Abstract

Homomorphic network coding signature is one of the countermeasures designed to combat the pollution attack when applying the vanilla network coding. The project is to compare the computational overhead of homomorphic signatures in random oracle/standard models on the intermediate nodes. The purpose is driven by the fact that a provable scheme in random oracle is not going to be a hundred percent secure when put into practice because of the difficulty of the instantiation of random oracle. If homomorphic signature in standard model has the same performance as the scheme in random oracle, the standard scheme would be preferred. To this end, we select four secure homomorphic network coding signature schemes and divide them into two groups, two RSA-based schemes[6][8] and two bilinear-based schemes[8][9]. The schemes in each group have the same algebraic setting and the comparisons are carried out in each group separately.

To conclude, what has been done in our project is shown as follows:

- A theoretical comparison of the schemes in the same group is discussed in Section 4.5.
- The communication model of homomorphic signature shows the three operations(*vry_NC*(), *vry_Sig*() and *combine*()) that will result in the computational overhead on the intermediate nodes, see Section 5.2.
- The identical implementation of the operation *vry_NC*() of the schemes in the same group will not lead to the time difference; therefore, the comparison will concentrate on *vry_Sig*() and *combine*(), see Section 6.1.
- The time difference of two RSA-based schemes is caused by the *combine*() operation since *vry_Sig*() costs approximately the same time in two schemes, see Section 7.1.4. The reason why the time difference of two *combine*() is the multi-exponentiation, see the Analysis part in Section 7.1.2.
- The time difference of two bilinear-based schemes is caused by the *vry_Sig*() operation since *combine*() costs approximately the same time in two schemes, see Section 7.2.3. The reason why the time difference of two *vry_Sig*() is the exponentiation in bilinear groups, see the Analysis part in Section 7.2.1.
- Homomorphic signatures in random oracle/standard models are not comparable especially when the network coding application is for sending large files and requires a higher security level; whereas, they are somewhat comparable when the size of file is moderate and requires moderate security level, see Section 8.1.

Acknowledgements

I would like to express my deepest thanks to my supervisor, Bogdan Warinschi, for his continuous support to the project, his invaluable comments and his encouragement for the past six months.

I would also like to thank Elizabeth Quaglia from the University of London for her patience and her detailed explanations of my questions concerning the project.

Thanks to my family and friends for their support and encouragement.

Contents

1 Introduction

Network coding [1][25] is a novel and elegant routing technique that serves as an alternative to the traditional network routing scheme. Different from the store and forward mechanism, network coding allows the intermediate nodes to combine the received packets before transmitting. The advantages brought by network coding have made it suitable for wireless and ad-hoc network topologies where a central control is hard. The most noticeable benefit is the significant improvement of network performance, that is, the increase in the information flow in the distributed networks. In addition, network coding offers an approach to improve the network robustness resulting from packet loss since the target node can recover the original file if it receives a sufficient number of correct packets. According to the way the intermediate nodes modify the received packets, network coding is divided into linear and non-linear network coding; our work will focus on linear network coding. Linear network coding is saying that the intermediate nodes make a linear combination on the incoming packets.

While network coding maximally improves the throughput in the network flow, it is susceptible to pollution attacks where the intermediate nodes may behave maliciously and modify the packets in a wrong way before transmitting. The intermediate nodes will combine the corrupted packets into new invalid packets and then propagate them into the network, which will lead to the phenomenon, pollution propagation. There are two main classes of techniques to deal with pollution attack, information-theoretic approaches and cryptographic approaches. The first one introduces redundancy to the packets which significantly increases the overhead in communication. The other one is based on one of the two primitives: homomorphic hash functions and homomorphic signatures. We will give a more detailed description of these two techniques in Section 2.2.

In the work, we will concentrate on homomorphic network coding signatures. According to the oracle the schemes use, there are two kinds of homomorphic signatures: the schemes in random oracle model and the schemes in standard model. As the names imply, the random signatures are defined over random oracle, while the standard signatures work over non-random oracle. The works [5][10] show that to implement a provable secure cryptographic scheme, an appropriate random oracle(always the hash function) is needed; however, it is difficult to find a hash function that generates true random outputs and this has resulted in a secure scheme in random oracle turning out to be insecure when implementing. Consequently, a provable standard scheme which performs at least as well as the random scheme would be preferred to tackle the problem introduced by the instantiation of random oracle. This has put forward to the research of our project,

that is, to compare the performance of homomorphic signatures in random oracle and homomorphic signatures in standard model in terms of the processing delay on the intermediate nodes in the network from a practical point of view.

Specifically, the work will investigate the processing overhead of four secure homomorphic signature schemes in random oracle/standard models according to the cryptographic overhead during transmission. These comparisons are based on the operations on the intermediate nodes that will lead to the computational overhead. Four secure homomorphic network coding signatures are illustrated as follows:

- RSA-based signature scheme in random oracle model proposed by Gennaro *et al.* in [13];
- RSA-based signature scheme in standard model presented by Catalano *et al.* in [11];
- Signature scheme over bilinear groups based on the co-computational Diffie Hellman (co-CDH) assumption in random oracle in [7] by Boneh *et al.*;
- Signature scheme over bilinear groups under q-Strong Diffie-Hellman (q-SDH) assumption in standard model in [11] by Gennaro *et al.*.

Notice that the first two schemes and the last two both have similar algebraic backgrounds and they have been proven to be secure under respective computational assumptions. Therefore, we separate four schemes into two groups, the first two RSA-based schemes in one group(refer to *RSAGroup*) and the last two schemes work over bilinear groups(refer to *BilinearGroup*). Comparisons in each group will be discussed respectively.

Outline of our work. We do not assume any background in network coding and so give a brief introduction of related information of network coding and mathematical basis. In Chapter 2, we give the background of linear network coding and the related techniques on combating pollution attack which is followed by our contributions. Relevant computational assumptions and definitions are illustrated in Chapter 3. Chapter 4 discusses four signature schemes in detail and gives a comparison from a theoretical standpoint at the end. Chapter 5 is the communication models of network coding and homomorphic signature; we will illustrate their operation differences and then conclude what operations will lead to cryptographic overhead. The implementation details of four schemes are given in Chapter 6 and Chapter 7 gives an illustration of implementation results which are associated with theoretical analyses. Chapter 8 concludes what we have done and gives a critical evaluation of our work which is followed by four possible extensions of the project.

2 Background

2.1 Linear Network Coding

In linear network coding [25], a file $\mathcal{V}$ is regarded as a set of n-dimensional vectors $(v^{(1)}, ..., v^{(m)})$ defined over some finite fields or the integers. On the source node, when a file $\mathcal{V}$ is to be transmitted, $m (m \ll n)$ properly augmented vectors $w^{(1)}, ..., w^{(m)}$ are created where $w^{(i)}$ is the $v^{(i)}$ prepended by a unit vector $u^{(i)}$ of length m with 1 in the i-th position and 0 in the others. Then the augmented vectors $w^{(i)}$ can be expressed as $(u^{(i)}, v^{(i)})$. In this way, the set of augmented vectors $(w^{(1)}, ..., w^{(m)})$ forms a basis of a subspace $\mathcal{W}$ in $\mathbb{F}^{m+n}$.

The i-th intermediate node receives μ vectors $w^{(1)}, ..., w^{(\mu)}$ from its incoming edges and combines them into a new vector $w = \sum_{j=1}^{\mu} \alpha_{i,j} w^{(j)}$, where $\alpha_{i,j}$ is chosen from the underlying field. After that, the node sends out the resulting vector through its outgoing edges. The coefficient $\alpha_{i,j}$ used by the i-th intermediate node can be chosen at random by each node, constructed by a central authority or set by the network applications; we will consider the first case which is called random network coding. As illustrated in [25][14][16], random network coding has almost the same performance as network coding with fixed coefficients.

In order to recover the original file, any destination node must receive at least m correct vectors, say $(\overline{w}^{(1)}, ..., \overline{w}^{(m)})$. Each received vector has the form described above, that is $\overline{w}^{(i)} = (\overline{u}^{(i)}, \overline{v}^{(i)})$, such that $\overline{u}^{(i)}$ is the left-most m positions and $\overline{v}^{(i)}$ is the right-most n positions. Let $\overline{\mathcal{V}}$ be the matrix of m vectors $(\overline{v}^{(1)}, ..., \overline{v}^{(m)})$ and $\overline{U}$ be $(\overline{u}^{(1)}, ..., \overline{u}^{(m)})$. The original file can then be obtained from

$$\mathcal{V} = \overline{\mathcal{U}}^{-1} \cdot \overline{\mathcal{V}}.$$

In fact, the linear network coding scheme can be applied to the original file set $(v^{(1)}, ..., v^{(m)})$ but not necessarily to the properly augmented vectors $w^{(1)}, ..., w^{(m)}$; however, in our work, we assume that the scheme is running on a properly augmented basis.

2.2 Dealing with Pollution Attack

Linear network coding described above serves as a technique to successfully recover the original file and improves the network robustness in the network since target nodes can recover the original file from any m uncorrupted and linearly independent packets. However, the honest behaviour of all involved nodes and a secure communication channel are required for network coding to work correctly. As mentioned in Chapter 1, an intermediate node may behave maliciously and

modify the packets in a bad way before sending them into the network; this will result in pollution propagation and the impossible reconstruction of the file.

Before we examine the methods of combating pollution propagation, it is meaningful to discuss two trivial schemes that are proposed to deal with the pollution attack but fail to solve the problem [13][11]. One is to sign each packet on the source node before transmitting which is infeasible as the network coding mechanism combines packets on the intermediate nodes(this makes the verification of the combined packet become impossible). The other one is to sign the entire file on the source node which can help the destination nodes to verify a received file; nonetheless, once the invalid packets are injected into the network, the receivers are under a Denial-of-Service attack because the verification process can only be carried out when the files are recovered.

We have mentioned the two main approaches to deal with data pollution when using linear network coding in Chapter 1, information-theoretic approaches and cryptographic approaches.

Information-theoretic approaches. Information-theoretic schemes introduce redundancy to the original packets by using error-correcting code for the receivers to recover the original file if sufficient valid packets are received [15][16][18]. The security of these methods does not depend on computational assumptions but on the additional information in the packets which will significantly increase the communicational overhead. Moreover, to successfully retrieve the original file, this scheme allows only a limited number of corrupted nodes, maliciously modified packets and insecure communication links.

Cryptographic approaches. Cryptographic schemes are based on cryptographic assumptions and can achieve lower overhead in communication than information-theoretic approaches. These methods allow the receiver to verify the incoming packets before combining them; and, more importantly, they make it possible for a target node which receives less than m valid packets to recover a portion of the file. In this way, the cryptographic approach offers an efficient mechanism to control malicious nodes by allowing the honest node to check the correctness of incoming packets. The cryptographic approach is based on either homomorphic hashing [13][17][7] or homomorphic signature[13][7][12].

Our work will concentrate on network coding schemes using homomorphic signatures. A network coding scheme based on homomorphic signatures has the following linear homomorphic property [13][11]: for any vectors a, b and scalars α, β,

the equation $\text{Sign}(\alpha a + \beta b) = \text{Sign}(a)^{\alpha}\text{Sign}(b)^{\beta}$ always holds. In linear network coding, homomorphic property means that there exists a combining algorithm for an intermediate node to produce a valid signature σ on vector $w = \sum_{i=1}^{\mu} \alpha_i w^{(i)}$ upon μ signature $\sigma_1, ..., \sigma_\mu$ corresponding to μ incoming vectors $w^{(1)}, ..., w^{(\mu)}$ and scalar coefficients $\alpha_1, ..., \alpha_\mu$.

Related works. There are a few homomorphic network coding schemes in the random oracle model. Boneh *et al.* first introduce the definition of network coding signature scheme and propose a specific construction of homomorphic network coding signature scheme over bilinear groups in [7](details will be discussed in Section 4.3); this scheme has proved to be secure under the Computational Deffie-Hellman (CDH) assumption. In 2010, another concrete example of homomorphic network coding built on standard RSA assumption is presented by Gennaro *et al.* in [13](details will be discussed in Section 4.1); Catalano *et al.* show that we can apply small coefficients for linear combinations to improve the performance regardless of whether the scheme works over a large finite field or integers. Boneh and Freeman propose one homomorphic network coding signature scheme in [9] based on lattices and its security depends on the problem of searching short vectors in integer lattices; this construction allows the intermediate nodes to verify vectors over binary fields. In later work by Boneh and Freeman in [8], another homomorphic network signature scheme for polynomial functions is introduced; it is the first scheme to evaluate multivariate polynomials rather than linear functions; and it has proved to be secure by using ideal lattices in random oracle.

