0 \cdot g_1^4)^{sk} \tag{4.3}$$

To verify this signature, we first compute the pairing $p_1 = e(\sigma, g')$, after which we compute $p_2 = e(h_1^1 \cdot h_2^0 \cdot g_1^4, vk) = e(h_1^1 \cdot h_2^0 \cdot g_1^4, g'^{sk})$.

Finally, we check whether the two pairings are equal:

$$\begin{aligned} p_1 &= e(\sigma, g') \\ &= e((h_1^1 \cdot h_2^0 \cdot g_1^4)^{sk}, g') \\ &= e(h_1^1 \cdot h_2^0 \cdot g_1^4, g')^{sk} \\ &= e(h_1^1 \cdot h_2^0 \cdot g_1^4, g'^{sk}) \\ &= e(h_1^1 \cdot h_2^0 \cdot g_1^4, vk) \\ &= p_2 \end{aligned} \tag{4.4}$$

We see that using the pairing operation, we can check whether a signature is valid or not.

4.2.2. Combining

To explain this operation, we work out an example where the packet length is 1, and the file length is 2. Our packets to be signed are $p_1 = [5, 1, 0]$ and $p_2 = [11, 0, 1]$. During signing, our signatures will become:

$$\begin{aligned} \sigma_1 &= (h_1^1 \cdot h_2^0 \cdot g_1^5)^{sk} \\ \sigma_2 &= (h_1^0 \cdot h_2^1 \cdot g_1^{11})^{sk} \end{aligned} \tag{4.5}$$

We will combine these signature using the function $f = 4a + 2b$. When we use the combine operation, we exponentiate the signatures based on the given function and multiply the signatures together:

$$\begin{aligned}
\sigma' &= \sigma_1^{f_1} \cdot \sigma_2^{f_2} \\
&= ((h_1^1 \cdot h_2^0 \cdot g_1^5)^{sk})^4 \cdot ((h_1^0 \cdot h_2^1 \cdot g_1^{11})^{sk})^2 \\
&= ((h_1^1 \cdot h_2^0 \cdot g_1^5)^4 \cdot (h_1^0 \cdot h_2^1 \cdot g_1^{11})^2)^{sk} \\
&= ((h_1^4 \cdot h_1^0) \cdot (h_2^0 \cdot h_2^2) \cdot (g_1^{20} \cdot g_1^{22}))^{sk} \\
&= (h_1^{4+0} \cdot h_2^{0+2} \cdot g_1^{20+22})^{sk} \\
&= (h_1^4 \cdot h_2^2 \cdot g_1^{42})^{sk} \\
&= Sign(sk, m = [42, 4, 2])
\end{aligned} \tag{4.6}$$

As we can see, due to the homomorphic property, multiplying signatures together combines them as if an addition to the message was performed, and raising a signature to a number corresponds to multiplication.

Due to the homomorphic property, given a signature σ on a message m, we can get a valid signature on any linear combination of m (e.g. $5m, 23m$ or even m^2), simply by using the combine operation. This would break the unforgeability property of digital signatures. To deal with this situation, we have to change the definition of unforgeability of a homomorphic signature slightly as we discussed before. It changes such that only a linear combination of m should be possible to create by anyone who is not the signer, however, anything but that should still be considered a forgery.

4.3. Variants of Homomorphic Signature Schemes

There are homomorphic signature schemes that not only offer homomorphic combining of signatures but also facilitate additional characteristics. We call these variants of the basic homomorphic signature scheme, and we discuss them in the following paragraphs.

4.3.1. Network Coding (NC)

A homomorphic signature scheme designed for network coding is adapted such that a message to be signed is always a vector. We call these messages packets, a term from computer networking to indicate a single unit of data to be sent over the network. Network coding packets (the data to be signed) consist of a vector of actual data, followed by a unit vector $\vec{u}$ which indicates which part of the file this packet is. This unit vector is further used during the combination of packets and their decoding. They allow the target node to perform a matrix inversion based on the values that remain in the augmented part of a vector.

For network coding signatures (NCS), during the setup operation, the packet length and the file length are configured. This is because the amount of chosen generators which are part of the public parameters is dependent on these numbers. During the combination phase, the entire vector gets multiplied by a scalar coefficient (code).

4.3.2. Identity-Based (ID)

Earlier in chapter 2 we discussed that in public-key cryptography, every entity has a public and a private key. Trusting that a public key belongs to someone is a non-trivial task, and requires a lot of identity and certificate management. To combat this issue, the idea of identity-based cryptography was proposed [44]. Instead of generating a key pair from complete randomness, we appoint a key generation centre (KGC) that generates a secret key based on a supplied

identity. This allows us to use our own identity as a public key. An identity can for example be an email address.

For an identity-based, homomorphic signature scheme, the setup operation is changed to generate a master secret key for the key generation centre, as well as a master public key which is part of the public parameters. The key generation algorithm is replaced by the **Key Extract** algorithm, which takes as input the identity of an entity. Anyone who wants to register for a key provides the KGC with their identity, and in return receives a secret signing key.

4.3.3. Multi-Key (MK)

When we want to be able to combine signatures signed by multiple different parties, we need a special variant of homomorphic signatures, called multi-key homomorphic signatures. This property makes it possible to combine and verify signatures signed under different signing keys. They allow for multi-party computations, a feature that is very useful as it allows collaborative work over authenticated data.

In a multi-key setting, each signer generates a pair of keys. Signatures for multi-key LHSSs are bigger than those of 'single-key' schemes. This is because the signatures have to store additional information on which party contributed which part of the homomorphic signature.

4.3.4. Context Hiding (CH)

Some schemes offer additional functionality which prevents anyone from learning which messages were signed as an input of a homomorphic signature. This privacy-preserving feature is called context hiding. When context hiding is applied to multi-key HSS, a distinction can be made between who the inputs should be hidden from. If the inputs are only hidden to an outside party, who was not involved in the signing of any of the signatures that are homomorphically combined, it is called *externally context hiding*. If additionally, anyone involved in the signing process is also not able to learn what other entities provided as inputs to signatures, we call it *internally context hiding*.

4.3.5. Designated Entities (DE)

When the combining or verifying operation needs to be performed by a specified entity, we can use an HSS with designated entities. This idea was introduced by Jakobsson et al.[27]. These kinds of signature schemes allow the signer to *designate* someone to combine those signatures, whereas the combiner can further designate someone else to verify the homomorphically combined signature. There also exists a variant in which there is only a designated combiner, while verification of signatures is performed publicly.

DE-HSS are useful when it is of importance that a specific person performs the combination or verification of the signatures. While the proposed DE-HSSs are designed for network coding, they are not particularly useful in such a setting. Only in the specific setting where there is only 1 intermediate node, e.g. $S \rightarrow I \rightarrow T$, this scheme would work. This is because only a designated node can combine signatures. This defeats the purpose of network coding, where signed packets are combined at every intermediate node. A setting in which this construction *is* useful, however, is verifiable cloud computing. By designating the expected identity to combine signatures, you can be convinced that the work was actually done by the expected party.

5

Implementation details

In this section, we discuss the signature schemes that we implement. We explain how the implementation was done, and how we compare the various schemes.

What does it mean for a comparison to be fair? Regarding real-world implementation, this means that we have programmed everything in such a way that the same design choices were made. All implementations were done using the same programming language, using the same security level and the same building blocks. This makes sure that one scheme does not have an advantage over another scheme because of programming design choices.

5.1. Implementation

The implementation of the signature schemes was done using the Rust programming language. This language is designed for performance as well as safety, which makes it a great choice for implementing cryptographic protocols.

The Rust language offers a large number of additional libraries which offer additional functionality, called crates. One of these is the BLS12-381 crate. This crate offers basic functionality for operations on the curve, such as point additions in groups, exponentiating points by given scalar values, but also hashing arbitrary numbers to the curve, as well as pairing group elements together.

The implementation of crate BLS12-381 was done in such a way that operations take a constant amount of time. This means that it does not matter if you add the numbers 2 and 2 together, or 62810950891 and 35284008516. As long as the operations are the same, the amount of time is not affected by the size of the number (so any addition always takes the same amount, as well as any multiplication). This is an important feature, as it helps to mitigate *side channel attacks*. In such an attack, an adversary wants to learn as much as possible about the operations that take place in a protocol, in the hopes of extracting for example a secret key. If operations do not take the same amount of time, but instead vary based on the size of a number, one could learn for example from timing how long it takes to sign a message what the secret key is (this is a very simplified example).

Schemes that use hash functions are implemented with the Secure Hashing Function 2 (SHA2) algorithm, specifically SHA256. This hash algorithm was required for hashing items to point on the curve with the BLS12-381 crate, and to keep our comparison as fair as possible we decided to use this hash function for all other hashing-related operations as well.

Choosing secret keys and random generators of groups requires what is called a random number generator (RNG). Since the security of cryptographic protocols relies on these numbers, it is very important that a good RNG is used with output that is indistinguishable from true randomness. Otherwise, keys can be analyzed which can be used in an attack. For our work, we use a cryptographically secure pseudo-random number generator (CSPRNG) called ChaCha20. This CSPRNG is faster for software implementations than the previous standard and is well studied, making it a good choice for our implementations.

5.1.1. Network Coding optimization

Schemes that were designed for network coding have been optimized with regard to the signing and verifying operation. Network coding packets are augmented to either prepend or append the packet data with a unit vector indicating which part of the file this packet is. As part of the signing and verifying operation, these schemes loop over the numbers of the unit vector, as we have seen in Algorithm 1 in the signing and verifying procedures. During this part of the algorithm, the index is hashed to a $\mathbb{G}_1$ element, after which they raise it to the corresponding unit vector, which we illustrate with the following example:

$$\prod_{i=1}^{m} \mathbb{H}(i)^{v_i} \text{ for } v = [0, 1, 0, 0, 0] \tag{5.1}$$

As can be seen, this equation can be reduced to $\mathbb{H}(2)^1$, as all other hashed values will be raised to 0 and result in the identity element. To optimize this operation, we check whether the current part of the unit vector is non-zero before doing any (unnecessary) hashing operations. While this might introduce a side channel to the code, the only information to be inferred is which part of a file is signed. We argue that this is a reasonable consideration, as the operation speed changes from $O(m)$ to $O(1)$. Furthermore, the signature for a network coding packet is intended to be transmitted together with the packet. This would therefore also reveal the index of the transmitted packet.

5.2. Testing method

To be able to compare the performance of signature schemes, we measure the duration of the signing, verifying and combining operation. The setup and key generation functions have been omitted since they are not run as frequent as the other operations. The setup operation is only run once for the instantiation of a signature scheme, while the key generation algorithm is run only once for each participating entity. Parties that do not need to sign messages, but instead only verify signatures, do not even have to run the key generation algorithm. This makes the duration of these two algorithms of little to no importance to someone who must choose a signature scheme.

The messages to sign for each test are selected at random. Messages are chosen as scalar values of the underlying field of the bilinear group. Since operations on the field and group items are implemented in constant time, it does not matter for the duration of the operations what numbers are signed.

In this work, we adopt the naming of network coding to describe the data to be signed. We consider non-augmented packets as the data to be signed, and the number of signatures that will be combined together is described by the file length. As we want to know how signature schemes behave under different configurations, we vary the file and packet length throughout our tests. This allows us to analyse the signature schemes' behaviour under various circumstances. We have chosen to vary both the file length and the length of the packet between 1

and 128, in steps of powers of two ([1,2,4,8,16,32,64,128]).

We measure the duration of doing an operation 100 times, after which we record the average duration. This provides us with a stable measurement of the duration of the operations.

The experiments were carried out on a computer equipped with an Intel Core i7-8750H @ 12x 4.1GHz CPU with 6 cores and 32Gb of ram.

5.3. Definition of performance

Performance can be interpreted in different ways. As we discuss real-world implementations of homomorphic signature schemes, we define performance in this context based on the speed at which a signature scheme performs a certain operation, and by the size of keys and signatures. When one signature scheme signs faster than another signature scheme, we say that its performance is better. Some signature schemes feature additional properties such as multi-key support. These additional features introduce tradeoffs to be made. We can thus define these tradeoffs in terms of performance, by looking at the difference in speed of operations with signature schemes that do not offer these features.

5.4. Overview of implemented schemes

In this section, we briefly name the schemes we have implemented. Schemes are named after the author of a work (up to the first three letters of a name), followed by the last two digits of the year the work was published. Interesting implementation details will be provided as well.