In 2011, Attrapadung *et al.* [2] introduce the first homomorphic network coding scheme defined over bilinear groups of composite order in standard model; the security of this construction is proved by using the dual encryption technique proposed by Waters [26]. However, the scheme has resulted in expensive computational overhead because of the dependence on groups of composite orders; and even though the order of bilinear groups is changed to a prime number, the overhead is still expensive when compared to other signature schemes in random oracle model. In the same year, Catalano *et al.* apply the notion of Adaptive Pseudo-free groups in homomorphic network coding signature in [12]; its security has been proved under strong RSA assumption in standard model. In comparison to the first RSA-based scheme in random oracle [13], the computational overhead of this scheme is not so significant; nonetheless, the size of the signature is much larger because of the larger random exponent. The two most recent signature schemes in standard model are presented by Catalano *et al.* in [11] which we are going to discuss in our project, one is based on RSA assumption and the other one on q-SDH assumption.

2.3 Our Contributions

In this project, we examine how the oracle models(random oracle and standard) affect the performance of secure homomorphic network coding signature schemes from the perspective of computational overhead. To this end, the two most recent signature schemes in standard model are chosen and compared with the other two in random oracle defined over the same algebraic backgrounds. Four homomorphic signatures are divided into two groups, two RSA-based schemes(*RSAGroup*) and two bilinear-based schemes(*BilinearGroup*); and the comparison is carried out in each group separately.

In Chapter 5, we will see that three operations(*vry_NC*(), *vry_Sig*() and *combine*()) is going to cause the computational overhead in homomorphic signature; among those operations, the comparison in our work concentrates on *vry_Sig*() and *combine*() since the implementations of *vry_NC*() of the schemes in the same group are identical. *combine*() causes the time difference of two schemes in *RSAGroup* because the standard scheme needs to do extra computations(the multi-exponentiation is the most costly one). *vry_Sig*() is the reason of the time difference of two schemes in *BilinearGroup* as four extra computations(the exponentiation in bilinear groups is the most costly one) in bilinear groups are required in standard scheme. In addition, the time difference becomes significant as the number of downloaded blocks, the dimension of the information vectors and the security level increase.

Four homomorphic signatures are implemented and the experimental results are illustrated and analysed closely related to the definitions of the schemes. Based on the results, we have found three key points. At first, schemes in random oracle/standard models are not comparable especially when (1) the number of downloaded blocks on the intermediate nodes is large, (2) the dimension of the initial vectors before prepended with unit vectors is large, and (3) the applications require a higher security level by increasing the size of the security parameters which may also result in the increase in the size of other parameters; the above conditions will significantly raise the time difference of the schemes in different oracle models. Besides, schemes in random oracle/standard models are somewhat comparable when the generation size is small and the security parameter is moderate. This means that to transmit a small file or to use a moderate security parameter, the performance of the standard scheme is in the same efficiency class as that of the random oracle schemes. Small files or a moderate security parameter will greatly save the computation time on the intermediate nodes; however, the schemes in standard model still need more time and the requirement is that the extra time is in a tolerable range. Finally, a more

customized implementation should be introduced to achieve a better network performance. To design application-based signature schemes is essential under the circumstances that efficiency and security are both important while how fast and how secure vary in different applications.

The words 'large', 'small' and 'moderate' we use in the above paragraph are closely related to the network coding applications and can only be discussed when a certain application environment is set; this is not in the scope of our project. A more precise analysis of the schemes in random oracle/standard models will be illustrated in later chapters.

3 Definitions and Preliminaries

In this chapter, we introduce relevant mathematical backgrounds used in our work, bilinear groups, three computational assumptions and the random oracle/standard models, and the definitions of (homomorphic) network coding signatures. Let $negl(n)$ denote a negligible function mapping from N to R^+ where n in this chapter is the security parameter.

3.1 Bilinear Groups

In what follows we give a description of the definition of bilinear groups as done in [7].

Definition 1(Bilinear Groups) A bilinear group can be expressed as a tuple $(\mathbb{G}, \mathbb{G}', \mathbb{G}_T, e, \varphi)$ and the tuple has the following properties:

1. $\mathbb{G}$, $\mathbb{G}'$ and $\mathbb{G}_T$ are three cyclic groups of the same prime order, and the group operations and random sampling in these groups can be computed efficiently.
2. There exists an efficiently computable map $e : \mathbb{G} \times \mathbb{G}' \rightarrow \mathbb{G}_T$ with the following two properties:
   (a) Bilinearity: the equation $e(g^a, h^b) = e(g, h)^{ab}$ holds for any $g \in \mathbb{G}, h \in \mathbb{G}'$ and $a, b \in \mathbb{Z}$.
   (b) Non-degeneracy: if g and h are generators of groups $\mathbb{G}$ and $\mathbb{G}'$ respectively, then $e(g, h)$ is the generator of $\mathbb{G}_T$.
3. There is an efficiently computable isomorphism, $\varphi : \mathbb{G}' \rightarrow \mathbb{G}$.

The cryptographic applications require that the discrete logarithm problem is difficult in the bilinear groups $\mathbb{G}$, $\mathbb{G}'$ and $\mathbb{G}_T$; that is, given g, g^x in the same groups, it is infeasible to find x.

3.2 Computational Assumptions

We set out by introducing the RSA assumption, then the co-CDH assumption, and finally the q-SDH assumption.

To define the RSA assumption, we first consider an RSA experiment $RSA_{\mathcal{A},Gen}(n)$ for a given algorithm $\mathcal{A}$ and security parameter n. The parameter generation phase $\text{Gen}(1^n)$ runs on the input 1^n to obtain (N, e, d), where N is an RSA modulus, integer $e > 0$ is relatively prime to $\phi(N)$ and integer $d > 0$ satisfying

$ed \equiv 1 \pmod{\phi(N)}$. y is chosen randomly from $\mathbb{Z}_N^*$. Algorithm $\mathcal{A}$ is given N, e, y and outputs $x \in \mathbb{Z}_N^*$. The output of the experiment is 1 if the equation $x^e \equiv y \pmod{N}$ holds, and 0 otherwise.

Next, we can give the definition of RSA assumption based on the RSA experiment.

Definition 2(RSA Assumption) The RSA problem is hard relative to RSA experiment $RSA_{\mathcal{A},Gen}(n)$ if for any probabilistic polynomial-time algorithm $\mathcal{A}$ the following inequality holds:

$$Pr[RSA_{\mathcal{A},Gen}(n) = 1] \leq negl(n).$$

Let $\mathbb{G}$ and $\mathbb{G}'$ be two cyclic groups of prime order p, and g and g' are the generators of $\mathbb{G}$ and $\mathbb{G}'$ respectively. We say $(\mathbb{G}, \mathbb{G}')$ are bilinear groups if $\mathbb{G}'$ is isomorphic to $\mathbb{G}$ and there exists a group $\mathbb{G}_T$ satisfying a bilinear map $e : \mathbb{G} \times \mathbb{G}' \to \mathbb{G}_T$.

We describe the co-CDH problem based on the definition of bilinear groups. The co-CDH problem [7] in $(\mathbb{G}, \mathbb{G}')$ is defined as to compute g^x given g, g' and $(g')^x$ where $\mathbb{G}$, $\mathbb{G}'$, g and g' are as defined above. Let $co\text{-}CDH_{\mathcal{A},Gen}(n)$ be a co-CDH experiment; this experiment outputs 1 when the co-CDH problem is solved and 0 if not.

Definition 3(co-CDH Assumption) The co-CDH problem is hard relative to co-CDH experiment co-$CDH_{\mathcal{A},Gen}(n)$ if for any probabilistic polynomial-time algorithm $\mathcal{A}$, the following inequality holds:

$$Pr[co - CDH_{\mathcal{A},Gen}(n)(n) = 1] \leq negl(n).$$

q-Strong Diffie Hellman Assumption(q-SDH) was first introduced by Boneh and Boyen in [6] and its definition is built on the bilinear groups. q-SDH problem can be reduced to the CDH problem.

Definition 4(q-SDH Assumption) q-SDH Assumption in $(\mathbb{G}, \mathbb{G}')$ states that for any probabilistic polynomial-time algorithm $\mathcal{A}$ which is given a $(q+2)$-tuple $(g, g', (g')^x, (g')^{x^2}, ..., (g')^{x^q})$ and outputs a pair $(c, g^{\frac{1}{x+c}})$ satisfying the following inequality:

$$Pr[A(g, g', (g')^x, (g')^{x^2}, ..., (g')^{x^q}) = (c, g^{\frac{1}{x+c}})] \leq negl(n)$$

where x is chosen randomly, $c \in \mathbb{Z}_N^*$ and n is the security parameter.

3.3 Random and Standard Oracle Model

We categorize the homomorphic network coding signature schemes into two classes in terms of the oracle it uses, the random oracle signature and the standard model signature. The schemes in random oracle uses random oracle; while, the schemes in standard model doesn't use random oracle.

The paradigm of random oracle model [5][10] is that a scheme is first designed and proved to be secure using random oracle H and then implemented by instantiating the random oracle H with a hash function $H(x)$. Canetti *et al.* show in [10] that some existing cryptographic schemes are secure when assuming the true random oracle exists; however, they turn out to be insecure when being implemented; that is, in random oracle model, theoretical security does not imply practical security since there does not exist a cryptographic hash function that can mimic a true random oracle. On the contrary, in [5], Bellare *et al.* propose that, random oracles are practical by showing that an appropriately chosen hash function(such as standard hash function used in a non-standard way) is able to retain the provable security of the cryptographic scheme.

The disadvantages random oracle brings have encouraged the research of the scheme in standard model and, in the field of homomorphic network coding, several signature schemes in standard model have been proposed. However, most of them have either worse performance when compared to the schemes in random oracle or expensive communicational overhead. The two most recent standard signature schemes in [11] are said to be in the same efficiency class as the random schemes [13][7] that work over the same algebraic settings; we will examine it from an experimental perspective later.

3.4 (Homomorphic) Network Coding Signature

Boneh *et al.* first abstract the problem of network coding signature scheme in [7]. A network coding scheme is the one that signs a subspace $\mathcal{W} \subset \mathbb{F}_p^{m+n}$ on the source nodes such that, on the intermediate nodes, vector $w \in \mathcal{W}$ will be accepted and $w \notin \mathcal{W}$ will be rejected. In this section, we are going to show the definitions of the (homomorphic) network coding signature scheme and the security notion of network coding signature according to [7].

On the source node, each file will be assigned with a unique file identifier fid for honest nodes to distinguish multiple files during transmission. The

file identifier fid can be the file name or chosen randomly from a predefined samplable set. m is the dimension of vector spaces and n is the upper bound of the dimension of initial information vectors. We assume the file identifiers are chosen from set $\mathcal{I}$ and the dimension of the initial vector is set to l.

Definition 5 A **Network Coding Signature Scheme** is a tuple of three probabilistic polynomial-time algorithms (Gen, Sign, Vrfy) satisfying the following:

Gen(1^k, m, n). The key-generation algorithm Gen takes as input security parameter k and two integers m, n where m and n are as defined above. It outputs a pair of keys (vk, sk) where vk is the verification key and sk is the secret key.

Sign(sk, fid, $\mathcal{W}$). The signing algorithm takes as input a secret key sk, a file identifier $fid \in \mathcal{I}$ and a m-dimensional subspace $\mathcal{W}$ of basis vectors $w^{(1)}, ..., w^{(m)} \in \mathbb{F}^{m+l}$, $1 \leq l \leq n$. It outputs m tuples $(fid, w^{(i)}, \sigma_i)$, $1 \leq i \leq m$.

Vrfy(vk, fid, w, σ). The verification algorithm takes as input a verification key vk, a file identifier $fid \in \mathcal{I}$, a vector w and a signature σ. It outputs 1 if $w \in \mathcal{W}$ or 0 if $w \notin \mathcal{W}$.

It is required that from every k, every (vk, sk) output by Gen(1^k), and all $\mathcal{W} \subset F^{m+l}$, $1 \leq l \leq n$, it holds that Vrfy(vk, fid, w, Sign(sk, fid, $\mathcal{W}$)) = 1 where $w \in \mathcal{W}$.

Security of Network Coding Signatures. To show the security notion of the network coding signature scheme, we describe the game between a challenger and an adversary. At the setup phase, the challenger runs on Gen(1^k, m, n) to obtain (vk, sk) and gives the public key vk to the adversary. At the signature queries phase, the adversary is allowed to query signatures on vector spaces $\mathcal{W} \subset F^{m+l}$, $1 \leq l \leq n$. If the adversary asks for the signature on vector space $\mathcal{W}$, the challenger randomly chooses a file identifier $fid_i \in \mathcal{I}$ and computes $\sigma_i \leftarrow$Sign(sk, fid_i, $\mathcal{W}_i$) and sends σ_i back to the adversary. Finally, the adversary outputs a forgery (fid^*,w^*,σ^*).

The game between the adversary and the challenger is used to define the security notion of network coding signature schemes. The adversary wins the game if Vrfy(vk,fid^*,w^*,σ^*)= 1 and either (1) $fid^* \neq fid_i$, for all i(*type-I forgery*), or (2)$fid^* = fid_i$, for some i, but $w^* \notin \mathcal{W}_i$(*type-II forgery*). A signature scheme is secure if and only if any probabilistic, polynomial-time adversary can

win the game with at most negligible probability.

Homomorphic network coding signature is a special case of network coding signature; it is a network coding signature with the homomorphic property. Next, we give the formal definition of homomorphic network coding signature as done by Boneh *et al.* in [7].

Definition 6 A **Homomorphic Network Coding Signature Scheme** is a tuple of four probabilistic polynomial-time algorithms (Gen, Sign, Vrfy, Combine) satisfying the following:

Gen(1^k, m, n). The key-generation algorithm Gen takes as input security parameter k and two integer m, n where m and n are as defined above. It outputs a pair of keys (vk, sk) where vk is the verification key and sk is the secret key.

Sign(sk, fid, $\mathcal{W}$). The signing algorithm takes as input a secret key sk, a file identifier $fid \in \mathcal{I}$ and a m-dimensional subspace $\mathcal{W}$ of basis vectors $w^{(1)}, ..., w^{(m)} \in \mathbb{F}^{m+l}$, $1 \leq l \leq n$. It outputs m tuples $(fid, w^{(i)}, \sigma_i)$ where σ_i is the signature.

Vrfy(vk, fid, w, σ). The verification algorithm takes as input a verification key vk, a file identifier $fid \in \mathcal{I}$, a vector w and a signature σ. It outputs 1 if $w \in \mathcal{W}$ or 0 if $w \notin \mathcal{W}$.

Combine(vk, fid, $\{(w^{(i)}, \alpha_i, \sigma_i)\}_1^\mu$). We assume the intermediate node receives μ incoming packets. The combining algorithm takes as input a verification key vk, a file identifier $fid \in \mathcal{I}$ and a set of triples $(w^{(i)}, \alpha_i, \sigma_i)$. σ_i is the signature on the vector $w^{(i)}$ and scalar α_i is in $\mathbb{F}$. If all input signatures σ_i are valid, then σ is a valid signature for $\sum_{i=1}^{\mu} \alpha_i w^{(i)}$. It outputs the signature σ.

It is required that from every k, every (vk, sk) output by Gen(1^k), the following holds:

1. for all $fid \in \mathcal{I}$ and all $w \in F^{m+l}$, $1 \leq l \leq n$, if $\sigma \leftarrow$Sign(sk, fid, w), then Vrfy(vk, fid, w, σ) $= 1$.

2. for all $fid \in \mathcal{I}$, any $w^{(i)} \in F^{m+l}$, $1 \leq l \leq n$ and all sets of triples $\{(w^{(i)}, \alpha_i, \sigma_i)\}_1^\mu$, if Vrfy($vk$, fid, $w^{(i)}$, σ_i) $= 1$ for all i, then it holds that Vrfy(vk, fid, $\sum \alpha_i w^{(i)}$, Combine(vk, fid, $\{(w^{(i)}, \alpha_i, \sigma_i)\}_1^\mu$)) $= 1$.

4 Homomorphic Network Coding Signatures

In this chapter, we will illustrate the definitions of four homomorphic network coding signature schemes in random oracle/standard models. The two homomorphic RSA-Based signature schemes will be examined first and then the other two over bilinear groups. Let m be the number of initial vectors and n the upper bound to the dimension of the information vectors and $m, n > 0$. M denotes the upper bound of the coordinates in the initial vectors. Since the values of m and M are set by the network coding applications or the signature schemes in advance, the total length of information in each block v_i is determined if the file size is known. Therefore, we can play with n to adjust M and m in the defined scope. In this chapter, we assume that the dimension of the information vectors l is always equal to n.

4.1 RSA-Based Scheme in Random Oracle *RRSA*

In this section, we give a precise description of the RSA-based scheme in random oracle(to which we refer to *RRSA*, R stands for random oracle).

In *RRSA* construction, parameter L is used to define the upper bound of the distance from the source node to the targets. The benefit of choosing L is to achieve a good decoding probability when the distance is limited to $\leq L$; normally, L is not too large and is fixed to 20 in our implementation. Given L, a bound B is defined to be the upper bound of components in any vector u in transit and satisfies $B = (mQ)^L$. Then the largest possible coordinates in any vector u is BM to which we refer B^*, that is, $B^* = BM$. Prime Q defines the domain $\mathbb{Z}_Q = \{0, ..., Q-1\}$ from which the intermediate nodes choose coefficients for linear combination. Specifically, the intermediate node will compute a linear combination of valid incoming packets over the integers with the coefficients randomly selected from the domain $\mathbb{Z}_Q$. Random oracle is simulated by using a hash function H and this function takes a file identifier fid and index $i(i = 0, ..., m)$ as input to generate m values in the subgroup of quadratic residues QR_N.

The first homomorphic RSA-based signature *RRSA* can be described as a tuple (NGen, NSign, NVrfy, Combine) presented in [13].

NGen(1^k, m, n). The *RRSA* algorithm takes as input security parameter k, the values of m and n and parameters B and B^* defined above. It outputs a pair of keys (vk, sk). The public key vk is $(N, e, g_1, ..., g_n)$ where N is a RSA composite, e is a public exponent which is a prime larger than mB^*

and $g_1, ..., g_n$ are generators of the subgroup of quadratic residues QR_N. The private key sk is d such that $ed \equiv 1 \pmod{\phi(N)}$.

NSign(sk, fid, $\mathcal{W}$). The signing algorithm takes place on the source node. Before transmitting a file, the source node partitions the original file $\mathcal{V}$ into vectors of length n and encodes them into a properly augmented bases $(w^{(1)}, ..., w^{(n)})$, $w^{(i)} \in \mathbb{Z}^{m+n}$. It then randomly chooses a file identifier fid for the file and applies hash function H to generate m values $h_1, ..., h_m$ in field QR_N, $h_i = H(i, fid)$, $i = 1, ..., m$. Finally, the source node computes the signature σ on each w in subspace $\mathcal{W}$:

$$\sigma = (\prod_{i=1}^{m} h_i^{u_i} \prod_{j=1}^{n} g_j^{v_j})^d \mod N$$

where $g_1, ..., g_n$ are parts of the public key vk and $h_1, ..., h_m$ are file-specific. The source then transmits the vector $w^{(i)}$ and its corresponding signature along with the file identifier fid into the network through its outgoing edges.

NVrfy(vk, fid, w, σ). To verify a signature σ on a vector $w = (u_i, ..., u_m, v_1, ..., v_n)$. The intermediate node proceeds as follows:

1. Check the coordinates in the u_i's are positive and smaller than B.
2. Check the coordinates in the v_i's are positive and smaller than B^*.
3. Retrieve the public key vk and compute $h_i = H(i, fid)$ for $i = 1, ..., m$ using the known hash function. Check that the following equation holds

$$\sigma^e = \prod_{i=1}^{m} h_i^{u_i} \prod_{j=1}^{n} g_j^{v_j} \mod N. \qquad (1)$$

4. If all the checks pass, the algorithm outputs 1, or outputs 0.

Combine(vk, fid, $\{(w^{(i)}, \alpha_i, \sigma_i)\}_1^\mu$). Once an intermediate node receives μ incoming packets and each packet has passed the verification check, it randomly chooses μ coefficients $\alpha_1, ..., \alpha_\mu$ from $\mathbb{Z}_Q = \{0, ..., Q-1\}$ and then computes $w^* = \sum_{i=1}^{\mu} \alpha_i w^{(i)}$ and its corresponding signature $\sigma^* = \prod_{i=1}^{\mu} \sigma_i^{\alpha_i}$ mod N.

Efficiency. This RSA-based scheme over integers avoids pairing computations needed in the signature scheme over bilinear groups and uses small coefficients(8-bits) α_i chosen from the domain $\mathbb{Z}_Q = \{0, ..., Q-1\}$. This design helps to reduce the computational overhead since, in the combine algorithm, the multiplications are with small integers and the exponents of the exponentiations are 8-bits integers;

it also reduces the communication overhead when the distance from the source to the targets doesn't exceed the value L. Gennaro *et al* shows that only to choose a small integer Q, such as $Q = 257$, is enough to allow a good performance and also a good decoding probability(the probability that the target node can recover the file correctly). The signing algorithm requires one full exponentiation and one multi-exponentiation; the verification algorithm needs one multi-exponentiation; and the combine algorithm contains one multi-exponentiation and some multiplications and additions. Each signature is one of the elements in $\mathbb{Z}_N$.

4.2 RSA-Based Scheme in Standard Model *SRSA*

We are going to show in detail the RSA-based scheme in standard model(to which we refer to *SRSA*, *S* stands for standard model).

In *SRSA* construction, a parameter λ is applied to specify the domain $\mathcal{I}$ for file identifiers and its relation to M is $2^\lambda > M$. The set of prime numbers of exactly $(\lambda + 1)$ bits forms the space $\mathcal{I}$, and each prime number is greater than 2^λ. The coefficients in the scheme for linear combination are chosen from the domain $\mathbb{F}_e$ where the public exponent e is equal to the file identifier in $\mathcal{I}$, namely, $e = fid$. Since the source node chooses a file identifier fid for each file and each fid determines the domain of the coefficients for linear combination, the finite fields over which linear combination is done are different in files.

This precise *SRSA* scheme is shown in a tuple (NGen, NSign, NVrfy, Combine) as done in [11].

NGen(1^k, fid, m, n). The *SRSA* randomly chooses two safe prime p, q of length $k/2$ and computes $N = pq$. $g, g_1, ..., g_n, h_1, ..., h_m$ are chosen at random from $\mathbb{Z}_N^*$. The security parameter λ is specified to define the domain $\mathcal{I}$ of the file identifiers. The public key vk is set to $(N, g, g_1, ..., g_n, h_1, ..., h_m)$ and the secret key sk is d such that $ed \equiv 1 \pmod{\phi(N)}$.

NSign(sk, fid, $\mathcal{W}$). The signing algorithm is performed on the source node. On input a vector space $\mathcal{W} \subset \mathbb{F}_M^{m+n}$ of the form $(w^{(1)}, ..., w^{(m)})$ and each $w^{(i)} = (u^{(i)}, v^{(i)})$. As specified before, e is a RSA public exponent and $e = fid$. The source chooses an integer s in $\mathbb{Z}_e$ at random and uses its secret key d to compute the signature for each vector $w \in \mathcal{W}$.

$$x = (g^s \prod_{i=1}^{m} h_i^{u_i} \prod_{j=1}^{n} g_j^{v_j})^d \mod N.$$

The signature $\sigma = (s, x)$ is for the vector w. The source node calculates all the signatures corresponding to each vector $w^{(i)} \in \mathcal{W}$ and then sends out to the network through its outgoing edges.

NVrfy(vk, fid, w, σ). On input a vector $w = (u_1, ..., u_m, v_1, ..., v_n)$ and a signature $\sigma = (s, x)$. The verifying algorithm proceeds as follows.