5.4.1. Bon09

This is the first scheme to apply pairings of asymmetric bilinear groups to facilitate homomorphic signatures. In their work, two schemes based on pairings are suggested, named NCS_0 and NCS_1. The former is a generic scheme, with messages to be signed being a vector of numbers, while the latter is a scheme designed for network coding. In this work, we have chosen to implement scheme NCS_1, since most other schemes are designed for network coding as well. This makes a comparison between these schemes easier. We recall from earlier, the method of signing for this scheme is as follows:

Algorithm 2 Sign algorithm of Bon09

function Sign($pp, \overrightarrow{m}, sk, fid$)
  parse $\overrightarrow{m} = (u_1, ..., u_m, v_1, ..., v_n)$
  $\sigma = (\prod_{i=1}^{m} \mathcal{H}(fid, i)^{u_i} \cdot \prod_{j=1}^{n} g_j^{v_j})^{sk}$
  return σ
end function

5.4.2. Cat12

This work presents two solutions, one based on the RSA problem and one based on the discrete log problem. We implement the solution based on the DL problem. This work proves the security of their scheme in the standard model instead of in the random oracle model.

During the setup of this scheme, generators $h, h_1, ..., h_m$ of $\mathbb{G}_1$ are selected corresponding to the length of the file, in addition to the generators for the length of the packet as in Bon09.

As part of signature generation, a random number $s \in {Z_q}^*$ is chosen, and the signing operation changes to:

Algorithm 3 Sign algorithm of Cat12

function Sign($pp, \overrightarrow{m}, sk, fid$)
parse $\overrightarrow{m} = (u_1, ..., u_m, v_1, ..., v_n)$
s $\leftarrow_r {Z_q}^*$
$\sigma = (h^s \prod_{i=1}^{m} h_i^{u_i} \cdot \prod_{j=1}^{n} g_j^{v_j})^{\frac{1}{sk+fid}}$
return σ, s
end function

5.4.3. Lin17

This is the first work to present a homomorphic signature scheme with designated entities. It features a designated combiner and a designated verifier. The security of the scheme is proven in the random oracle model. The signing operation requires the signing key of signing party A, and the public key of the combining party B. This scheme uses three Hash functions to map arbitrary data, message vectors and points in $\mathbb{G}_T$ to points in $\mathbb{G}_1$.

To facilitate the designated combiner functionality, the signing operation is changed to bind the signature to the identity of the combiner:

Algorithm 4 Sign algorithm of Lin17

function Sign($pp, \overrightarrow{m}, sk_A, vk_B, fid$)
parse $\overrightarrow{m} = (u_1, ..., u_m, v_1, ..., v_n)$
$\sigma = (\prod_{i=1}^{m} \mathbb{H}_1(fid, i)^{u_i} \cdot \prod_{j=1}^{n} g_j^{v_j})^{sk_A}$
$\sigma \leftarrow \sigma \cdot \mathbb{H}_3(e(\mathbb{H}_2(\overrightarrow{m}), vk_B)^{sk_A})$
return σ
end function

The first part of the signing process is the same as seen before, but the part after that makes is so that only party B can combine these signatures.

Then, to designate a verifier during the combining algorithm, the designated combiner 'strips off' his designated combiner part by calculating $H_3(e(H_2(\vec{v}), vk_a)^{sk_b})^{-1}$, and multiplies it to the corresponding signature. We say that this 'strips off' the designated combiner part, because after multiplying with the inverse of the designated combiner binding, we are left with the same signature as the ones produced by Bon09. This can be seen in Equation 5.2.

$$
\begin{aligned}
\sigma' &= \sigma \cdot H_3(e(H_2(\vec{v}), vk_a)^{sk_b})^{-1} \\
&= \prod_{i=1}^{m} H_1(id, i)^{v_{n+i}} \prod_{j=1}^{n} g_j^{v_j})^{sk_a} \cdot H_3(e(H_2(\vec{v}), vk_b)^{sk_a}) \cdot H_3(e(H_2(\vec{v}), vk_a)^{sk_b})^{-1} \\
&= \prod_{i=1}^{m} H_1(id, i)^{v_{n+i}} \prod_{j=1}^{n} g_j^{v_j})^{sk_a} \cdot H_3(X) \cdot H_3(X)^{-1} \\
&= \prod_{i=1}^{m} H_1(id, i)^{v_{n+i}} \prod_{j=1}^{n} g_j^{v_j})^{sk_a}
\end{aligned}
\tag{5.2}
$$

After that, all signatures are combined as usual; by raising them to coefficients and multiplying

them together. Finally, the homomorphic signature is paired with the public key of the verifier C, making him the only party able to verify this signature:

Algorithm 5 Combine algorithm of Lin17

function Combine($pp, \vec{\sigma}, \vec{f}, vk_C$)
Require: $|\vec{\sigma}| = |\vec{f}|$
σ' $\leftarrow$1
for $1 \leq i \leq |\vec{\sigma}|$ **do**
$\sigma' \leftarrow \sigma' \cdot (\sigma_i \cdot [H_3(e(H_2(\vec{v}), vk_a)^{sk_b})^{-1}])^{f_i}$
end for
σ' $\leftarrow e(\sigma', vk_C)$
return σ'
end function

5.4.4. Sch17

This signature scheme is proven in the standard model and provides context hiding. Furthermore, what is interesting about this scheme is that its key pair consists of a regular signature scheme key pair and a key for a pseudo-random function. Signatures are produced as follows:

Algorithm 6 Sign algorithm of Sch17

function Sign($pp, \vec{m}, i, sk, \Delta = fid$)
z $\leftarrow PRF_K(\Delta)$
Z $\leftarrow g_2^z$
$\sigma_\Delta \leftarrow Sign_{sig}(sk_{sig}, Z|\Delta)$
$\Lambda \leftarrow (h_i \cdot \prod_{i=1}^{n} g_j^{-m[j]})^z$
return σ= $(\sigma_\Delta, Z, \Lambda)$
end function

As we see, using the PRF on the file id, the equivalent of a key pair is generated (z, Z) for a specific dataset. By signing the dataset information, a verifier can be assured of its legitimacy.

During the combination phase, only one σ_Δ is kept, as all signatures on the same dataset will produce the same Z values using the keyed PRF. The Λ is combined as we have seen in the other schemes.

5.4.5. Li18

The signature scheme by Li offers multi-key support. Signatures are generated just like in the Cat12. What has changed, however, is that signatures now also contain an array to store signatures of other identities.

During the initial signing, the Xs corresponding to identities other than the signer's are set to 1 (or the identity element). When signatures are combined, however, these X's are filled by the signatures of other identities. This way the multi-key ability is facilitated. This does however increase the size of multi-key signatures, by 1 $\mathbb{G}_1$ element for each identity. Furthermore, the verification algorithm also scales with the number of identities, as a pairing with the public key of each involved identity has to be performed.

Algorithm 7 Sign algorithm of Li18

function Sign($pp, \overrightarrow{m}, sk_{id}, id, fid$)
parse $\overrightarrow{m} = (u_1, ..., u_m, v_1, ..., v_n)$
s $\leftarrow_r {Z_q}^*$
$X_{id} = (h^s \prod_{i=1}^{m} h_i^{u_i} \cdot \prod_{j=1}^{n} g_j^{v_j})^{\frac{1}{sk_{id}+fid}}$
$\sigma \leftarrow (X_1, ..., X_{id}, ..., X_t, s)$ ▯ $X_j = 1$ for each $j \neq id$
return σ
end function

5.4.6. Zha18

The signature scheme by Zhang offers identity-based key generation. This feature changes the setup operation to generate a master secret key for the KGC: $msk \leftarrow_R Z_q$. The master public key, which is part of the public parameters, is calculated as h^{msk}. To then generate a key for an identity, the key extract algorithm is run:

Algorithm 8 Key Extract algorithm of Zha18

function KeyExtract(mpk, msk, id)
r $\leftarrow_R Z_q$
$y = r + msk \cdot \mathbb{H}_0(h^r, id)$
return $sk_{id} = (y, h^r)$
return $p_1 == p_2$
end function

We see that the signing key consists of a y value and a value h^r. The latter is a public value, which is provided to verifiers. One might think that this means there is still some certificate management involved, but this h^r value only works in combination with the right identity id, so no certificates validating this 'public key' are required.

During verification, using the provided value h^r and the signer's id, the hash value $\mathbb{H}_0(h^r, id)$ is computed, which allows the verifier to do the pairing operation we have seen in the other verifying algorithms.

Algorithm 9 Verify algorithm of Zha18

function Verify($mpk, id, h^r, fid, \overrightarrow{m}, \sigma$)
$p_1 \leftarrow e(\sigma, h)$
$p_2 \leftarrow e(\prod_{i=1}^{m} \mathbb{H}_1(fid, i)^{u_i} \cdot \prod_{j=1}^{n} g_j^{v_j}, h^r \cdot mpk^{\mathbb{H}_0(h^r, id)})$
end function

5.4.7. Sch18

The signature scheme by Schabüser features context hiding, multi-key support. It uses a key-pair of a regular signature scheme, as well as a key for a pseudo-random function as part of its key-pair. Furthermore, scalar values corresponding to the size of a message are chosen as part of the private key, which are used to compute public values in $\mathbb{G}_T$:

The signing algorithm starts off similar to Sch17, as it also computes a signature on the dataset.

We observe that this signing algorithm involves more steps than we have seen previously. The R and the S components are global and are used to preserve the homomorphic property

Algorithm 10 KeyGen algorithm of Sch18

function KeyGen(pp)
K $\leftarrow_R \mathcal{K}$
$(sk_{sig}, vk_{sig}) \leftarrow KeyGen_{sig}(\lambda)$
$x_1, \ldots, x_n, y \leftarrow_R Z_q$
for $1 \leq i \leq n$ **do**
$h_i = g_t^{x_i}$
end for
Y $\leftarrow g_2^y$
return $sk = (K, sk_{sig}, x_1, \ldots, x_n, y), vk = (pk_{sig}, h_1, .., h_n, Y)$
end function

Algorithm 11 Sign algorithm of Sch18

function Sign($pp, \overrightarrow{m}, sk, \Delta, l$)
parse $l = (id, \tau)$
z $\leftarrow PRF_K(\Delta)$
Z $\leftarrow g_2^z$
$\sigma_\Delta \leftarrow Sign_{sig}(sk_{sig}, Z|\Delta)$
r,s $\leftarrow_R Z_q$
R $\leftarrow g_1^{r-ys}$
S $\leftarrow g_2^{-s}$
A = $(g_1^{x_\tau + r} \cdot \prod_{j=1}^{n} \mathbb{H}_j^{ym[j]})^{\frac{1}{z}}$
C = $(g_1^s \cdot \prod_{j=1}^{n} \mathbb{H}_j^{m[j]})$
$\Lambda \leftarrow \{(id, \sigma_\Delta, Z, A, C)\}$
return σ= (Λ, R, S)
end function

throughout combining signatures from multiple parties. The A and the C components are randomized to provide internal context hiding. These components are not global, but instead, they are separated per involved identity.

To combine these signatures, the R and the S components are combined as we have seen before; by raising them to provided coefficients and multiplying them together. A similar operation happens to the A and C components, but they are grouped by the identity that provided the individual signatures. So, all A components of signatures provided by party P are homomorphically combined, as are the C components. We thus see that the size of the homomorphic signature scales linearly with the number of participating parties.

5.4.8. Ara19

The signature scheme by Aranha and Pagnin features multi-key support. Signatures are produced on single messages instead of on vectors of messages (packets). This is the only signature scheme which does not sign vectors.

During key generation, a random id is selected from the ID space, which is bound to an identity. Signatures are produced as follows:

Algorithm 12 Sign algorithm of Ara19

function Sign(pp, m, sk, l)
$\gamma = (\mathbb{H}(l) \cdot g_1^m)^{sk_{id}}$
$\mu = m$
return $\sigma = (id, \gamma, \mu)$
end function

We see that this is a very basic version of the scheme Bon09, without support for network coding. The signature consists of three parts, the identity, the actual signature and the message. The reason the identity and the message are kept as part of the signature is to allow multi-key combinations. During combination, the γ part is combined as in Bon09, but messages are combined on a per identity basis. This allows the reconstruction of the signature in the verification procedure. For each involved identity, a pairing operation is performed with the corresponding messages and verification key.

Algorithm 13 Verify algorithm of Ara19

function Verify($\mathcal{P}, \{vk_{id}\}, \sigma, m$)
parse $\mathcal{P} = (f, l_1, ..., l_n)$
parse $\sigma = (\gamma, \mu_1, ..., \mu_t)$
$v_1 \leftarrow m == \sum_{k=1}^{t} \mu_k$
$c = \prod_{j=1}^{t} e(g_1^{m_j} \cdot \prod_{i \in ID_j} \mathbb{H}(l_i)^{f_i}, vk_{id_j})$
$v_2 \leftarrow e(\gamma, g_2) == c$
return $v_1 \wedge v_2$
end function

We observe that the size of the signature scales with the number of involved identities, as do the number of operations of the verification algorithm.