1. Check that e is an odd number of size $\lambda + 1$.
2. Check that all the u_i's, v_i's and s are chosen from $\mathbb{Z}_e$.
3. Check that the following equation holds.

$$x^e = g^s \prod_{i=1}^{m} h_i^{u_i} \prod_{j=1}^{n} g_j^{v_j} \mod N. \qquad (2)$$

4. If all the checks pass, the algorithm outputs 1, or outputs 0.

Combine(vk, fid, $\{(w^{(i)}, \alpha_i, \sigma_i)\}_1^{\mu}$). Once an intermediate node receives μ incoming packets and each packet has passed the verification check, it randomly chooses μ coefficients $\alpha_1, ..., \alpha_\mu$ and computes as follows.

First, computes

$$w^* = \sum_{i=1}^{\mu} \alpha_i \cdot w^{(i)} \mod e. \quad w' = (\sum_{i=1}^{\mu} \alpha_i \cdot w^{(i)} - w)/e.$$

$$s^* = \sum_{i=1}^{\mu} \alpha_i s_i \mod e. \quad s' = (\sum_{i=1}^{\mu} \alpha_i s_i - s)/e.$$

Let $w' = (u', v')$. Then computes

$$x^* = \frac{\prod_{i=1}^{\mu} x_i^{\alpha_i}}{g^s \prod_{i=1}^{m} h_i^{u_i'} \prod_{j=1}^{n} g_j^{v_j'}} \mod N \qquad (3)$$

Finally, it outputs a new signature $\sigma^* = (s^*, x^*)$.

Efficiency. As the *RRSA* construction, *SRSA* avoids using expensive pairing operations. Signing requires one full exponentiation, one multi-exponentiation in domain $\mathbb{Z}_e$ and the selection of a random prime number s. The verification operation costs one exponentiation with a prime of $(\lambda+1)$-bits and one multi-exponentiation with λ-bits exponents. The combining algorithm consists of many arithmetic operations, especially one multi-exponentiation as done in the verification algorithm. Each signature consists of an element in $\mathbb{Z}_N$ and an integer $s \in \mathbb{Z}_e$. Notice that the communication overhead is not going to increase as the (*mod* e) operation in combining algorithm; this allows no limitation in the distance from the source to the targets.

4.3 Scheme over Bilinear Group in Random Oracle *RCDH*

The first homomorphic network coding scheme over bilinear groups is proposed by Boneh *et al.* in [7]; it has proved secure under the co-CDH assumption(to which we refer to *RCDH*).

The signature scheme can be described as a tuple (NGen, NSign, NVrfy, Combine) as done in [7].

NGen(1^k, m, n). On input a security parameter k, and two integers m, n. Do the following:

1. Generate a bilinear group tuple $\gamma = (\mathbb{G}, \mathbb{G}', \mathbb{G}_T, e, \varphi)$ such that $\mathbb{G}$, $\mathbb{G}'$ and $\mathbb{G}_T$ are three bilinear groups of prime order $p > 2^k$ satisfying a bilinear map $\varphi : \mathbb{G} \times \mathbb{G}' \to \mathbb{G}_T$. Randomly pick generators $g_1, g_2, ..., g_N \in \mathbb{G} \setminus \{1\}$ and $g' \in \mathbb{G}' \setminus \{1\}$.
2. Randomly choose s in $\mathbb{F}_p$, and let μ be $(g')^s$.
3. Let denote with $H : \mathbb{Z} \times \mathbb{Z} \to \mathbb{G}$ a hash function.
4. Output a prime p and a key pair (vk, sk) where the public key vk is $(\gamma, H, g_1, ..., g_N, g', \mu)$, while the secret key sk is s.

NSign(sk, fid, $\mathcal{W}$). On input a secret key sk, a file identifier fid, and a vector space $\mathcal{W} = (w^{(1)}, ..., w^{(m)})$ where $w^{(i)} \in \mathbb{F}_p^{m+n}$. It outputs a signature on vector $w \in \mathcal{W}$ by computing

$$\sigma = (\prod_{i=1}^{m} H(fid, i)^{u_i} \prod_{j=1}^{n} g_j^{v_j})^s.$$

The source node computes all signatures for the vectors and sends out the vectors and their corresponding signatures along with the file identifier to the network through its outgoing links.

NVrfy(vk, fid, w, σ). Given a vector $w = (u_1, ..., u_m, v_1, ..., v_n)$, and a signature σ. It outputs 1 if the following equation holds

$$e(\sigma, g') = e(\prod_{i=1}^{m} H(fid, i)^{u_i} \prod_{j=1}^{n} g_j^{v_j}, \mu), \tag{4}$$

otherwise outputs 0.

Combine(vk, fid, $\{(w^{(i)}, \alpha_i, \sigma_i)\}_1^{\mu}$). We assume that the intermediate node receives u incoming packets $w^{(1)}, ..., w^{(\mu)}$ and every packet has passed the verification check. It then randomly chooses u coefficients $\alpha_i \in \mathbb{F}_p$ for linear

combination. The algorithm computes $w^* = \sum_{i=1}^{\mu} \alpha_i \cdot w^{(i)}$ and $\sigma^* = \prod_{i=1}^{\mu} \sigma_i^{\alpha_i}$ and outputs a pair (w^*, σ^*).

Efficiency. The *RCDH* construction has constant-size public key and constant-size per-packet signatures which makes the scheme suited for network coding applications. The signing algorithm requires one full exponentiation and one multi-exponentiation. The verification algorithm needs two pairing computations and one multi-exponentiation. Each signature is one of the elements in $\mathbb{G}$ which means that we can choose as short as possible group elements in $\mathbb{G}$ to reduce the communication overhead.

4.4 Scheme over Bilinear Group in Standard Model *SSDH*

This section is going to discuss the homomorphic network coding signature scheme based on q-SDH in standard model[11](to which we refer to *SSDH*).

This scheme is adapted from the one proposed by Hofheinz and Kiltz [17] which is built on the concept of Programmable Hash Functions. In *SSDH* construction, all operations are defined over $\mathbb{Z}_p$ where prime p is specified by the key generation algorithm; for instance, the $(mod\ p)$ operation in combining algorithm. File identifiers are elements in $\mathbb{Z}_p^*$.

The precise signature scheme is a tuple (NGen, NSign, NVrfy, Combine) as shown in [11].

NGen(1^k, m, n). Generate a bilinear group tuple $\gamma = (\mathbb{G}, \mathbb{G}', \mathbb{G}_T, e, \varphi)$ such that $\mathbb{G}, \mathbb{G}', \mathbb{G}_T$ have prime order $p > 2^k$. $\mathbb{G}, \mathbb{G}', \mathbb{G}_T$ are three bilinear groups satisfying a bilinear map $\varphi : \mathbb{G} \times \mathbb{G}' \to \mathbb{G}_T$ and g and g' are generators of $\mathbb{G}, \mathbb{G}'$ respectively. Do the following:

1. Randomly choose an integer z in space $\mathbb{Z}_p$ and let $Z = (g')^z$.
2. Pick random elements $h, h_1, ..., h_m, g_1, ..., g_n$ from group $\mathbb{G}$.
3. Output a key pair (vk,sk) where the public key vk is set as $(p, Z, h, h_1, ..., h_m, g, g', g_1, ..., g_n)$ and the secret key sk is z.

NSign(sk, fid, $\mathcal{W}$). The signing algorithm takes as input a file identifier fid randomly chosen from $\mathbb{Z}_p^*$, and a properly augmented vector basis $\mathcal{W} = (w^{(1)}, ..., w^{(m)})$ where $\mathcal{W} \subset \mathbb{F}_p^{m+n}$. It chooses a random number $s \in \mathbb{Z}_p$ and computes the signature on w

$$x = (h^s \prod_{i=1}^{m} h_i^{u_i} \prod_{j=1}^{n} g_j^{v_j})^{\frac{1}{z+fid}}.$$

Then the signature for vector w is $\sigma = (x, s)$. The source node calculates m signatures on m vectors and transmits the signatures associated with corresponding vectors into the network.

NVrfy(vk, fid, w, σ). The algorithm verifies a signature σ on the corresponding vector $w = (u_1, ..., u_m, v_1, ..., v_n)$ by checking if the following equation holds:

$$e(x, Z \cdot (g')^{fid}) = e(h^s \prod_{i=1}^{m} h_i^{u_i} \prod_{j=1}^{n} g_j^{v_j}, g'). \tag{5}$$

It outputs 1 when the equation is satisfied, otherwise outputs 0.

Combine(vk, fid, $\{(w^{(i)}, \alpha_i, \sigma_i)\}_1^{\mu}$). We assume that the intermediate node receives μ incoming packets $w^{(1)}, ..., w^{(\mu)}$ where $w^{(i)} \in \mathbb{F}_p^{m+n}$ and every packet has passed the verification check. It then randomly chooses μ coefficients α_i for linear combination from $\mathbb{F}_p$ and computes

$$x^* = \prod_{i=1}^{\mu} x_i^{\alpha_i}. \quad s^* = \sum_{i=1}^{\mu} \alpha_i \cdot s_i \mod p.$$

Finally, it outputs $\sigma^* = (x^*, s^*)$.

Efficiency. Each signature contains an element in group $\mathbb{G}$ and a number in $\mathbb{Z}_p$. A signing operation needs one full exponentiation in $\mathbb{G}$ and one multi-exponentiation in $\mathbb{G}$ and the verification algorithm requires two pairing computations and one multi-exponentiation in $\mathbb{G}$ in addition to two exponentiation(one in $\mathbb{G}$ and the other one in $\mathbb{G}'$) and two multiplications(one in $\mathbb{G}$ and the other one in $\mathbb{G}'$). We will see later that the four extra group operations in contrast to *RCDH* construction are the main cause of the time difference of processing overhead between *RCDH* and *SSDH*.

4.5 Summary and Comparison

After introducing four secure homomorphic signature schemes, we are going to make a comparison of the schemes over different oracle models from a theoretical perspective.

RSAGroup consists of the RSA-based scheme in random oracle by Gennaro et al.[13] and the scheme in standard model by Catalano et al. [11]. In addition to the different oracles they use, the two schemes are distinct in three aspects as discussed in [11]. First, in random oracle model, the RSA-based scheme is to have a better performance only when the distance between the source and the targets

is not too large(say, up to 20 to 30 hops); while, in standard model, the operation ($mod\ e$) in the combining algorithm makes the value of the vector coordinate always less then e which allows no limitation in the path length. Second, the size of the public key in *SRSA* scheme is linear in $(m + n)$ in contrast to the constant key size in *RRSA* scheme since some parameters are generated on the intermediate nodes using random oracle. Finally, the signature size and the time needed for signing and verification algorithms in these two schemes are comparable to some extent. The last point concerns the efficiency which we will look into in the project.

BilinearGroup contains the scheme based on the co-CDH assumption in random oracle in [7] by Boneh *et al.* and the other one under the q-SDH assumption in standard model in [11] by Gennaro *et al.*. The combining algorithm in *SSDH* construction has the advantage of avoiding the communicational overhead growing without limitation by ($mod\ p$) when the distance becomes longer. However, to use standard oracle, the *SSDH* scheme has to do two extra exponentiations(in $\mathbb{G}$ and $\mathbb{G}'$) and two extra multiplications(in $\mathbb{G}$ and $\mathbb{G}'$) in the verification algorithm; this has resulted in the extra computational overhead as the operations which involve bilinear groups are normally costly.

5 Communication Models

In this chapter, we assume that the intermediate nodes have enough memory space and computing power to do expensive computations. We are going to first give a brief introduction of the communication models of network coding and homomorphic signature and then compare them according to the models given.

5.1 Network Coding

According to Figure 1, when a file is to be transmitted, it is divided into m blocks and each block is represented as a vector; the source node A then calls the function *encode*() to prepend the unit vector of length m to each vector. Once the intermediate node B receives μ incoming blocks, it first starts *vry_NC*() to check the linear independence of the received blocks and discards the blocks that don't pass the check; and then node B runs *com_NC*() to do a linear combination of the valid blocks to a new one. Figure 1 shows that the computational overhead on the intermediate node is caused by *vry_NC*() and *com_NC*().

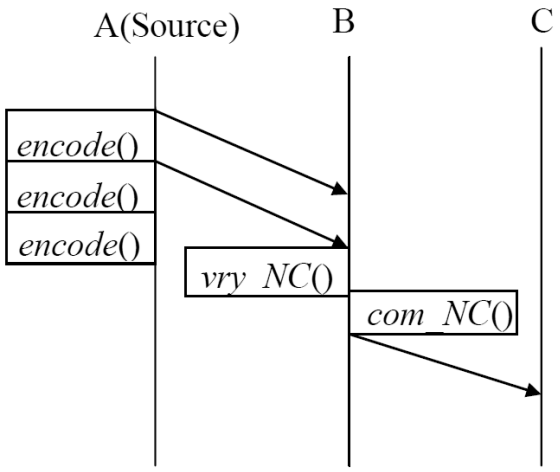

Figure 1: Network Coding

5.2 Homomorphic Signature

In homomorphic network coding signature, to transmit a file, the source node A starts *encode*() on the original file to generate coded blocks, and then calls *vry_Sig*() to sign those blocks; after that, the node A sends out the coded blocks along with the signatures through its outgoing edges. When the intermediate node

B receives μ blocks, it first checks the linear independence of the received blocks by applying *vry_NC*() and then continues to validate the signatures by running *vry_Sig*() function; finally it combines the blocks that pass the verification check; that is, it calls *com_NC*() to combine the information part of the blocks and *com_Sig*() to combine the signatures. Notice that, in addition to the operations done in network coding, the verification algorithm in homomorphic signature needs to check the validity of the signature and the combining algorithm needs to combine the signatures associated with the valid blocks. Here, we refer to two operations *com_NC*() and *com_Sig*() to *combine*(). The processing delay in homomorphic network coding is caused by *vry_NC*(), *vry_Sig*() and *combine*().

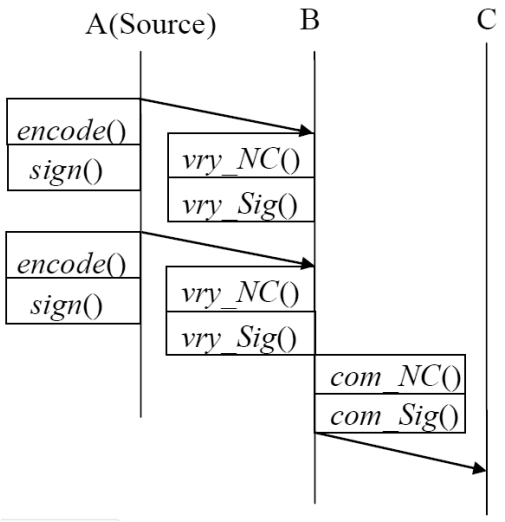

Figure 2: Homomorphic Signatures

5.3 Summary and Comparison

From the above illustrations of the communicational models, we can see that both network coding and homomorphic signature incur the computational overhead in transit. Though, on the intermediate nodes, network coding only performs two operations, *vry_NC*() and *com_NC*(), it faces the challenge of pollution propagation. Homomorphic network coding signatures are designed to deal with the drawbacks of network coding; nonetheless, it results in expensive cryptographic overhead because of four extra operations, *Sign*(), *vry_NC*(), *vry_Sig*() and *com_Sig*(). The latter three operations will lead to computational overhead while the first one will not result in overhead since the signing operation is only carried out on the source node.

In later chapters, we will see that in network coding, the processing time of *vry_NC*() and *com_NC*() are negligible when compared to the overhead in homomorphic signature schemes since these two computations usually require multiplications and additions that are not expensive. However, in homomorphic signatures, two extra operations done on the intermediate nodes, *vry_Sig*() and *combine*(), require complex cryptographic computations(multiplication, exponentiation and modular) which will lead to heavy computational overhead and then affect the performance of a signature scheme.

To conclude, we can learn that homomorphic signature will cause expensive computational overhead when compared to the network coding while the security level it obtains is higher than that of network coding. And to achieve the purpose of the project, that is, to compare the computational overhead of the schemes in random oracle/standard model, our focus is on the three extra operations(*vry_NC*(), *vry_Sig*() and *combine*()) done in homomorphic signature to see how large the time difference of each operation of the schemes in the same group is.

6 Implementations

To compare two groups(*RSAGroup* and *BilinearGroup*) mentioned above separately from an experimental perspective, we implement four signature schemes on Ubuntu 11.10 which is installed in workstation 8.0 virtual machine on an Intel Core i5 processor(2.3GHZ, 4GB). GNU Multiple Precision Arithmetic Library(GMP) [20] is used to deal with the large integers and OpenSSL [23] is for implementing the random oracle. For the implementation of pairing operation, we choose the Pairing-Based Cryptography(PBC) [21] library which is built on the GMP library.

GMP library is one of the popularly-used big number libraries which provides arbitrary precision arithmetic. This library is designed to be as fast as possible and said to be the fastest of all big number libraries so far. The high speed has made it a good choice for cryptographic applications, especially network coding schemes in which efficiency is one of the main requirements. To implement pairing, routines like elliptic curve arithmetic, elliptic curve generation and pairing computation are provided in PBC library and by using the functions available in GMP library, PBC library is able to achieve reasonable pairing time. An additional advantage of PBC library is when implementing pairing operations, we don't need to know the precise knowledge of elliptic curves since the API is abstract enough for users with an elementary background of pairings. OpenSSL is an open source toolkit to implement SSL/TCL security protocol and various hash functions are available as well; SHA2 in OpenSSL is applied to instantiate the random oracle.

In four schemes, the upper bound of coordinates of the initial vectors M is set to $(2^t - 1)$ meaning that the length of the specified coordinates is no more than $t$bits. The number of blocks a file is divided into is set to m and each block consists of n coordinates before prepended with the unit vectors. Accordingly, the generation size is at most $(2^t - 1) * m * n$. In general, the values of M and m are fixed in advance by the network coding applications; therefore, to transmit a file, we can play with the value of n to let M and m meet the conditions set by the applications.

This chapter is going to introduce the implementation details of four schemes; first, we introduce the Gaussian Elimination and then two RSA-based schemes followed by two bilinear-based schemes.

6.1 Gaussian Elimination

In linear algebra, Gaussian elimination is a standard algorithm for solving simultaneous linear equations; it can also be used to compute the determinant of a matrix or to find the rank of a matrix. Two processes are contained in Gaussian

elimination, 'Forward Elimination' and 'Back Substitution'. Forward Elimination is aimed at transforming a matrix into the form of triangular or echelon or a degenerate equation by applying elementary row operations; a degenerate equation implies that there is no unique solution for the set of functions; whereas the triangular or echelon forms indicate that there is a unique solution. Back Substitution is used to solve the resulting equations from Forward Elimination to obtain the solutions. Another popular application of Gaussian elimination is to check the linear independence of a family of vectors. Linear independence of a set of vectors is, in linear algebra, that none of the vectors can be expressed as a combination of other vectors in the same set. As mentioned, in network coding, the intermediate nodes have to check the linear independence of received blocks and discard the blocks that depend on the others by calling $vry\_NC()$. In our experiment, we use the algorithm Gaussian elimination to verify linear independence.

The implementation of $vry\_NC()$ can be shown by using the following pseudo code. The main idea is to apply the first process of Gaussian elimination, Forward Elimination, to reduce the matrix formed by the received blocks(each row represents one block) and then check whether there exists a row with every components set to 0; if there doesn't exist a row with all coordinates set to 0, the vectors are linearly independent and the intermediate nodes would store these blocks for further modification.

```
if(row>column){
    Linear Dependence.
    Return.
}else {
    for(i = 0;i < column;i++){
        /* Find the row with the largest first number */
        FindMaxRow();
        /* Swap the maxrow and ith row */
        Swap();
        /* Search row with all coordinates set to 0 */
        if(Exit one row with all zeros){
            Linear Dependence.
        }else{
            Linear Independence.
        }
    }
}
```

The above code shows the implementation of *vry_NC*(). The idea is the same in the four schemes we examine in our project; however, according to the algebraic setting the schemes defined over, the schemes in the same group(*RSAGroup* or *BilinearGroup*) have the identical implementations of *vry_NC*(). This implies that the *vry_NC*() algorithms in the same group will not cause the time difference of the schemes. Consequently, to compare the schemes in the same group in terms of computational overhead, we can safely leave the comparison of *vry_NC*() algorithms and concentrate on *vry_Sig*() and *combine*().

6.2 Implementations of *RRSA* and *SRSA*

This section is going to give the implementation details of two RSA-based schemes in *RSAGroup*. Given the identical implementations of *vry_NC*() in two schemes, the focus is on *vry_Sig*() and *combine*()

6.2.1 Implementation Comparisons

Two tables, Table 1 and Table 2, illustrate the scope of parameters concerned with two RSA schemes in random oracle and standard models respectively. In both schemes, the value of t is set to 192 which means that the magnitude of the components in the initial vectors is no more than $(2^{192} - 1)$ and the size of the file identifier fid is 193bits. In *RRSA* scheme, the distance L is 20 hops and the coefficients are randomly chosen from $\mathbb{Z}_Q$(Q is set to 257 in our implementation); the public exponent e is larger than $m(mQ)^L M$ as specified in the scheme definition. In *SRSA* model, the coefficients are from the field $\mathbb{F}_e$ and e is 193bits which is equal to fid and defined by one of the security parameters λ. Notice that the public exponent e in *RRSA* will change with m and is larger than e in *SRSA*. Nonetheless, this is not going to affect the time difference of computational overhead in two schemes since public exponent e is only used on the source node for signing algorithms.

Table 1: Parameters of *RRSA*

Parameters	M	q	fid	L	e	α_i
Range	192bits	$= 257$	$< 2^{193}$	$= 20$	$> m(mQ)^L M$	$\in \mathbb{Z}_Q$

From these two tables, we notice that two RSA-based schemes have the same size of M and fid; and other parameters are chosen according to the definitions of the schemes. In our experiments, we will change the value of m and n respectively to see the variation of the processing time for *vry_Sig*() and *combine*().

Table 2: Parameters of *SRSA*

Parameters	M	e	fid	α_i
Range	192bits	193bits	193bits	$\in \mathbb{F}_e$

Our project will also examine how the security parameter k affects the cryptographic overhead of two schemes. The value of k is set to 512 and 1024 and Table 3 shows the other parameters which depend on the value of k. Notice that g_i's/$h_i's$ are generated on the source node and send out with the information in standard model; whereas in the random oracle model, only g_i's are generated at first, m h_i's are constructed on the fly on the intermediate nodes using random oracle.

Table 3: Parameters of Two RSA Scheme

Parameters	k	N	p/q	g_i/h_i
size(bits)	1024	1024	512	1024
size(bits)	512	512	256	512

6.2.2 Parameter Generations

It is worth mentioning how we choose the parameters g_i's and h_i's in *RRSA* and *SRSA*. Before that, we first illustrate the definition of quadratic residue and then examine two theorems which serve as theory evidences.

Quadratic Residue. An integer $a \in \{0, ..., N-1\}$ is in QR_N if and only if there exists an integer x such that $x^2 \equiv a(mod\ N)$.[19]

Theorem 1. If $\mathbb{G}$ is a group of prime order p, then $\mathbb{G}$ is cyclic and all elements in $\mathbb{G}$ except the identity are generators of $\mathbb{G}$.[19]

Theorem 2. N is a composite number with two prime multipliers p and q, and $y = (y_p, y_q)(y$ is the least common multiple of y_p and $y_q)$. If y_p is a quadratic residue modulo p and y_q is a quadratic residue modulo q, then y is a quadratic residue modulo N.[19]

***RRSA* scheme**. In *RRSA* construction, g_i's are generators of the subgroup of quadratic residue QR_N. According to the two theorems above, we generate g_i's as follows:

1. Randomly choose r_1 from $\mathbb{Z}_p$ and compute $r_1 = r_1^2(mod\ p)$; (Theorem 1)

2. Randomly choose r_2 from $\mathbb{Z}_q$ and compute $r_2 = r_2^2 (mod\ q)$; (Theorem 1)

3. Compute $g_i = (r_1, r_2)$. (Theorem 2)

h_i's are in the domain QR_N which need to be calculated on the intermediate nodes. Since the output of a hash value is a random number, we need to do extra calculation to map the random outputs into QR_N. We generate h_i's as follows:

1. Compute $r = \text{SHA2}(fid + i)$;

2. Compute $h_i = r^2 (mod\ N)$ to map the hash value into QR_N.

***SRSA* scheme**. In *SRSA* construction, g_i's and h_i's are random numbers in $\mathbb{Z}_N^*$ and they are selected according to the following steps:

1. Randomly choose r_1 from $\mathbb{Z}_p$;

2. Randomly choose r_2 from $\mathbb{Z}_q$;

3. Compute $g_i/h_i = (r_1, r_2)$ (the least common multiple of r_1 and r_2).

6.3 Implementations of *RCDH* and *SSDH*

In this section, we are going to discuss the implementation details of two bilinear-based schemes.