During the combination of signatures, to facilitate multi-key functionality, the contributions of each signer are stored in an array of μs.

5.4.9. Sch19

The signature scheme by Schabüser features context hiding, multi-key support. The signing and verification of this signature scheme are slightly changed with respect to Sch18. In Sch18, the R and the A component are bound to the scalar y. In Sch19, R and A are no longer bound to y, but now instead component C is bound to it. This shuffling around allows the verification algorithm to reuse pairings that are already computed, saving one pairing operation during verification.

Algorithm 14 Sign algorithm of Sch19

function Sign($pp, \overrightarrow{m}, sk, \Delta, l$)
parse $l = (id, \tau)$
z $\leftarrow PRF_K(\Delta)$
Z $\leftarrow g_2^z$
$\sigma_\Delta \leftarrow Sign_{sig}(sk_{sig}, Z|\Delta)$
r,s $\leftarrow_R Z_q$
R $\leftarrow g_1^{r-s}$
S $\leftarrow g_2^{-s}$
A = $(g_1^{x_\tau + r} \cdot \prod_{j=1}^{n} \mathbb{H}_j^{m[j]})^{\frac{1}{z}}$
C = $(g_1^s \cdot \prod_{j=1}^{n} \mathbb{H}_j^{m[j]})^{\frac{1}{y}}$
$\Lambda \leftarrow \{(id, \sigma_\Delta, Z, A, C)\}$
return σ= (Λ, R, S)
end function

5.4.10. Li20

The signature scheme by Li offers identity-based key generation. Further, the signing operation of this scheme differs from most constructions, as operations on $\mathbb{G}_1$ are mostly replaced by operations on Z_q. The key extraction algorithm is the same as that of Zha18 [47]. The signing operation becomes:

Algorithm 15 Signing algorithm of Li20

function Sign($pp, \overrightarrow{m}, sk, \tau, fid$)
s $\leftarrow \sum_{j=1}^{n} \mathbb{H}_2(fid, j, \tau) \cdot v_j$
$\sigma \leftarrow \prod_{i=1}^{m} \mathbb{H}_1(fid, i)_i^u \cdot g^s$
$\sigma \leftarrow \sigma^{sk}$
return σ
end function

5.4.11. Lin21

The signature scheme by Lin offers designated combining, with public verification. It is designed for network coding. The designated combination part of the signature scheme is the same as that of Lin17. What changes in this scheme is that the signatures are publicly verifyable after combining.

5.5. Sizes and characteristics

The sizes of key pairs and signatures vary between the schemes. Most schemes have a scalar number as a signing key, an element of $\mathbb{G}_2$ as a public key and signatures as elements of $\mathbb{G}_1$. In Table 5.1, we show for each scheme what the respectful sizes are. Furthermore, in

Table 5.1: Overview of sizes expressed in group and field elements, and an indication of supporting Multi Key (MK), Context Hiding (CH), Identity-based key extraction (ID-B) and designating combiner (C) or verifier (V) (DE)

Scheme	Signature Size	SK Size	VK Size	MK	CH	ID-B	DE
Bon09 [11]	$1\mathbb{G}_1$	$1\ \mathbb{Z}_q^*$	$1\mathbb{G}_2$				
Cat12 [13]	$1\mathbb{G}_1, 1Z_q$	$1Z_q$	$1\ \mathbb{G}_2$				
Lin17 [36]	$1\mathbb{G}_1$	$1Z_q$	$1\mathbb{G}_2$				C&V
Sch17 [41]	$1\mathbb{G}_1, 1\mathbb{G}_2, \sigma_{sig}$	sk_{sig}, sk_{prf}	vk_{sig}		✓		
Sch18 [42]	$1\mathbb{G}_1, 1\mathbb{G}_2, m \cdot$ (id, $\sigma_{sig}, 2\mathbb{G}_1, 1\mathbb{G}_2$)	sk_{sig}, sk_{prf} $m + 1Z_q$	$vk_{sig}, 1\mathbb{G}_2$ $m\mathbb{G}_T$	✓	✓		
Zha18 [47]	$1\mathbb{G}_1$	$1Z_q$	$1\mathbb{G}_2$,id			✓	
Li18 [33]	$m\mathbb{G}_1, 1Z_q$	$1Z_q$	$1\mathbb{G}_2$				
Ara19 [4]	$1\mathbb{G}_1, mZ_q$	$1Z_q$	$1\mathbb{G}_2$	✓			
Sch19 [43]	$1\mathbb{G}_1, 1\mathbb{G}_2, m \cdot$ (id, $\sigma_{sig}, 2\mathbb{G}_1, 1\mathbb{G}_2$)	sk_{sig}, sk_{prf} $m + 1Z_q$	$vk_{sig}, 1\mathbb{G}_2$ $m\mathbb{G}_T$	✓	✓		
Li20 [34]	$1\mathbb{G}_1$	$1Z_q$	$1\mathbb{G}_2$,id			✓	
Lin21 [37]	$1\mathbb{G}_1$	$1Z_q$	$1\mathbb{G}_2$				C

Table 5.2: Security assumptions per signature scheme

Scheme	Bon09	Cat12	Lin17	Sch17	Sch18	Zha18	Li18	Ara19	Sch19	Li20	Lin21
DL	x			x	x				x		
CDH				x					x		x
DDH					x						
co-CDH	x					x		x		x	x
co-BDH			x								
q-SDH		x					x				
GBDH			x								
FDHI					x				x		

this table, we give an overview of the characteristics of all the signature schemes. It indicates whether a scheme provides context hiding abilities, multi-key support, designated entities, and identity-based key generation.

In Table 5.2 we group the signature schemes by their hardness assumptions. For each hardness assumption, we show which schemes rely upon these assumptions to prove the security of their scheme.

Finally, in Table 5.3 we provide an overview of the security model in which the signature schemes are proven.

Table 5.3: Security model per signature scheme

Model	Bon09	Cat12	Lin17	Sch17	Sch18	Zha18	Li18	Ara19	Sch19	Li20	Lin21
ROM	x		x			x		x		x	x
Standard		x		x	x		x		x		

6

Results

In this chapter, we present our results of the duration of the various operations. We first present which scheme is the fastest overall, after which we discuss in detail the performance of the signature schemes per category.

We categorize the signature schemes based on the features they offer. These features are network coding, multi-key, context hiding, identity-based key generation and designated entities.

6.1. Overall

To visualize which signature scheme is the fastest in each of our file- and packet-length configurations, we have created surface plots for each operation. For each file-packet combination, a tile is created in these plots. The colours of the tiles are based on the signature scheme that performs the current operation the fastest. We now discuss our findings per operation.

6.1.1. Signing

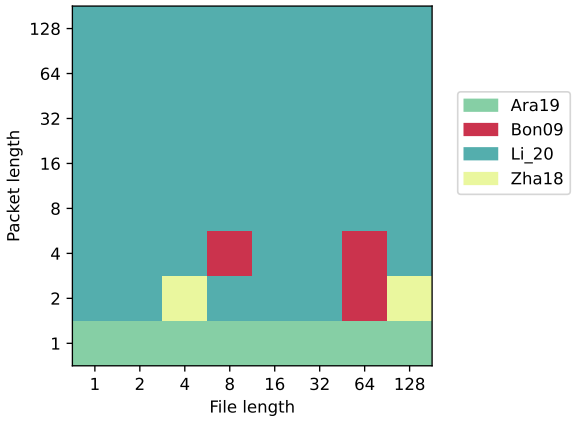


Figure 6.1: Surface plot of fastest signing schemes. Each tile on the surface is coloured corresponding to the scheme that is fastest in that file-packet length configuration. The length of the file and packet increase with steps of powers of two.

In Figure 6.1 we clearly see that Li20 is the fastest scheme in most file-packet length configurations. Looking at this surface plot, we can make two interesting observations. Above

a packet length of 4, Li20 is always the fastest scheme. Further, when the packet length is exactly one, Ara19 is actually the fastest scheme. This scheme is designed in such a way that it is easy to understand, to help introduce people to the concept of homomorphic signatures. As a side effect, this has resulted in a signature scheme that is very fast for signing single messages.

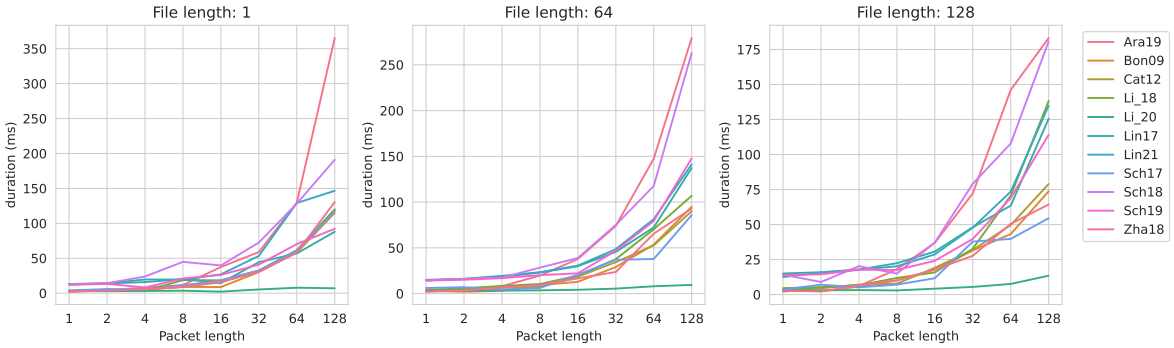


Figure 6.2: Sign duration for constant file length (1,64,128) and varying packet length.

To look at the overall performance of each scheme, we have plotted the sign duration for a constant file length and a varying packet length. In Figure 6.2 we see that as the packet length increases, the scheme Li20 seems to be the least affected. Where the signing duration of other schemes rapidly increases as the packet length grows, Li20 has a slower increase in duration.

Further, we see that Ara19 and Sch18 are affected the most by the increase in packet length. Ara19 is not designed for packets (vectors of numbers) but instead works on single messages. When we want to sign a packet using Ara19, we thus need to create a signature for each part of the vector, making this scheme packet length dependent. Other schemes that do have a mechanism to deal with vectors of messages are seen to not increase as rapidly as these other schemes.

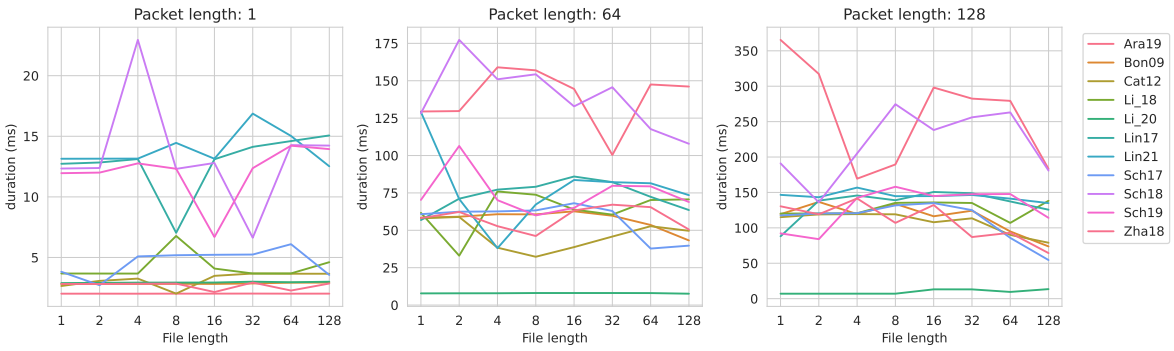


Figure 6.3: Sign duration for constant packet length (1,64,128) and varying file length.

To look at the impact of the file length on the duration of the sign operation, we have created a line plot for constant packet lengths with variable file lengths. When we look at these graphs in Figure 6.3, we see that changing the file length does not impact the signing duration of single packets much. Apart from what seems to be outliers, we observe a mostly consistent signing duration. The duration of the sign operation is thus not dependent on the file length. For the network coding signature schemes, this is a side effect of the optimization we wrote.

Had we not created this optimization, the sign duration would greatly be dependent on the file length, as the number of hash operations, as well as group additions and exponentiations, would depend linearly on the length of a file.

6.1.2. Verifying

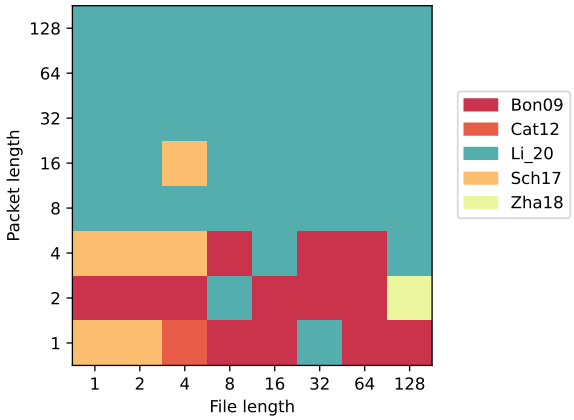


Figure 6.4: Surface plot of fastest verifying schemes.