6.3.1 Implementation Comparisons

Two bilinear-based signature schemes apply asymmetric pairing available in PBC library. There are seven kinds of pairings in PBC library, from Type A to Type G; among which, Type B and Type C pairings haven't been implemented. Type A results in the fastest pairing but is only suitable for symmetric pairing. In order to do efficient asymmetric pairing, we feed Type D parameters into the pairing algorithm. Type D pairing is constructed on curves of degree 6 and has the order of a prime number or a prime multiplied by a small constant; complex multiplication method is applied to generate Type D curves. In Type D pairing, the elements in group $\mathbb{G}$ are expressed in a small size, normally larger than 170bits; and by using a trick, the size of elements in group $\mathbb{G}'$ can be only three times longer(generally the size of $\mathbb{G}'$ should be six times longer without the trick used in PBC library). The other types of pairing are so costly that they are not recommended in the time-critical applications.

There are a few Type D parameters provided and they are represented as triples, the discriminant, the size of the elements in $\mathbb{G}$ and the size of group orders. In our experiment, we use the parameter d277699-175-167 which indicates that the prime orders of three bilinear groups are of size 167bits and the size of the elements in $\mathbb{G}$ is 175bits. Therefore, we set the size of p(as defined in the schemes) to be 167bits as well as the security parameter k; p and k are of the same size but should conform to the condition $p > 2^k$. Then t can be up to 167 as the coordinates of vectors should be in $\mathbb{F}_p$. As the type of the pairing indicates, the element size of group $\mathbb{G}$ is accordingly set to 175bits and the element size of the second group $\mathbb{G}'$ is tripled, 525bits. The scheme in random oracle needs to calculate m h_i's on the intermediate nodes. As shown in Table 4, the file identifier fid in random oracle is exactly of $k$bits; however, fid can be chosen from the field $\mathbb{Z}_p^*$; for simplicity, we assume that the size of fid is exactly $k$bits in both schemes.

Table 4: Parameters of Bilinear Groups

Parameters	k	p	G_1	G_2	fid
RCDH	167bits	167bits	175bits	525bits	167bits
SSDH	167bits	167bits	175bits	525bits	$\in \mathbb{Z}_p^*$

We notice that both bilinear-based schemes apply the same size of M and are using the same type of pairing with the same parameter set. The same as the comparisons in *RSAGroup*, we will check how the computational overhead varies with m and n respectively. However, we are not going to examine how the security parameter k affects the processing overhead of two schemes. As known, pairing is a time-consuming operation and even in a highly optimized curve like BN curve[3], the time needed for one pairing operation is also 20 times longer than that for exponentiation. Therefore, to apply a pairing-based network coding scheme, a pairing with efficient performance is required. In PBC library, we can see from the above analysis that Type D is the most attractive one for our implementation. Typically, in a Type D pairing, the size of the element in $\mathbb{G}$ is larger than 170bits; so we choose the set of parameters d277699-175-167 that just meets this condition to achieve a certain security level. d277699-175-167 is the parameter set already generated and stored in the PBC file directory.

6.3.2 Parameter Generations and Library Functions

Here, we give the instantiation of random oracle and discuss some programming functions in PBC library.

Hash Value Generation. We have mentioned that, in *RCDH* construction, m h_i's are established on the fly. These values, as defined, are in group $\mathbb{G}$ and this has raised the problem of how to map the hash values into the group $\mathbb{G}$. In our experiment, we use the hash function SHA-256 in OpenSSL to generate the hash value; the process is shown as follows:

1. Randomly choose g from $\mathbb{G}$;
2. Compute r=SHA256($fid + i$);
3. Compute $h_i = g^r$.

Notice that the generation of m hash values are done over the field $\mathbb{Z}_p$ and all these results are then treated as the exponents in the exponentiations with the basis picked up from $\mathbb{G}$.

Pairing. There are steps to follow when running PBC library to complete one pairing operation. Several pairing functions are available that may achieve the same goal, but we only focus on the one we use in our project. To do pairing computation, we have to initialize the pairing variable at first; and then we call the pairing function

$$element\_pairing(element\_t\ out,\ element\_t\ in_1,\ element\_t\ in_2).$$

This function takes two inputs of elements in_1 from $\mathbb{G}$ and in_2 from $\mathbb{G}'$ and outputs out in $\mathbb{G}_T$. Based on the platform we implement, one pairing takes approximately 5.43ms.

Exponentiation. Another function we have to mention is the one to implement exponentiation

$$element\_pow\_zn(element\_t\ x,\ element\_t\ a,\ element\_t\ n).$$

It computes $x = a^n$ where a can be an element in $\mathbb{G}$ or $\mathbb{G}'$ and n is a value chosen from a ring or a field. We will see later that the function costs approximately the same amount of time as the pairing operation when the size of the element in the group where a chosen from becomes large. In our experiment, with the base numbers from group $\mathbb{G}'$, one exponentiation takes nearly 6.23ms; while, with the base numbers from group $\mathbb{G}$, the amount of time is only 0.77ms which is almost seven times less. Notice that the time for exponentiation with the base numbers in $\mathbb{G}'$ is comparable to the time for pairing.

Multiplication. Compared to exponentiation, multiplication in groups $\mathbb{G}$ or $\mathbb{G}'$ is not costly

element_mul(*element_t* n, *element_t* a, *element_t* b).

It computes $x = ab$ where a and b can be chosen from the bilinear groups $\mathbb{G}$ and $\mathbb{G}'$, rings or fields. When one scheme needs to do one more multiplication(in groups $\mathbb{G}$ or $\mathbb{G}'$) and one more exponentiation(in groups $\mathbb{G}$ or $\mathbb{G}'$) than the other scheme, the time difference of these two schemes is mainly caused by the exponentiation rather than the multiplication; this is because the processing time of multiplication is negligible when compared to the time needed for exponentiation.

The discussion of three programming functions in PBC library will support for our analyses of the experimental results in Chapter 7.

7 Results and Analyses

In what follows we will compare the schemes in two groups(*RSAGroup* and *BilinearGroup*) mentioned above separately from an experimental perspective to see whether homomorphic signatures in random oracle/standard models are comparable or not. The comparisons are associated with precise analyses and followed by a brief summary of this chapter.

Four schemes we describe above apply the method Gaussian Elimination in *vry_NC*() to validate the linear independence of received blocks. The implementations of Gaussian Elimination have been explained in Section 6.1; the experimental results demonstrate that the time needed for the *vry_NC*() algorithm depends on the number of downloaded blocks and the overhead can be neglected when compared to other algorithms(*vry_Sig*() and *combine*()). We have found that the *vry_NC*() implementations of the schemes in *RSAGroup* or *BilinearGroup* are identical; as a result, the processing time for *vry_NC*() can be safely ignored when comparing schemes in the same group.

Therefore, our comparison will only focus on two algorithms, *vry_Sig*() and *combine*(). We assess the *vry_Sig*() algorithm at the targets by measuring the time for verifying m signatures. For simplicity, the download delay at the targets is not taken into consideration. The operation *combine*() is analysed by calculating the time needed for combining m blocks on the source node.

Notice that we assume the number of downloaded blocks is equal to the number of blocks a file is divided into; that is, the number of downloaded blocks is fixed to m. The variable n is the dimension of the information vectors. In this chapter, m denotes the number of downloaded blocks on the intermediate nodes.

7.1 Comparisons of *RRSA* and *SRSA*

We will set out to compare two RSA-based schemes in *RSAGroup*; our analysis concentrates on exploring how distinct the computational overhead of two schemes by observing how the values of m, n and k affect the processing time for *vry_Sig*() and *combine*(). The implementation details have been illustrated in Section 6.2.

7.1.1 Verification Algorithm

This section is going to analyse *vry_Sig*() in *RRSA* and *SRSA*. The *vry_Sig*() algorithms in two schemes consist of the same computational operations, one exponentiation and one multi-exponentiation. The discussion is based on $k = 512$.

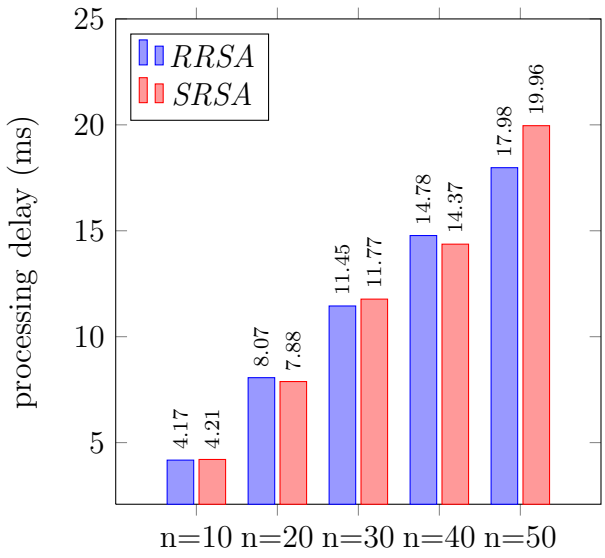

Figure 3: Verification Overhead($m = 8$)

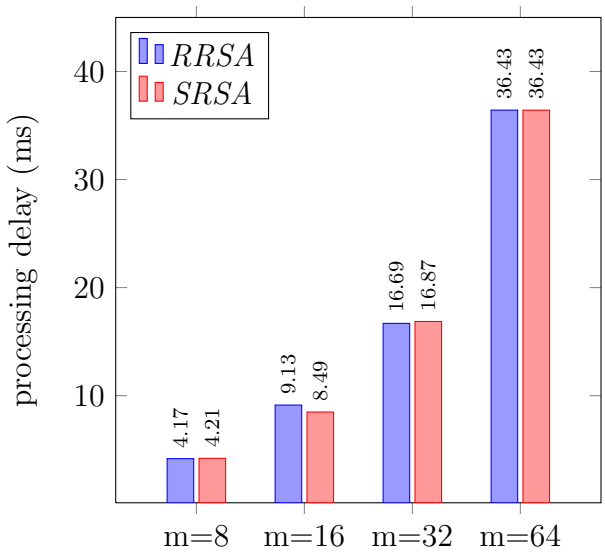

Figure 4: Verification Overhead($n = 10$)

Figure 3 and Figure 4 illustrate the processing delay of verification algorithms in two schemes when k is 512. The time required for this operation depends on the values m and n; we will examine the variation according to these two values respectively. Figure 3 shows how n affects the $vry\_Sig()$ time when m is

constant($m = 8$ in our case; n is changed from 10 to 50 with an increase of 10 every time); whereas in Figure 4, n is constant but m changes ($n = 10$ in this case; m is doubled every time).

These two figures demonstrate a clear feature that the *vry_Sig*() algorithms in these two schemes take almost the same amount of time when m and n have the same values. Specifically, when m and n are set to 8 and 10 respectively, the verification time for the scheme in random oracle is 4.17ms in contrast to 4.21ms in standard model; the time spent is approximately the same. Besides, we also notice a steady rise in the time needed for *vry_Sig*() in *RRSA* or *SRSA* when the variable in X-axis(m in Figure 3 or n in Figure 4) grows; the feature worth mentioning is that the time doubles as the number of downloaded blocks m doubles(see Figure 4). The variation tendency shows that the overhead of the *vry_Sig*() algorithms depends on the values of m and n.

Analysis. Let us analyse the *vry_Sig*() algorithms in Section 4.1 and Section 4.2; they are mainly different in the following two aspects. First, in *RRSA*, *vry_Sig*() needs to generate m h_i; whereas, in *SRSA*, these values are set and transferred with the file blocks. Second, in comparison to Equation (1), Equation (2) needs to do two extra operations, one is the exponentiation with the exponent which can be up to 193bits in our case and the other one is the multiplication in $\mathbb{Z}_N$. From the experimental results, the excess computations done in *RRSA* and *SRSA* require approximately the same amount of time and these computations are cheaper when compared to the multi-exponentiation operation.

7.1.2 Combining Algorithm

This section gives the comparisons of *combine*() algorithms in two schemes; as the same as the *vry_Sig*() comparison, the analysis is based on $k = 512$.