When we look at the overall fastest verifying schemes in the surface-plot in Figure 6.4, again, we see that Li20 is the best performing scheme out of all the signature schemes, as most tiles are coloured by Li20. For lower packet lengths, Sch17, Bon09, Cat12 and Zha18 are sometimes the fastest schemes, while when the packet length reaches above four, Li20 is again the fastest verifying scheme. We can explain this similarity between the results of the signing and verification algorithms because both operations are similar in nature. To verify if a signature matches a message, the signature is reconstructed up until the point where it is bound to a secret key. We can see this in the verification algorithm of Bon09 in Algorithm 1 and of Ara19 in Algorithm 13. During verification, it is instead paired to a public key, which makes comparing to the signature possible. So since these operations are inherently similar, the duration results are similar as well.

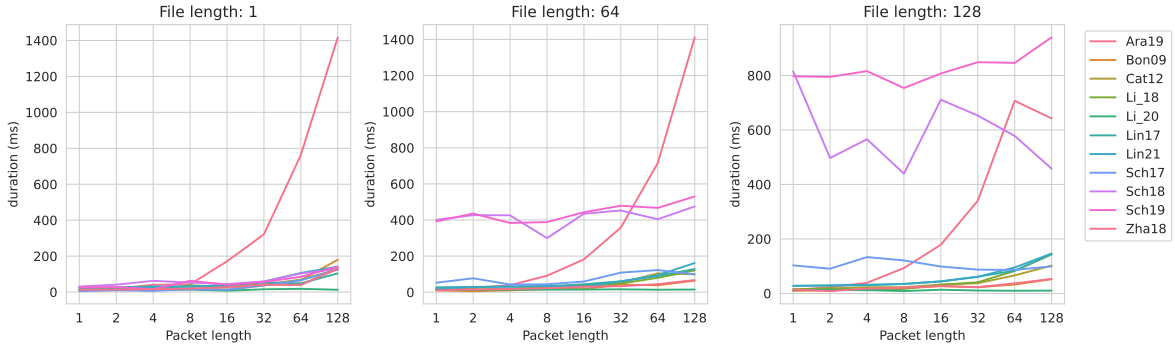


Figure 6.5: Verify duration for constant file lengths (1,64,128) and varying packet lengths. Ara19 depends on the packet length and is therefore seen to increase linearly as the packet length increases. Context hiding multi-key schemes perform pairing operations based on the file length and are therefore seen to increase in duration as the file length increases throughout the subplots.

In most cases, the results are similar to the results of the signing operation. When we look at Figure 6.5, there are notable differences in the context hiding schemes. While before they did

take longer to sign messages, the differences have now become more apparent. The increase in duration for these two schemes is very significant, as the duration is orders of magnitudes larger than all other schemes, except for Ara19.

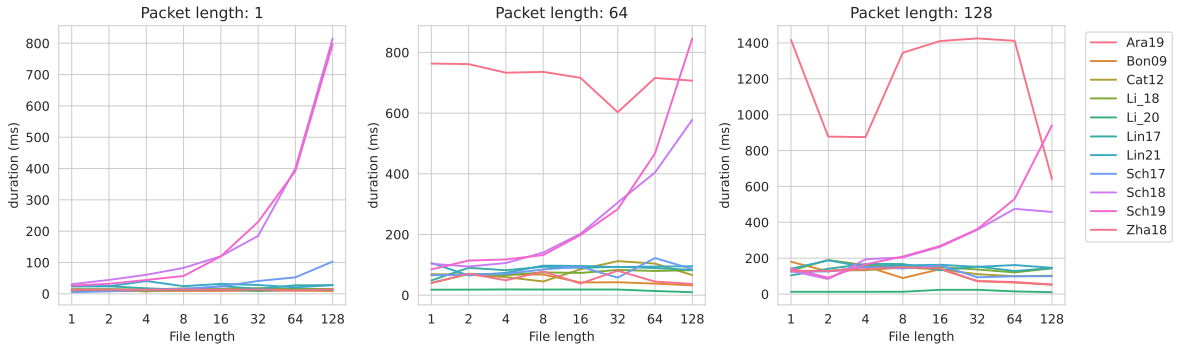


Figure 6.6: Verify duration for constant packet length (1,64,128) and varying file length.

Furthermore, the verification duration for these two schemes is now dependent on the file length, while before this did not have any impact on the duration. This can clearly be observed in Figure 6.6. Where the other schemes behave mostly constant for a changing file length, the context hiding multi-key schemes instead increase linearly with the length of the file.

Lastly, the network coding signature schemes have been optimized with regard to the verify operation. The previously discussed optimization is not applicable to the context hiding multi-key signature schemes. This makes the difference between the CH-MK schemes even larger.

6.1.3. Combining

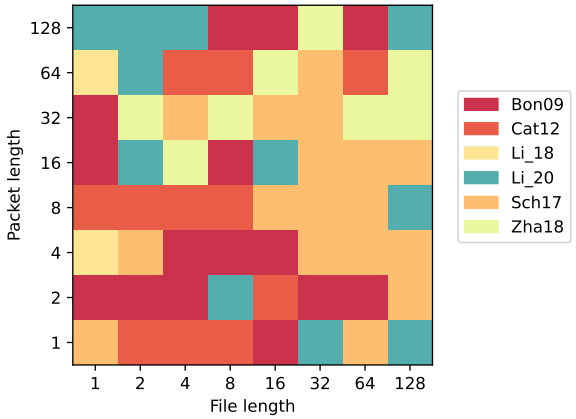


Figure 6.7: Surface plot of fastest combining schemes.

When we look for the overall fastest combining scheme in the surface plot in Figure 6.7, we see that there is no clear better scheme this time. The surface plot is very scattered, which indicates that there is no one scheme that is clearly better. This is because for the signature schemes where a signature is 1 element of $\mathbb{G}_1$, the combing operation is always the same. As we recall, the basics of combining signatures together are exponentiating and point additions. Due to the constant time implementation of the operations in the groups of BLS12-381, the combine operation is time independent from the signature value and the combination coeffi-

cients.

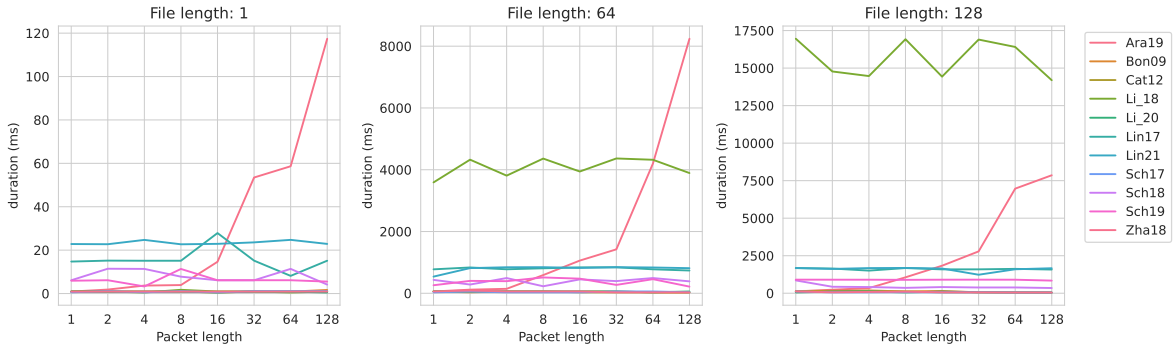


Figure 6.8: Combine duration for constant file length (1,64,128) and varying packet length.

When we look at each scheme's performance in Figure 6.8, we see that most schemes behave very constant. We see that the packet length does not influence the combine duration for most schemes.

Schemes that are affected by the combine operation are Ara19 and Li18. For Ara19 this is the case because it needs to compensate for lacking vector support, while for Li18 the multi-key support requires more operations from the combine algorithm.

When we zoom in on the fastest signers we saw in Figure 6.7, we see in Figure 6.9 that the performance of these schemes is indeed very similar.

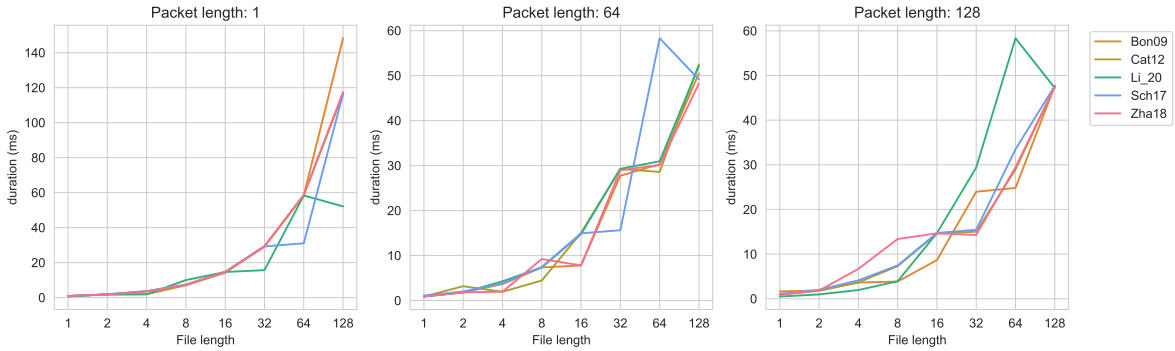


Figure 6.9: Combine duration of fastest combining signature schemes

6.1.4. Summary

We have observed the signature scheme Li20 is the overall fastest at signing messages and verifying signatures. The performance of the combine operation of signature schemes that have signatures which are a single element of $\mathbb{G}_1$ is very similar. For the specific scenario where packets are of length one, the signature scheme Ara19 is the fastest.

With these results, we can answer our first research question: RQ_1 What is the fastest pairing-based, linearly homomorphic signature scheme, when run on modern hardware?

We can say that for messages that are longer than 4 pieces of data, Li20 is the fastest signature scheme with regard to signing and verifying. With regards to the combine operation, Li20 is amongst one of the fastest schemes.

When packets are not vectors but are instead single messages, Ara19 is actually the fastest

signing scheme. In this setting, Ara19 is 1.5 times faster than Li20, as signing a single message takes 2 milliseconds for Ara19 while Li20 takes around 3 milliseconds. This result does not transfer over to verification, however.

6.2. Performance per scenario

In this section, we discuss the performance of the schemes based on the categorization of their features. We categorize based on whether a scheme supports network coding, multi-key, context hiding, identity-based key generation and finally whether they can assign designated entities.

6.2.1. Network coding

When we look at the results for schemes designed for network coding in Figure 6.10 and Figure 6.11, we immediately notice that Li20 is the fastest scheme in this category.

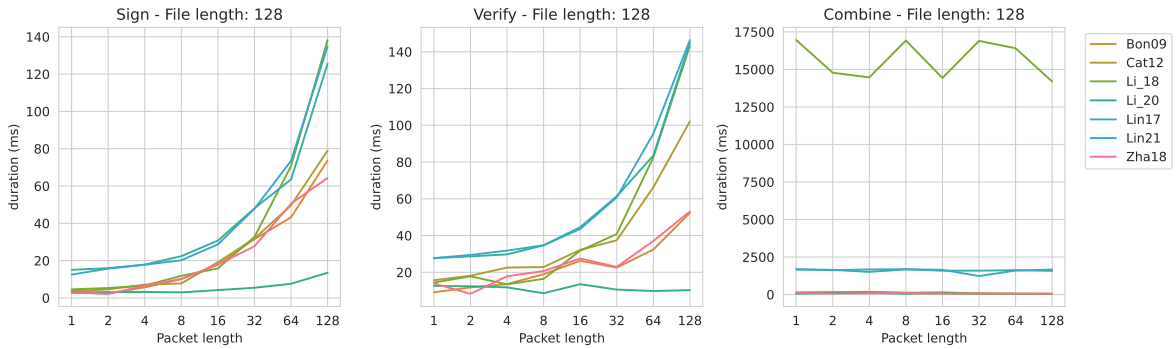


Figure 6.10: Sign, verify and combine operations duration of network coding schemes - Varying packet length for a constant file length of 128.

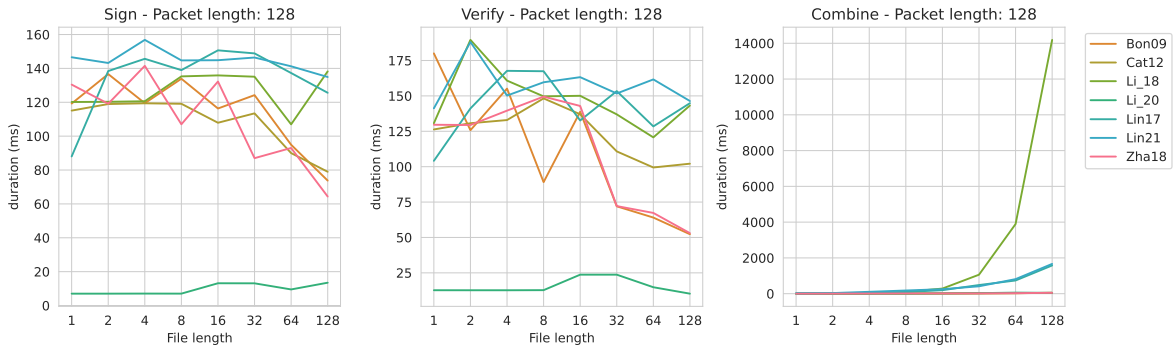


Figure 6.11: Sign, verify, combine duration for network coding schemes - Varying file length on a constant packet length of 128.

Looking at Figure 6.10, which shows the durations for constant file length and varying packet length, we see that the sign and verify durations of the schemes with designated entities, Lin17 and Lin21, are higher than the other network coding schemes. We can explain this difference because of the actions required to designate a signature to a combiner, which involves hashing and pairing operations. Everything before that operation is very similar in construction to the other schemes in this category. Furthermore, what is interesting, is that as the file length increases, the per-signature duration actually decreases. We can see this in Figure 6.11, as lines in are trending downwards when the file length increase for the sign and verify operations.

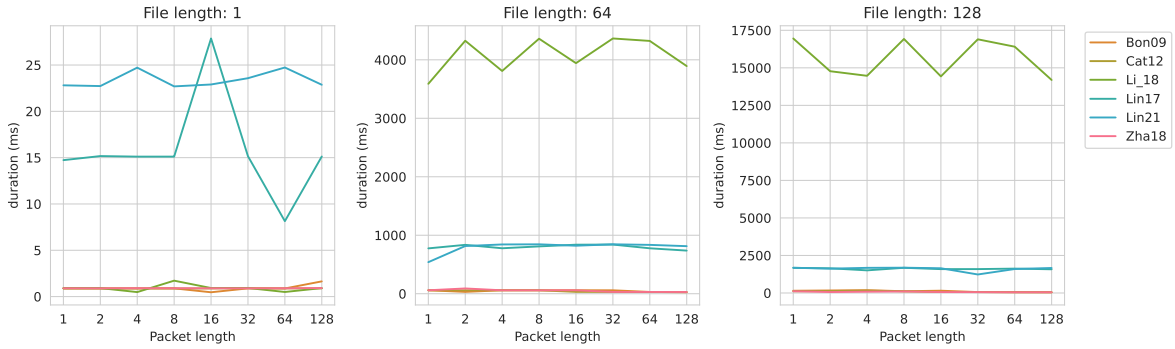


Figure 6.12: Combine duration of network coding schemes for constant file length (1,64,128) and varying packet length.

When we look at the combining performance in Figure 6.12, we first observe that the designated entity schemes take longer to combine their signatures. This is because, in this step, the designated combiner part of the signature gets 'stripped off', and the signature gets further prepared for either public verification or designated verification. Further, this figure shows that as the file length increases through the subplots, the combine duration of Li18 increases drastically. This is due to the way the combine algorithm is created to facilitate multi-key functionality.

6.2.2. Multi-key

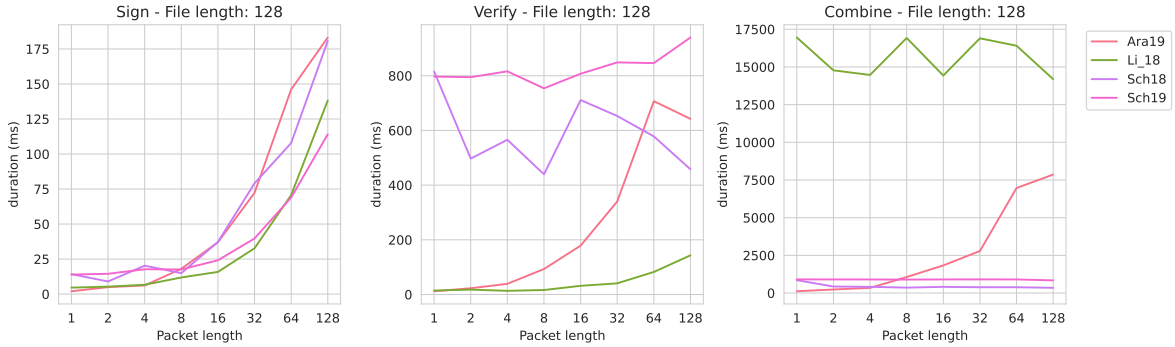


Figure 6.13: Sign, verify, combine duration for multi-key schemes - Varying packet length over a constant file length of 128.

The sign performance of the multi-key signature schemes behaves in a similar way when we look at the plot showing increasing packet length in Figure 6.13. We see that as the packet length increases, the sign duration increases linearly. If we look at the bottom left corner of the signing operation, we see that Ara19 is the fastest signer for packets of length one. To highlight this, we show in Figure 6.15 that Ara19 is always faster than the other schemes as long as the packet length is 1. For larger packet lengths, however, we see that Ara19 signs slower than the other schemes, closely tied with Sch18. Li18 and Sch19 both perform better for larger packet lengths.

In the middle plots of Figure 6.13 and Figure 6.14 we see that the verification operation is clearly the fastest for Li18. The verification duration of Ara19 increases a lot when the packet length increases. As the file length increases, the context hiding multi-key schemes also take longer to verify signatures. What is interesting is that we see that Sch18 is faster than Sch19.

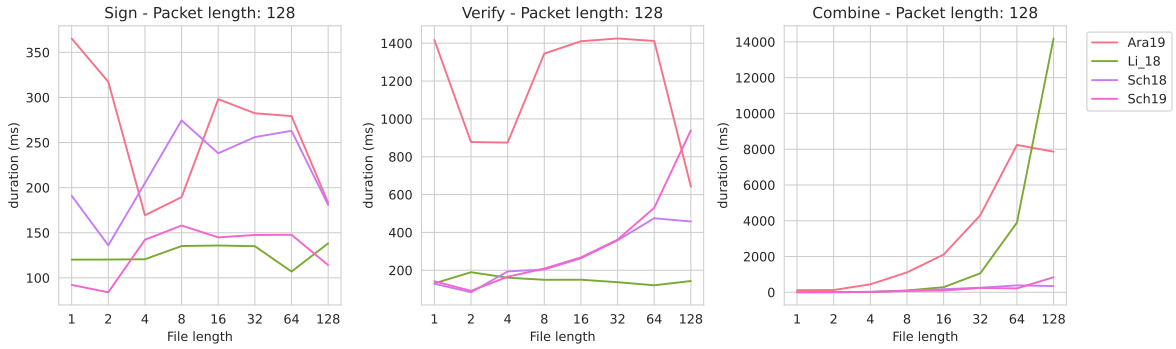


Figure 6.14: Sign, verify, combine duration for multi-key schemes - Varying file length over a constant packet length of 128.

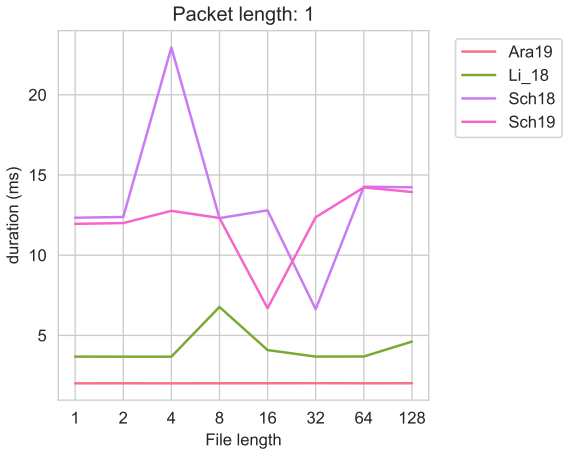


Figure 6.15: Sign duration of MK schemes, varying file length for a constant packet length of 1. In this scenario, Ara19 is faster than all other schemes, followed by Li18.

We had expected Sch19 to be faster due to the optimization that saved a pairing operation with respect to Sch18.

What we see when we look at the combining operation, is that Li18 scales very rapidly with the file length. While its combine duration is mostly constant over a varying packet length (Figure 6.13), the increase in duration based on the file length is very significant (Figure 6.14). It increases in such a rapid matter that at higher file lengths, it is even slower than Ara19. The combine performance of Sch18 and Sch19 is significantly better than that of Ara19 and Li18. While at a low packet length they are slower than Ara19, when the packet length grows above 8, the mostly constant behaviour results in faster combining than Ara19.

6.2.3. Context hiding

Regarding context hiding schemes, there is one scheme that is designed for single key usage, and two schemes support the use of multiple keys. In Figure 6.16 and Figure 6.17, we see that the single key scheme, Sch17, performs the best at signing messages. Further, we see that Sch18 always performs the worst of the three schemes at the sign operation.

When we look at the verifying performance, on the other hand, we see in Figure 6.17 that the verifying performance of the multi-key schemes is similar, until the file length grows to 64. From that point on the difference between Sch18 and Sch19 becomes very notable. We also clearly

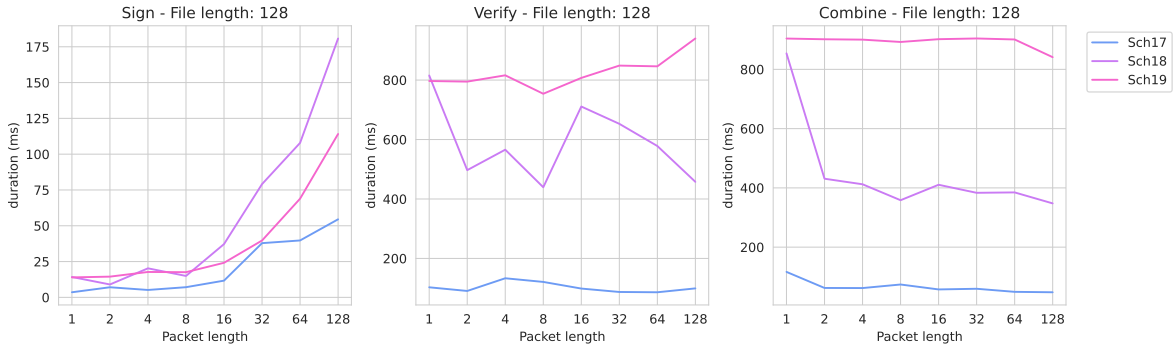


Figure 6.16: Sign, verify and combine duration of context hiding schemes - Varying packet length over a constant file length of 128.

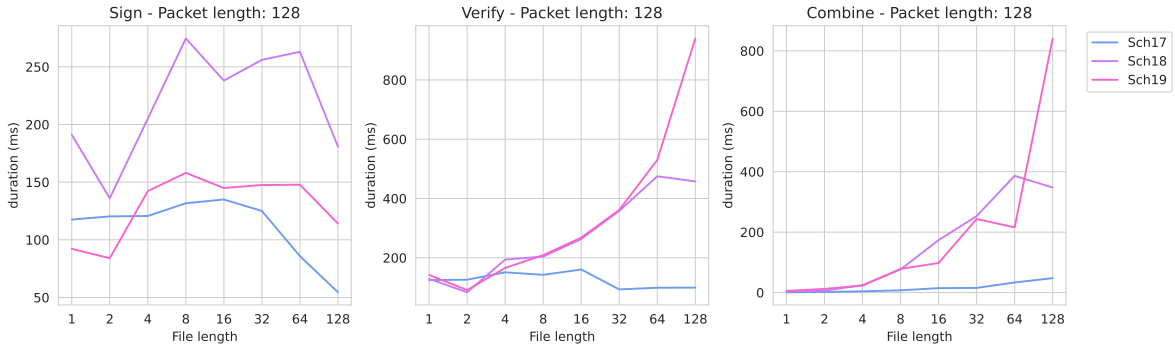


Figure 6.17: Sign, verify and combine duration of context hiding schemes - Varying file length over a constant packet length of 128.

see that the single key scheme verifies considerably faster than the multi-key schemes.

Regarding the combine operation, we can make a similar observation as for the verification operation. Sch18 and Sch19 behave in a similar manner, until the file length grows to 128, when Sch18 overtakes Sch19.

6.2.4. Identity-based key generation

The two schemes that support identity-based key generation have been discussed previously in the network coding category. We see that in all configurations, except for a packet length of one, Li20 outperforms Zha18 with regards to signing and verifying. The combine performance of these schemes is nearly the same since they both perform the except operation.