Results of the implementation illustrate that, in the random oracle model, the time for *combine*() is very cheap and far less than the *vry_Sig*() processing delay. The value of m is the only element that affects the *combine*() time in *RRSA* and the time shows a very small increase when m grows. From Figure 5, *combine*() takes around 0.66ms on average and the amount of time stays stable when the value of n changes from 10 to 50 and the value m is fixed to 8; from Figure 6, *combine*() costs 0.61ms when *vry_Sig*() takes 4.17ms(in the case $m = 8$, $n = 10$), and the combine time has only a slight rise to 0.92ms($m = 64$, $n = 10$) from 0.61ms when *vry_Sig()* takes 19.98ms. Nevertheless, this is not the case when it comes to the scheme in standard model where *combine*() has a noticeable increase when m or n changes. We notice that the combine time and the increase

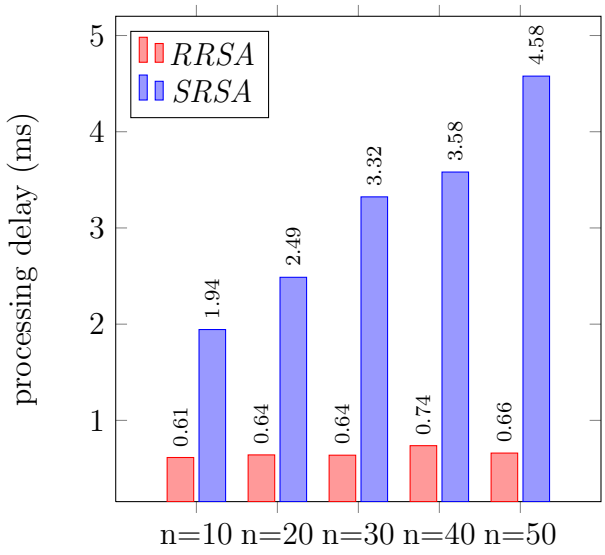

Figure 5: Combine Overhead(m=8)

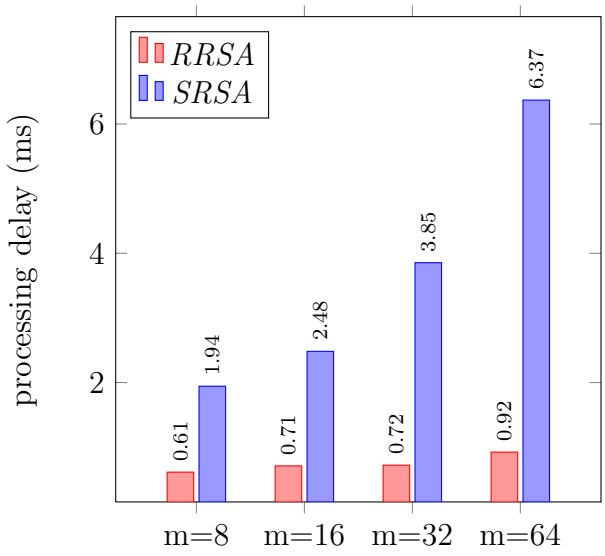

Figure 6: Combine Overhead(n=10)

in combine time in standard model are more significant than that in random oracle. For example, *combine*() in *SRSA* scheme takes 1.94ms($m = 8$ and $n = 10$) and the time increases to 4.58ms($m = 8$ and $n = 50$) and to 6.37ms($m = 64$ and $n = 10$); in standard scheme, *combine*() takes 6.37ms while *vry_Sig*() takes 19.96ms when $m = 64$ and $n = 10$. The difference becomes more obvious when k is set to 1024 which will be discussed in detail later. In addition, the amount of

time in standard model increases as more blocks are downloaded(shown in Figure 6) and is elapsed as the dimension n grows (shown in Figure 5).

Analysis. Notice that *vry_Sig*() brings no difference when comparing the computational overhead of two RSA-based schemes in random oracle/standard models. In the random oracle model, one *combine*() operation contains multiplications with 8-bit coefficients and exponentiations with 8-bit exponents; small coefficients make the computations less expensive. However, in standard model, the coefficients chosen can be up to 193 bits in our implementation which are more than 30 times than that in random oracle; and the *combine*() algorithm needs more cryptographic computations. We find that among those extra computations in *combine*() algorithm, the multi-exponentiation in Equation (3) in Section 4.2 is the most costly one; and it is the main cause of the time difference of the two *combine*() algorithms.

7.1.3 Security Parameter k

The above discussion is focused on $k = 512$. In our implementation, we change the value to 1024 and observe the same variation tendency discussed in Section 7.1.1 and Section 7.1.2. We also notice that the time differences are magnified when k becomes larger. We will examine how the security parameter k affects the processing time for *vry_Sig*() and *combine*() in respective scheme.

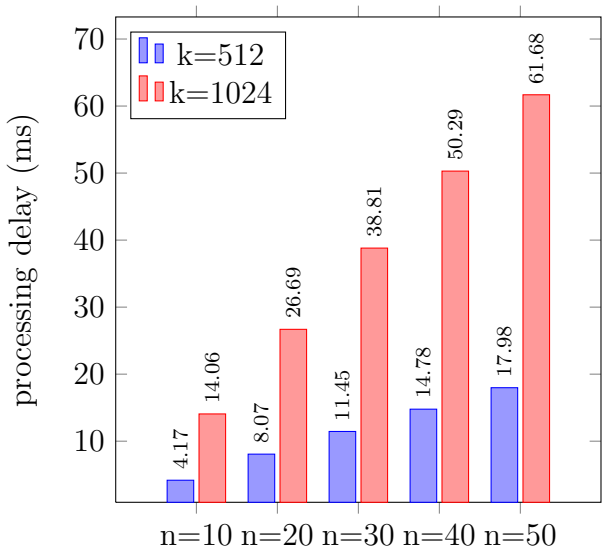

Figure 7: Verification Overhead($m = 8$) in *RRSA*

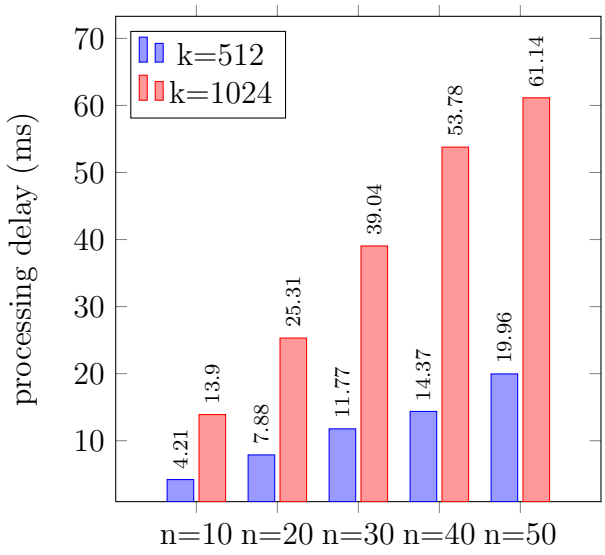

Figure 8: Verification Overhead($m = 8$) in *SRSA*

Figure 7 and Figure 8 demonstrate the processing delay of the algorithm *vry_Sig*() in random oracle and standard models according to two distinct k. From the figures, we can see that the verification time is at least tripled under the same m and n when k is doubled. The combine algorithm in *RRSA* scheme depends on the value of m and is very time-saving even when k is raised to 1024(an increase from 0.68ms($m = 8$, $n = 10$) to 1.22ms($m = 64$, $n = 10$)); whereas, in standard model, the combine operation becomes significantly expensive. Figure 9 and Figure 10 illustrate how the combine time varies with the values of m and n respectively when k is raised to 1024bits. The time for *combine*() increases from 4.85ms($m = 8$ and $n = 10$) to 12.80ms($m = 8$ and $n = 50$) and to 18.46ms($m = 64$ and $n = 10$). The same as the *vry_Sig*() algorithm, the combine time is more than tripled to that when $k = 512$ under the same m and n. In a nutshell, as the value k increases, the processing time needed for *vry_Sig*() and *combine*() gets larger as well as the time difference of two RSA-based schemes.

Analysis. We can see from Table 3 in Section 6.2 that when k is doubled, the size of the parameters N, g_i's and h_i's are doubled; for simplicity, the size of e stays the same when k changes. The increase in the parameters' size is the reason why the time for *vry_Sig*() in *RRSA* and *SRSA* or *combine*() in *SRSA* is twice more when k is raised to 1024 from 512. These parameters are important when doing calculations in *vry_Sig*() and *combine*(). The reason why the combine time in *RRSA* has only a slight increase is that there is only one operation (*mode* N) which will cause the excess processing time in the *combine*()

algorithm and this operation is in itself not expensive.

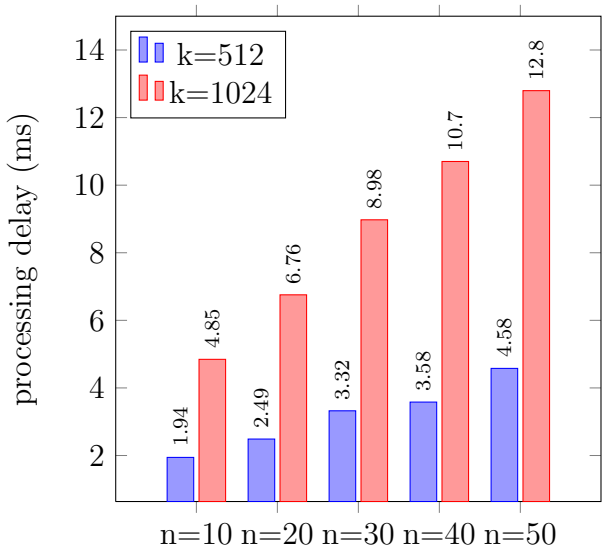

Figure 9: Combine Overhead(m=8) in *SRSA*

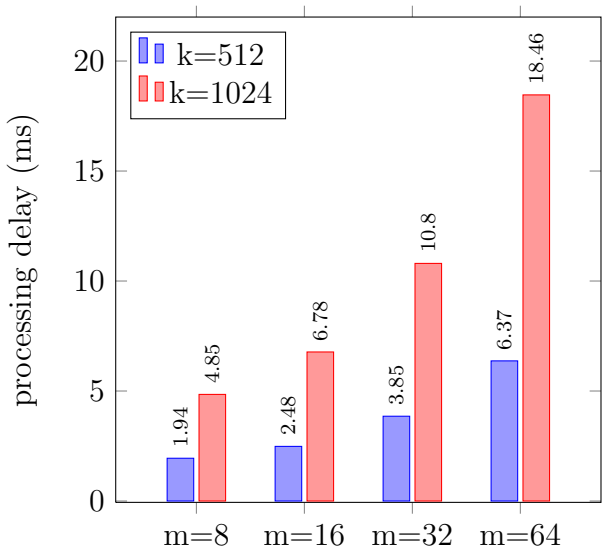

Figure 10: Combine Overhead(n=10) in *SRSA*

7.1.4 Summary and Comparison

From what we discussed above, it is noticeable that the overhead differences of *RRSA* and *SRSA* are primarily caused by the algorithm *combine*() since two

schemes take approximately the same amount of time to complete one *vry_NC*() algorithm and one *vry_Sig*() algorithm. Figure 11 is a combination of the results shown above when $k = 512$; it illustrates how the time for *vry_Sig*() and *combine*() changes according to the values of m and n. Clearly, the *RRSA*Verify line and the *SRSA*Verify line are overlapped meaning that *vry_Sig*() in two schemes takes generally the same amount of time. The gap between *RRSA*Combine line and *SRSA*Combine line indicates the time difference of two combine algorithms in random oracle/standard models; we can see that the gap becomes larger when m or n grows and will be more significant when a higher security level(k is set to a larger value) is required.