It would make sense to compare the key generation algorithms of the two signature schemes since that is what sets these two schemes apart from the other ones. However, the key generation is done in the exact same manner for both Zha18 and Li20. We thus keep our discussion to the previously stated results.

6.2.5. Designated Entities

The performance of the schemes with designated entities is very similar in all regards. The signing, verifying and combining durations only differ slightly, and no scheme is *always* faster than the other. This is because both schemes are very similar in construction. Where they start to differ is in the combine operation. Where one scheme prepares the scheme for public verification, the other scheme designates a verifier in this algorithm. We see that these

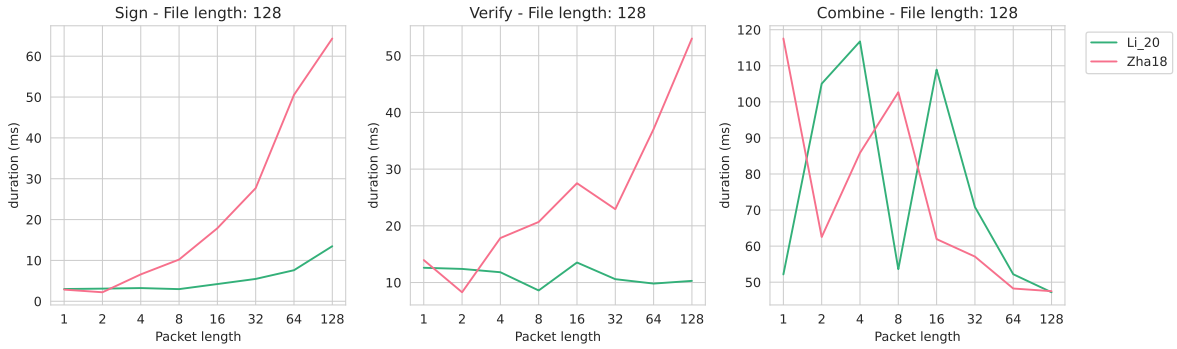


Figure 6.18: Sign, verify and combine duration of ID-B schemes - Varying packet length over a constant file length of 128.

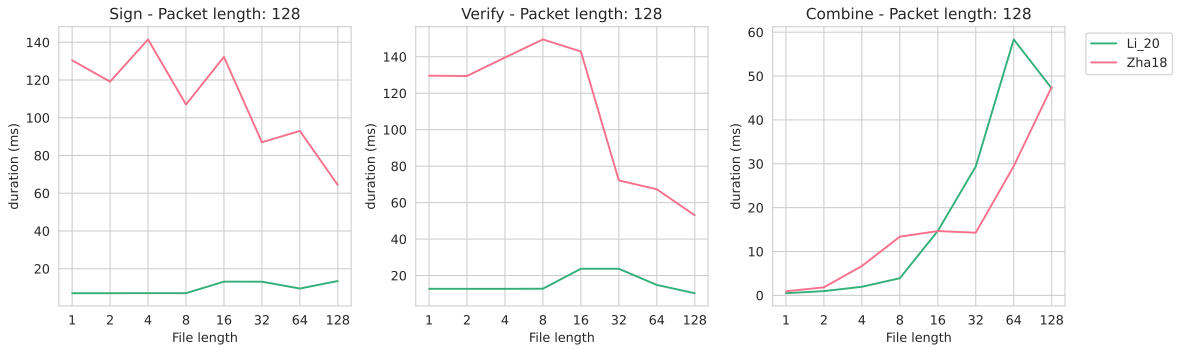


Figure 6.19: Sign, verify and combine duration of ID-B schemes - Varying file length over a constant packet length of 128.

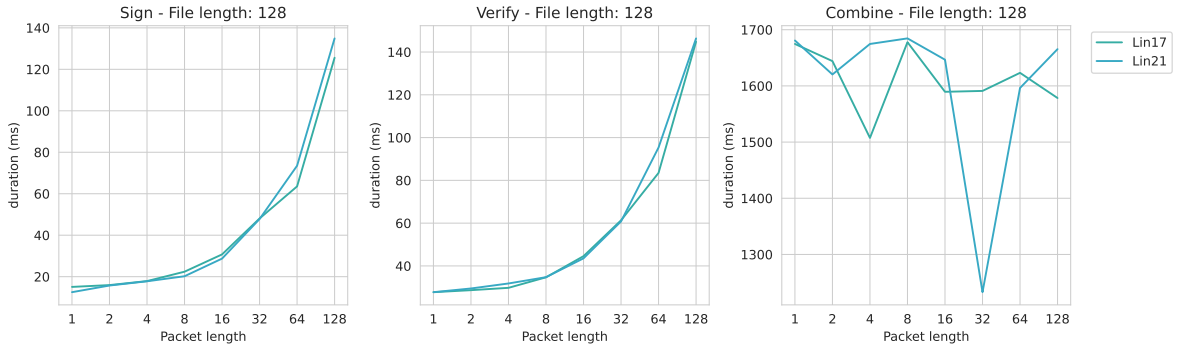


Figure 6.20: Sign, verify and combine duration of DE schemes - Varying packet length over a constant file length of 128.

steps take about the same time, as the combine performance is similar for both schemes as well.

6.3. Overhead of additional features

Finally, in this section, we analyse and present the performance impact of supporting extra features in a homomorphic signature scheme.

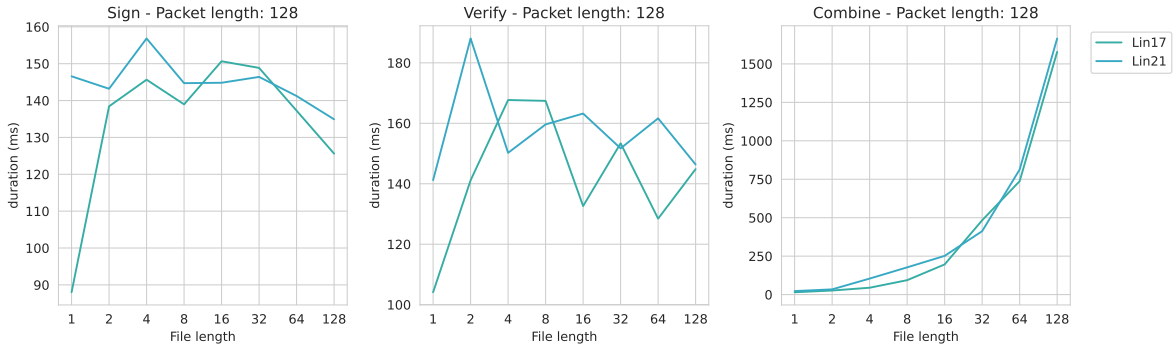


Figure 6.21: Sign, verify and combine duration of DE schemes - Varying file length over a constant packet length of 128.

6.3.1. Network coding

Network coding signatures add a computation to signatures which is based on the unit vector augmented to a packet. Depending on the implementation, this either adds 1 point addition and exponentiation to the sign operation, or it adds m of these operations to the signing process. Further, depending on the construction of a scheme, 1 or m hash operations are performed. This same observation can be made for the verifying operation. The combine performance is not affected by whether a scheme is designed for network coding or not. The size of keys and signatures is not impacted by the network coding property.

6.3.2. Multi-key

The impact of a multi-key homomorphic signature is observed in the combine and verify operations of a signature scheme. As we have seen in the constructions of Ara19 and Sch19, during the combine operation, components are stored depending on the number of identities involved in the homomorphic combination. The verify operation further has to do pairing operations with each verification key of the identities involved in the homomorphic signature. We thus conclude that multi-key support increases the verification duration and the signature size linearly with the number of identities.

6.3.3. Context hiding

To analyse the context hiding overhead, we look at the sign and combine operation of Sch19 and Bon09. Regarding verifying, we consider Sch17 and Bon09.

Context hiding schemes use values that masks the contributions of individual signers. To facilitate these masks, the signing key and the verification key sizes scale with the length of the file, e.g. the number of items to be combined.

Furthermore, the signing algorithm requires operations to be done on these masks. The context hiding schemes in these works use elements of $\mathbb{G}_T$ as masks. Operations on this group are very expensive, so the sign duration of the schemes is considerable longer than schemes that do not offer context hiding.

6.3.4. Identity-based key generation

We have seen that identity-based key generation has no further impact on the sign, verify and combine operations. The only part that differs between these schemes and schemes that use 'traditional' key generation is the use of a key generation centre that is involved with generating users' keys. We see the support of identity-based key generation not as overhead, but as a

system requirement.

6.3.5. Designated Entities

The support for designated entities is facilitated through pairing operations with the designated entity's public key. This operation is more expensive than computing on group elements, and as such, it increases the sign duration based on the time to perform a pairing.

Further, the combine operation is used to strip off its own designation from a signature, after which it is possible to designate an entity for verification or keep the signature publicly verifiable. The 'stripping off' is performed by calculating the inverse of the designation part of the signature, which means that a pairing operation is again involved in this process.

7

Application Analysis

In this chapter, we analyse what our findings mean for practical applications. Our analysis is done based on scenarios which define specific constraints of a signature scheme. The scenarios we consider are Network Coding, Smart Grids and Privacy-Preserving.

7.1. Network coding

As we recall, network coding is a technique to increase the throughput and resilience of computer networks. In the network, a source node wants to send a file through a connected network of intermediate nodes, to a target node. To do this, the file gets split up into multiple packets. These packets are first augmented with a unit vector which indicates which part of the file this packet is, before being sent to intermediate nodes. The intermediate nodes do not store and forward individual packets as in traditional networking, but instead, linearly combine the received packets together before forwarding them to the next node. This improves the throughput of the network. When the target node has received enough linearly independent packets, it can decode them to receive the original file back.

A problem that network coding faces is *pollution attacks*. In this attack, an intermediate node sends a corrupt packet instead of a linear combination of the packets it received. This one corrupt packet prevents the entire file from being decoded.

To prevent this attack vector, the source node can employ homomorphic signatures. By first signing the augmented packets before sending them to the network, the intermediate nodes can check the validity of the received packets. When a node encounters a corrupt packet, it can immediately discard it. Otherwise, the node combines the packets together as before, but now also combines the corresponding signatures using the homomorphic property.

What is important for network coding is that verification and combination are quick, to introduce as little computational overhead for intermediate nodes. This could otherwise severely impact the speed at which the network operates.

Furthermore, the size of signatures should be as small as possible, to allow as much room as possible for actual data to be sent.

The packet size in network coding is determined by the Maximum Transfer Unit. This is a value that depends on the transport layer protocol. The most commonly used protocol is ethernet frames over IP. This setting has an MTU of 1500 bytes. When we assume that a signature takes at least 96 bytes (1 element of $\mathbb{G}_1$), we are left with 1406 bytes. We encode our data as

points in the field Z_q, which takes 48 bytes to represent. We can thus have a packet length of roughly 1404/48 = 29.

We thus need to look at the performance of a network coding signature scheme for a packet length of 29. The closest packet length we tested to is 32. As we saw in our results, the scheme Li20 is the fastest scheme in this category with respect to signing and verifying, and the combining operation is amongst the fastest schemes. This scheme is thus clearly the right choice for this task. Verification of a single signature takes at most 16 ms, and combining signatures takes about 1ms per packet to combine.
We can now answer our second research question with regards to network coding:
$RQ_{2,1}$ Which pairing-based, linearly homomorphic signature scheme is the best to use for network coding?

The signature scheme Li20 is the best scheme to use for network coding, as it is the fastest at both the signing and verifying operations in this setting.

7.2. Smart Grid

In the smart grid setting, a measuring device in individual households reports the periodical power consumption to an aggregation node. For the sake of an example, we consider that for each block of houses, there is one central aggregation node. By periodically sending the power usage to this aggregation node, the electricity company can get a fine-grained insight into the power consumption throughout the day. This is not only used for billing purposes but can also provide interesting insights into trends in power usage.

To make sure that the measurement data is authentic, we can have the measuring devices sign the data before sending it to the aggregation node. This can provide the electricity company with the required trust that the measurement data is not tampered with either by a corrupt homeowner or by an entity in the network during transit. In turn, the homeowner can be assured of his total consumption over a period of time, for example, the monthly usage, by validating the homomorphically combined signature on his total power consumption.

The power consumption measuring devices are embedded devices which are designed for the sole purpose of measuring power consumption and reporting it to an aggregation node. As such, these devices do not have a lot of computational resources, nor do they possess a lot of storage capacity. This defines the requirements for a homomorphic signature scheme for the smart grid setting to have a small signing key, which fits in the limited available storage, and a fast signing operation, to account for the little computational resources. This is especially important when the granularity of the measurements increases to sub-second levels.

The aggregation node is considered to be more powerful than the measuring devices, and as such, the combine and verify operations are less constrained in this setting.

The choice for this setting is also affected by the length of messages to be signed. The measuring device can either first collect a number of measurements, before signing and transferring them, or they can sign and transfer each measurement individually. Furthermore, the level of combining is of importance as well. Signatures can either be aggregated by person, or by all residents of a house block. This latter option requires that the signature scheme support multi-party computation.

When single measurements are signed and transferred, we look at the results of the signing operation for packets of length 1. In this specific setting, Ara19 is the fastest scheme. When multiple measurements are aggregated before being reported, the scheme Li20 is the best

	Single key	Multi key
Single message	Ara19	Ara19
Multiple message	Li20	Li18 / Sch19

Table 7.1: Choice matrix for a homomorphic signature scheme to be used in a smart meter setting

choice as it has the fastest signing operation. The signing key size of both these schemes is the same, namely a single element of $\mathbb{G}_1$.

When we want to combine signatures of different households, we can either use Ara19 again for single messages, or we can use Li18 for reporting on a list of measurements.

While we assumed the computational resource of the aggregation node to be better than those of the measuring devices, we must still note that the combine operation of Li18 scales very poorly with a growing file length. The signing performance of Li18 is comparable to that of Sch19, which might make this a better-suited signature scheme for the multi-key, multi-message setting. This scheme, however, has a considerably larger signing key, which scales with the length of a file (which in this setting means the number of measurements that can be combined). This either limits the number of measurements that can be made throughout the day, to make the key fit in the memory, or it requires measuring devices with larger storage capacity. The choice for a multi-key/multi-message signature scheme is thus dependent on the specific system requirements.

We have visualized the choices that can be made in Table 7.1.
We now consider our research question $RQ_{2,2}$ Which pairing-based, linearly homomorphic signature scheme is the best to use in a multi-party setting?We see that the answer to this question depends on multiple considerations. The message length, the sign, verify *and* combine operation, as well as the key sizes, all play a role in deciding the optimal scheme. We thus answer this question with 'it depends on the constraints'.

7.3. Privacy-Preserving

When privacy is of importance, a homomorphically combined signature should reveal as little information as is required. We still need to know who contributed data to be able to verify signatures with the right public keys, but for a homomorphically combined signature, we do not necessarily need to know which entity provided which inputs to a signature. This concept is called context hiding, and it is especially useful when sensitive data such as medical records or measurements are analysed. While we do need to be assured of the authenticity of the data, we also need to take into account the privacy of the patients of which information is analysed.

The addition of privacy is most important in a multi-key setting, in which contributions of multiple parties are handled.

When choosing a privacy-preserving homomorphic signature scheme, we thus consider Sch18 and Sch19. Looking at the performance of these two signature schemes, we see that we must make a trade-off. The signing speed of Sch19 is faster than that of Sch18, while the verification and combination performance of Sch18 is faster.

This means that the choice depends on which operation occurs more often, or on which entity has access to more computational resources.

When the data signer is constrained to less computationally efficient hardware, the faster sign-

ing scheme Sch19 would be the right choice, while if the requirements indicate that verification and combination performance are more important, one would choose Sch18.
To answer our research question $RQ_{2,3}$ Which pairing-based, linearly homomorphic signature scheme is the best to use for a privacy-preserving setting?We again conclude that *it depends*. When the requirements indicate that signing performance is more important than verifying and combining, Sch19 is the right choice. If however verification and combination of signatures are weighted more, Sch18 is the better choice.

8

Discussion

8.1. Conclusion

Over the years, numerous homomorphic signatures have been designed. Their real-world application and evaluation have not been greatly analysed. Therefore, in this work, we set out to assess the performance of pairing-based, linearly homomorphic signature schemes. Our assessment consists of timing the duration of the sign, verifying and combining operations of different signature schemes, comparing sizes of keys and signatures, and analyzing the impact of supporting additions to homomorphic signature schemes.

To test the performance of the signature schemes under different settings, we have tried various configurations of file- and packet lengths.

By looking at the operation performance, we have identified the signature scheme that is fastest overall, as well as which schemes perform well in one of several categories. We have analyzed what impact our findings have on real-world applications by working out usage scenarios.

To answer our first research question, RQ_1 *What is the fastest pairing-based, linearly homomorphic signature scheme, when run on modern hardware?* We can say that our main finding is that the construction of Li20 is significantly faster than that of other schemes. Furthermore, it scales very well with larger file and packet sizes, and thus it is a great choice for signing larger datasets. By substituting group operations with hashing and scalar additions, a lot of time is saved in the signing and verifying algorithms.

Our second research question, RQ_2 *Which pairing-based, linearly homomorphic signature scheme is the best to use for network coding, multi-party computation or a privacy-preserving setting?*, has been answered in chapter 7. For a signature scheme in a multi-party setting, the result is not one-sided. We find that Li20 is the best choice for network coding, a consideration between Ara19, Sch19 and Li18 has to be made for a multi-party setting, and consideration between Sch18 and Sch19 must be made for a context-hiding signature scheme. While for the latter two scenarios this does not provide a single answer, we do provide the means to make a decision based on the specific requirements of a setting.

To answer our third research question, RQ_3 *What is the performance cost of supporting additional features to homomorphic signature schemes?* We found that NC increases the signing and verifying duration by up to m operations on points in $\mathbb{G}_1$ (m being the length of the file to be transmitted). The main impact of MK support is found in the verification procedure, which

requires at least a pairing operation for each identity involved in a homomorphically combined signature. DE schemes increase the duration of the signing procedure with a pairing and multiple point additions. ID-based schemes do not influence the duration of the signing, verifying or combining operations. They only change the manner of key generation.

Finally, we consider our main research question, *How can we assess pairing-based, linearly homomorphic signature schemes to be used in practice, based on their performance, and the size of their signatures and keys?* Based on the answers to our sub-research questions, we can say that we can assess the performance of a homomorphic signature scheme by the speed of their signing, verifying and combining operations, and the additional features they possess. The result of RQ_1 has highlighted that the number of point additions and exponentiations greatly influences the performance of the signing operation. This was made very clear by signature scheme Li20, which outperforms other schemes because it substitutes operations on points with operations on the field. We conclude that based on implementation performance, and the sizes of signatures and keys, Li20 is the overall best performing linearly homomorphic signature scheme for a single key setting.

What further influences the performance of a homomorphic signature scheme is multi-key support. We have observed that the verification procedure takes longer than the verification of single-key schemes. A tradeoff between verifying performance and the option to combine signatures from multiple clients must be made. This is the case for context hiding signature schemes as well. This privacy-preserving feature comes at a performance cost regarding signing and especially verifying. Finally, we conclude that identity-based key generation and network coding support do not impact the performance of a signature scheme, as the fastest scheme support both of these additional features.

In this thesis, our goal was to find out how we can assess the performance of a linearly homomorphic signature scheme. The motivation for this research was to aid in the decision-making process of selecting one of many available signature schemes. By implementing and open-sourcing eleven signature schemes, evaluating their performance, and discussing tradeoffs between different schemes, we can say that we have provided the means to make a decision based on the real-world performance of these signature schemes.

8.2. Future work

Future work in the field of studying homomorphic signature performance is to investigate more different signature schemes. For linearly homomorphic signature schemes, this regards constructions based on symmetric pairings, as well as constructions based on the RSA assumption. Furthermore, a comparison of signature schemes which support polynomial functions and fully homomorphic schemes would be very interesting. By studying implementations which offer the same security level, a clear tradeoff between performance and possible functionality offered by a signature scheme could be made.

Another suggestion for future works could be to look into the construction of a modular linearly homomorphic signature scheme. We have seen that while there are a lot of different signature schemes based on type II pairings, the basis of these schemes is still very similar. A modular homomorphic signature scheme could be useful to create a signature scheme which is tailored to specific needs. The modular parts of such a signature scheme would involve key generation (using a PKI or ID-based), multi-key support, designating entities and context hiding. While the separate papers on these topics were discussed in the thesis, based on them this modular system can be designed.

8.3. Limitations

Measuring the duration of the operations was done in such a way that the operation was repeated a number of times, after which the total duration was divided by this number to get an average measurement. If instead, the duration of individual operations was measured, we could have analysed the standard deviation of each scheme, which would have given us more insight into the consistency of the operations. This oversight was realized too late into the work, and no more time was available to run new experiments. In future work, measurements of single operations should be made instead.

Furthermore, our testing was limited to file lengths and packet lengths of up to 128. The steps in which these measurements are taken make it so that more measurements of lower lengths are taken, while measurements of bigger lengths might be more interesting to analyse. If we could run the experiments again, we would increase the measurement steps in steps of 20 or 40, and up to larger file and packet lengths. We noticed however that the experiments as they are currently designed already take very long to complete. Increasing the experiment size thus requires a lot of computation time.

Our measurements were run on a personal laptop, which was not dedicated to only running the experiments. This might have impacted our measurements, causing 'lag spikes' when background tasks occurred. This could explain the spikes we have seen in the graphs in chapter 6. A dedicated testing environment would solve this issue.

In this work, we have not considered the type of adversary against which a signature scheme is proven secure. There can be made a distinction between the capabilities of an active adversary, and those of a passive adversary. Signature schemes which are proven secure against an active adversary provide more security than those that are only secure against a passive adversary. Since we keep our analysis to only involve computational performance, we have not discussed the role of adversaries. This could however be of value for someone making decisions on which signature scheme to use.

Bibliography

[1] Shweta Agrawal et al. “Preventing Pollution Attacks in Multi-source Network Coding”. In: *Public Key Cryptography – PKC 2010*. Ed. by Phong Q. Nguyen and David Pointcheval. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2010, pp. 161–176. isbn: 978-3-642-13013-7. doi: 10.1007/978-3-642-13013-7_10.

[2] R. Ahlswede et al. “Network information flow”. In: *IEEE Transactions on Information Theory* 46.4 (July 2000). Conference Name: IEEE Transactions on Information Theory, pp. 1204–1216. issn: 1557-9654. doi: 10.1109/18.850663.

[3] *Announcing Approval of Federal Information Processing Standard (FIPS) 180-2, Secure Hash Standard; a Revision of FIPS 180-1*. Aug. 2002. url: https://www.federalregister.gov/documents/2002/08/26/02-21599/announcing-approval-of-federal-information-processing-standard-fips-180-2-secure-hash-standard-a (visited on 06/15/2022).

[4] Diego F. Aranha and Elena Pagnin. “The Simplest Multi-key Linearly Homomorphic Signature Scheme”. In: *Progress in Cryptology – LATINCRYPT 2019*. Ed. by Peter Schwabe and Nicolas Thériault. Vol. 11774. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019, pp. 280–300. isbn: 978-3-030-30529-1 978-3-030-30530-7. doi: 10.1007/978-3-030-30530-7_14. url: http://link.springer.com/10.1007/978-3-030-30530-7_14 (visited on 09/16/2021).

[5] Nuttapong Attrapadung and Benoît Libert. “Homomorphic Network Coding Signatures in the Standard Model”. In: *Public Key Cryptography – PKC 2011*. Ed. by Dario Catalano et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2011, pp. 17–34. isbn: 978-3-642-19379-8. doi: 10.1007/978-3-642-19379-8_2.

[6] Nuttapong Attrapadung, Benoît Libert, and Thomas Peters. “Efficient Completely Context-Hiding Quotable and Linearly Homomorphic Signatures”. In: *Public-Key Cryptography – PKC 2013*. Ed. by Kaoru Kurosawa and Goichiro Hanaoka. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2013, pp. 386–404. isbn: 978-3-642-36362-7. doi: 10.1007/978-3-642-36362-7_24.

[7] Elaine Barker. *Recommendation for Key Management Part 1: General*. NIST SP 800-57pt1r4. National Institute of Standards and Technology, Jan. 2016, NIST SP 800–57pt1r4. doi: 10.6028/NIST.SP.800-57pt1r4. url: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf (visited on 06/11/2022).

[8] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. “Constructing Elliptic Curves with Prescribed Embedding Degrees”. In: *Security in Communication Networks*. Ed. by Stelvio Cimato, Giuseppe Persiano, and Clemente Galdi. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2003, pp. 257–267. isbn: 978-3-540-36413-9. doi: 10.1007/3-540-36413-7_19.

[9] Paulo S. L. M. Barreto and Michael Naehrig. “Pairing-Friendly Elliptic Curves of Prime Order”. en. In: *Selected Areas in Cryptography*. Ed. by Bart Preneel and Stafford Tavares. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, pp. 319–331. isbn: 978-3-540-33109-4. doi: 10.1007/11693383_22.

[10] Dan Boneh and David Mandell Freeman. “Homomorphic Signatures for Polynomial Functions”. In: *Advances in Cryptology – EUROCRYPT 2011*. Ed. by Kenneth G. Paterson. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2011, pp. 149–168. isbn: 978-3-642-20465-4. doi: 10.1007/978-3-642-20465-4_10.

[11] Dan Boneh et al. “Signing a Linear Subspace: Signature Schemes for Network Coding”. In: *Public Key Cryptography – PKC 2009*. Ed. by Stanisław Jarecki and Gene Tsudik. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2009, pp. 68–87. isbn: 978-3-642-00468-1. doi: 10.1007/978-3-642-00468-1_5.

[12] Dario Catalano, Dario Fiore, and Luca Nizzardo. “On the Security Notions for Homomorphic Signatures”. In: *Applied Cryptography and Network Security*. Ed. by Bart Preneel and Frederik Vercauteren. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pp. 183–201. isbn: 978-3-319-93387-0. doi: 10.1007/978-3-319-93387-0_10.

[13] Dario Catalano, Dario Fiore, and Bogdan Warinschi. “Efficient Network Coding Signatures in the Standard Model”. In: *Public Key Cryptography – PKC 2012*. Ed. by Marc Fischlin, Johannes Buchmann, and Mark Manulis. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012, pp. 680–696. isbn: 978-3-642-30057-8. doi: 10.1007/978-3-642-30057-8_40.

[14] Dario Catalano, Dario Fiore, and Bogdan Warinschi. “Homomorphic Signatures with Efficient Verification for Polynomial Functions”. In: *Advances in Cryptology – CRYPTO 2014*. Ed. by Juan A. Garay and Rosario Gennaro. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2014, pp. 371–389. isbn: 978-3-662-44371-2. doi: 10.1007/978-3-662-44371-2_21.

[15] Denis Charles, Kamal Jain, and Kristin Lauter. “Signatures for Network Coding”. In: *2006 40th Annual Conference on Information Sciences and Systems*. 2006 40th Annual Conference on Information Sciences and Systems. Mar. 2006, pp. 857–863. doi: 10.1109/CISS.2006.286587.

[16] Yvo Desmedt. “Computer security by redefining what a computer is”. In: *Proceedings on the 1992-1993 workshop on New security paradigms - NSPW ’92-93*. Proceedings on the 1992-1993 workshop. Little Compton, Rhode Island, United States: ACM Press, 1993, pp. 160–166. isbn: 978-0-8186-5430-5. doi: 10.1145/283751.283834. url: http://portal.acm.org/citation.cfm?doid=283751.283834 (visited on 05/13/2022).

[17] W. Diffie and M. Hellman. “New directions in cryptography”. In: *IEEE Transactions on Information Theory* 22.6 (Nov. 1976). Conference Name: IEEE Transactions on Information Theory, pp. 644–654. issn: 1557-9654. doi: 10.1109/TIT.1976.1055638.

[18] Naina Emmanuel et al. “Structures and data preserving homomorphic signatures”. In: *Journal of Network and Computer Applications* 102 (Jan. 2018), pp. 58–70. issn: 10848045. doi: 10.1016/j.jnca.2017.11.005. url: https://linkinghub.elsevier.com/retrieve/pii/S1084804517303739 (visited on 07/22/2021).

[19] Dario Fiore et al. “Multi-Key Homomorphic Authenticators”. In: (2018), p. 41.

[20] David Mandell Freeman. “Improved Security for Linearly Homomorphic Signatures: A Generic Framework”. In: *Public Key Cryptography – PKC 2012*. Ed. by Marc Fischlin, Johannes Buchmann, and Mark Manulis. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012, pp. 697–714. isbn: 978-3-642-30057-8. doi: 10.1007/978-3-642-30057-8_41.

[21] Gerhard Frey and Hans-Georg Ruck. “A Remark Concerning m-Divisibility and the Discrete Logarithm in the Divisor Class Group of Curves”. en. In: *Mathematics of Computation* 62.206 (Apr. 1994), p. 865. issn: 00255718. doi: 10.2307/2153546. url: https://www.jstor.org/stable/2153546?origin=crossref (visited on 06/15/2022).

[22] Rosario Gennaro and Daniel Wichs. “Fully Homomorphic Message Authenticators”. In: *Advances in Cryptology - ASIACRYPT 2013*. Ed. by Kazue Sako and Palash Sarkar. Red. by David Hutchison et al. Vol. 8270. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 301–320. isbn: 978-3-642-42044-3 978-3-642-42045-0. doi: 10.1007/978-3-642-42045-0_16. url: http://link.springer.com/10.1007/978-3-642-42045-0_16 (visited on 09/22/2021).

[23] Rosario Gennaro et al. “Secure Network Coding over the Integers”. In: *Public Key Cryptography – PKC 2010*. Ed. by Phong Q. Nguyen and David Pointcheval. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2010, pp. 142–160. isbn: 978-3-642-13013-7. doi: 10.1007/978-3-642-13013-7_9.

[24] Craig Gentry. “Fully homomorphic encryption using ideal lattices”. In: *Proceedings of the forty-first annual ACM symposium on Theory of computing*. STOC ’09. New York, NY, USA: Association for Computing Machinery, May 31, 2009, pp. 169–178. isbn: 978-1-60558-506-2. doi: 10.1145/1536414.1536440. url: http://doi.org/10.1145/1536414.1536440 (visited on 06/13/2022).

[25] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. “A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks”. English. In: *SIAM Journal on Computing* 17.2 (Apr. 1988). Num Pages: 28 Place: Philadelphia, United States Publisher: Society for Industrial and Applied Mathematics, p. 28. issn: 00975397. doi: https://doi-org.tudelft.idm.oclc.org/10.1137/0217017. url: http://www.proquest.com/docview/919828126/abstract/A715F386E2B940FAPQ/1 (visited on 06/14/2022).

[26] Ryo Hiromasa, Yoshifumi Manabe, and Tatsuaki Okamoto. “Homomorphic Signatures for Polynomial Functions with Shorter Signatures”. In: (2013), p. 8.

[27] Markus Jakobsson, Kazue Sako, and Russell Impagliazzo. “Designated Verifier Proofs and Their Applications”. en. In: *Advances in Cryptology — EUROCRYPT ’96*. Ed. by Ueli Maurer. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1996, pp. 143–154. isbn: 978-3-540-68339-1. doi: 10.1007/3-540-68339-9_13.

[28] Don Johnson, Alfred Menezes, and Scott Vanstone. “The Elliptic Curve Digital Signature Algorithm (ECDSA)”. en. In: *International Journal of Information Security* 1.1 (Aug. 2001), pp. 36–63. issn: 1615-5262. doi: 10.1007/s102070100002. url: https://doi.org/10.1007/s102070100002 (visited on 06/15/2022).

[29] Robert Johnson et al. “Homomorphic Signature Schemes”. In: *Topics in Cryptology — CT-RSA 2002*. Ed. by Bart Preneel. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2002, pp. 244–262. isbn: 978-3-540-45760-2. doi: 10.1007/3-540-45760-7_17.

[30] Chris Karlof et al. “Distillation Codes and Applications to DoS Resistant Multicast Authentication”. en. In: (), p. 20.

[31] Jonathan Katz and Brent Waters. “Compact Signatures for Network Coding”. In: (2008), p. 14.

[32] S.-Y.R. Li, R.W. Yeung, and Ning Cai. “Linear network coding”. In: *IEEE Transactions on Information Theory* 49.2 (Feb. 2003). Conference Name: IEEE Transactions on Information Theory, pp. 371–381. issn: 1557-9654. doi: 10.1109/TIT.2002.807285.

[33] Tong Li et al. “A Homomorphic Network Coding Signature Scheme for Multiple Sources and its Application in IoT”. In: *Security and Communication Networks* 2018 (June 14, 2018), pp. 1–6. issn: 1939-0114, 1939-0122. doi: 10.1155/2018/9641273. url: https://www.hindawi.com/journals/scn/2018/9641273/ (visited on 11/11/2021).

[34] Yumei Li, Futai Zhang, and Xin Liu. “Secure Data Delivery with Identity-based Linearly Homomorphic Network Coding Signature Scheme in IoT”. In: *IEEE Transactions on Services Computing* (2020). Conference Name: IEEE Transactions on Services Computing, pp. 1–1. issn: 1939-1374. doi: 10.1109/TSC.2020.3039976.

[35] Benoît Libert et al. “Linearly homomorphic structure-preserving signatures and their applications”. In: *Designs, Codes and Cryptography* 77.2 (Dec. 1, 2015), pp. 441–477. issn: 1573-7586. doi: 10.1007/s10623-015-0079-1. url: https://doi.org/10.1007/s10623-015-0079-1 (visited on 02/14/2022).

[36] Cheng-Jun Lin et al. “Linearly Homomorphic Signatures with Designated Entities”. In: *Information Security Practice and Experience*. Ed. by Joseph K. Liu and Pierangela Samarati. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, pp. 375–390. isbn: 978-3-319-72359-4. doi: 10.1007/978-3-319-72359-4_22.

[37] Chengjun Lin, Rui Xue, and Xinyi Huang. “Linearly Homomorphic Signatures with Designated Combiner”. In: *Provable and Practical Security*. Ed. by Qiong Huang and Yu Yu. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 327–345. isbn: 978-3-030-90402-9. doi: 10.1007/978-3-030-90402-9_18.

[38] Victor S. Miller. “The Weil Pairing, and Its Efficient Calculation”. en. In: *Journal of Cryptology* 17.4 (Sept. 2004), pp. 235–261. issn: 0933-2790, 1432-1378. doi: 10.1007/s00145-004-0315-8. url: http://link.springer.com/10.1007/s00145-004-0315-8 (visited on 06/15/2022).

[39] Wouter Penard and Tim van Werkhoven. “On the Secure Hash Algorithm family”. en. In: (), p. 17.

[40] Yumi Sakemi et al. *Pairing-Friendly Curves*. Internet Draft draft-irtf-cfrg-pairing-friendly-curves-08. Num Pages: 54. Internet Engineering Task Force. url: https://datatracker.ietf.org/doc/draft-irtf-cfrg-pairing-friendly-curves-08 (visited on 06/15/2022).

[41] Lucas Schabhüser, Johannes Buchmann, and Patrick Struck. “A Linearly Homomorphic Signature Scheme from Weaker Assumptions”. In: *Cryptography and Coding*. Ed. by Máire O’Neill. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, pp. 261–279. isbn: 978-3-319-71045-7. doi: 10.1007/978-3-319-71045-7_14.

[42] Lucas Schabhüser, Denis Butin, and Johannes Buchmann. *Context Hiding Multi-Key Linearly Homomorphic Authenticators*. 629. 2018. url: https://eprint.iacr.org/2018/629 (visited on 02/15/2022).

[43] Lucas Schabhüser, Denis Butin, and Johannes Buchmann. “Context Hiding Multi-key Linearly Homomorphic Authenticators”. In: *Topics in Cryptology – CT-RSA 2019*. Ed. by Mitsuru Matsui. Vol. 11405. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019, pp. 493–513. isbn: 978-3-030-12611-7 978-3-030-12612-4. doi: 10.1007/978-3-030-12612-4_25. url: http://link.springer.com/10.1007/978-3-030-12612-4_25 (visited on 09/22/2021).

[44] Adi Shamir. “Identity-Based Cryptosystems and Signature Schemes”. en. In: *Advances in Cryptology*. Ed. by George Robert Blakley and David Chaum. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1985, pp. 47–53. isbn: 978-3-540-39568-3. doi: 10.1007/3-540-39568-7_5.

[45] Giulia Traverso, Denise Demirel, and Johannes Buchmann. *Homomorphic Signature Schemes*. SpringerBriefs in Computer Science. Cham: Springer International Publishing, 2016. isbn: 978-3-319-32114-1 978-3-319-32115-8. doi: 10.1007/978-3-319-32115-8. url: http://link.springer.com/10.1007/978-3-319-32115-8 (visited on 05/31/2021).

[46] Frederik Vercauteren. “Optimal Pairings”. In: *IEEE Transactions on Information Theory* 56.1 (Jan. 2010). Conference Name: IEEE Transactions on Information Theory, pp. 455–461. issn: 1557-9654. doi: 10.1109/TIT.2009.2034881.

[47] Yudi Zhang et al. “An Efficient Identity-Based Homomorphic Signature Scheme for Network Coding”. In: *Advances in Internetworking, Data & Web Technologies*. Ed. by Leonard Barolli, Mingwu Zhang, and Xu An Wang. Lecture Notes on Data Engineering and Communications Technologies. Cham: Springer International Publishing, 2018, pp. 524–531. isbn: 978-3-319-59463-7. doi: 10.1007/978-3-319-59463-7_52.

[48] Fang Zhao et al. “Signatures for Content Distribution with Network Coding”. In: *2007 IEEE International Symposium on Information Theory*. 2007 IEEE International Symposium on Information Theory. ISSN: 2157-8117. June 2007, pp. 556–560. doi: 10.1109/ISIT.2007.4557283.