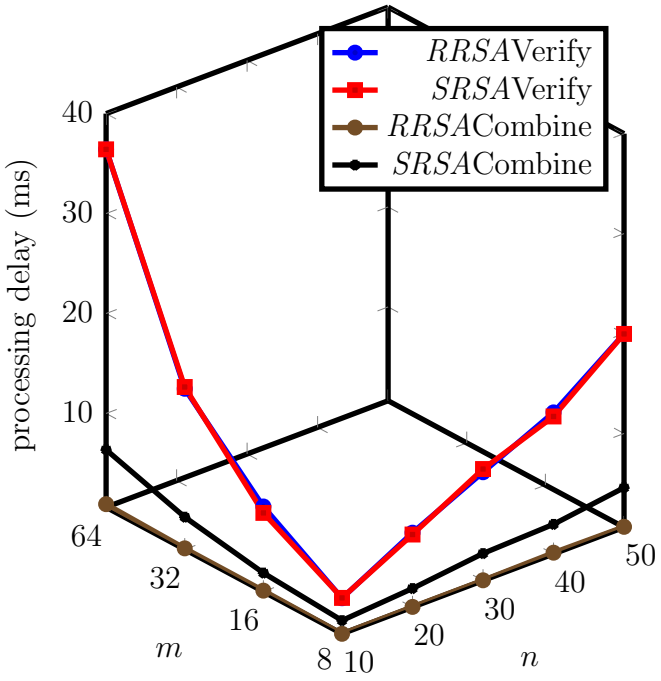

Figure 11: Processing Overhead($k = 512$)

To conclude, two RSA-based schemes are somewhat comparable in terms of processing delay when the values of m, n and k are small; however, computational overhead becomes markedly expensive when larger k, m and n are used. The reason is because the time difference of computational overhead in *RRSA* and *SRSA* is small with small k, m and n and gets larger when larger k, m and n are applied. To achieve a more secure scheme by using a larger k, efficiency is to be traded; this trade-off should be assessed according to the specific applications where homomorphic schemes are introduced.

7.2 Comparisons of *RCDH* and *SSDH*

This section is going to make a comparison of *RCDH* and *SSDH* in terms of *vry_Sig*() and *combine*(); the comparison is based on examining how the time difference of two bilinear-based schemes changes with m and n respectively. The implementation details have been shown in Section 6.3.

7.2.1 Verification Algorithm

Here, we are going to make a comparison of *vry_Sig*() algorithms of these two schemes over bilinear groups.

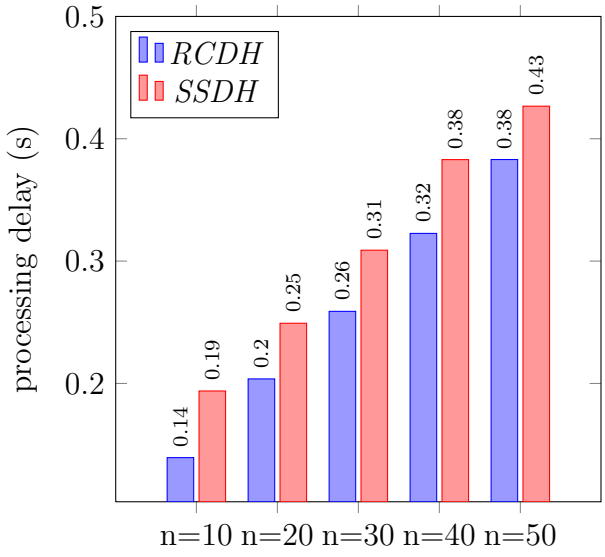

Figure 12: Verification Overhead($m = 8$)

Figure 12 and Figure 13 demonstrate how the *vry_Sig*() time changes with m and n respectively. According to Figure 12, we see that *vry_Sig*() algorithm in standard model always takes around 52ms longer than that in random oracle model whatever the value of n is; for instance, when $n = 50$, the time required in standard model is 0.43s compared to 0.38s in random oracle which is around 50ms less and the same phenomenon is observed when n is 10, 20, 30 and 40. In Figure 13, we see the time difference of two *vry_Sig*() algorithms doubles as m doubles. Clearly, when $m = 8$, the time difference(the *vry_Sig*() time in *SSDH* subtracts to the *vry_Sig*() time in *RCDH*) is 50ms which is doubled to 100ms when $m = 16$ and then to 170ms($m = 32$) and 440ms($m = 64$). Additionally, the time needed for *vry_Sig*() in both *RCDH* and *SSDH* shows a steady increase when the value of m or n increases. We find that the time doubles when the

number of downloaded blocks m doubles; in Figure 13, *vry_Sig*() takes 1.14s to verify 64 blocks in random oracle and this number is nearly twice that of the value 0.59s which is the time for verifying 32 blocks.

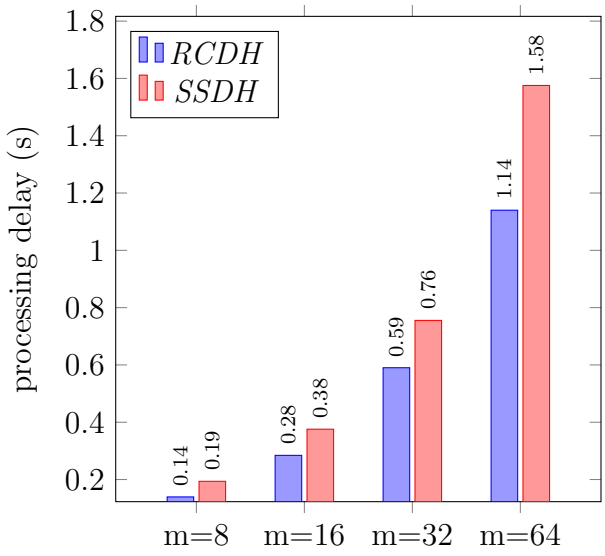

Figure 13: Verification Overhead($n = 10$)

Analysis. We now analyse the operation differences between two verification algorithms. Both verification algorithms consist of one multi-exponentiation and two pairings. In *RCDH*, *vry_Sig*() needs to compute m h_i on the intermediate nodes; whereas, in *SSDH*, the initialisation of m h_i is done on the source node but two multiplications (in $\mathbb{G}$ and $\mathbb{G}'$ respectively) and two exponentiations (in $\mathbb{G}$ and $\mathbb{G}'$ respectively) are required to complete the verification procedure(this is shown from Equation (4) in Section 4.3 and Equation (5) in Section 4.4).

In Figure 14 and Figure 15, we refer m hash mappings to *HashMapping* and four mentioned group operations to *GroupAlgo*. From Figure 14, we learn that *GroupAlgo* is much more expensive than *HashMapping*. When $m = 8$, the time needed for *HashMapping* is approximately 6.79ms; while, under the same condition, *GroupAlgo* takes about 52.57ms, which is 45.78ms longer. Besides, Figure 14 shows that the amount of time for *GroupAlgo* grows more significantly than that of *HashMapping*. Precisely, the *HashMapping* time has an increase of 41.35ms(from 6.79ms($m = 8$) to 48.14ms(m=64)) in contrast to 418.23ms(from 52.57ms($m = 8$) to 470.80ms($m = 64$)) in *GroupAlgo*. Figure 15 shows the processing difference between *GroupAlgo* and *HashMapping* when m has various values; this figure is constructed based on Figure 14. The time gap doubles when the value of m doubles; the time difference is 45.78ms when $m = 8$ and

increases to 100.75ms when $m = 16$, and then continues to extend to 198.67ms and 422.66ms when $m = 32$ and $m = 64$ respectively. In our experiment, it is noticeable that the block numbers m is the only element that influences the time difference between *HashMapping* and *GroupAlgo*.

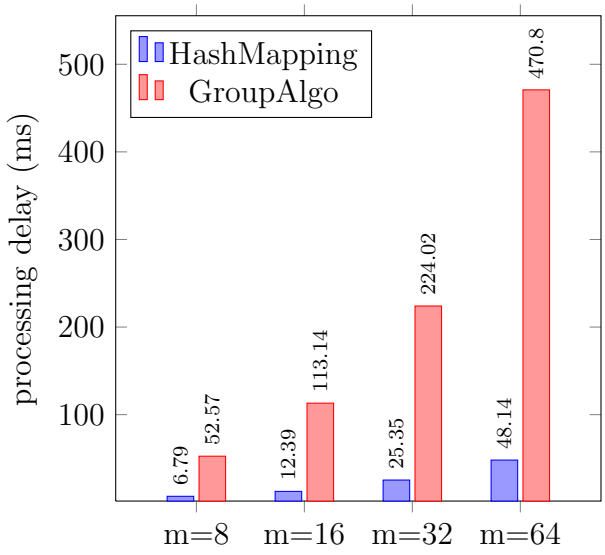

Figure 14: Processing Overhead(n=10)

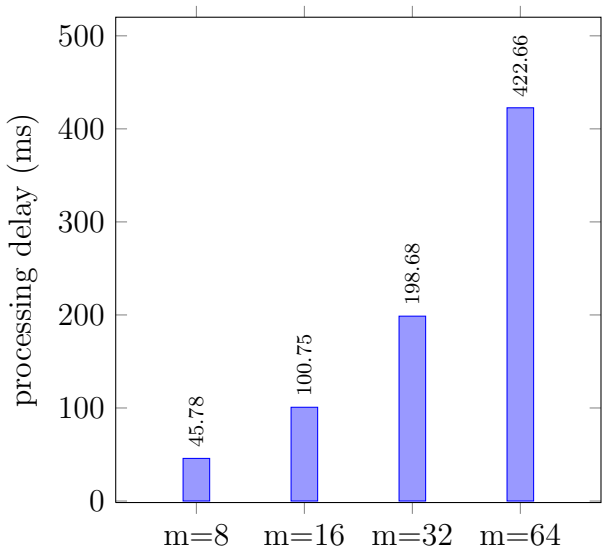

Figure 15: Time Difference(n=10) of HashMapping and GroupAlgo

Here, we check which operation in *GroupAlgo* contributes to the time difference. The focus is on two exponentiations since, relatively speaking, two multiplications are very cheap. We have mentioned that the size of the elements in $\mathbb{G}'$ is two times longer than that in $\mathbb{G}'$(175bits in $\mathbb{G}$ and 525bits in $\mathbb{G}'$). As discussed in Section 6.3.2, to do an exponentiation using the function *element_pow_zn*() will take a longer time if the base is set to the element in $\mathbb{G}'$; from the experiment, one exponentiation with the base numbers in $\mathbb{G}'$ costs 6.23ms on average in contrast to 0.77ms with base in $\mathbb{G}$. In our project, the measurement of *vry_Sig*() is to verify m blocks and one *vry_Sig*() in *SSDH* has to do one extra exponentiation in $\mathbb{G}'$; if we set m to 8, it turns out that 8 exponentiations will spend 49.84ms which occupies around 94.81% in 52.57ms, the time to complete 8 *vry_Sig*() in *SSDH*.

Clearly, this computational difference of *GroupAlgo* and *HashMapping* has led to the different processing delay of the *vry_Sig*() algorithms in two bilinear-based schemes. Figure 15 which illustrates the time difference of *HashMapping* and *GroupAlgo* can be used to demonstrate the overhead difference of two verification algorithms.

7.2.2 Combining Algorithm

In what follows we consider the *combine*() overhead in these two bilinear-based schemes.

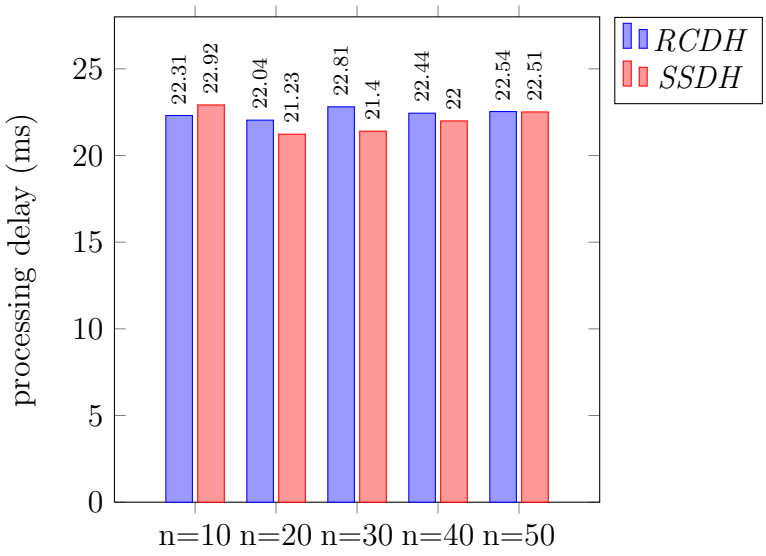

Figure 16: Combine Overhead(m=8)

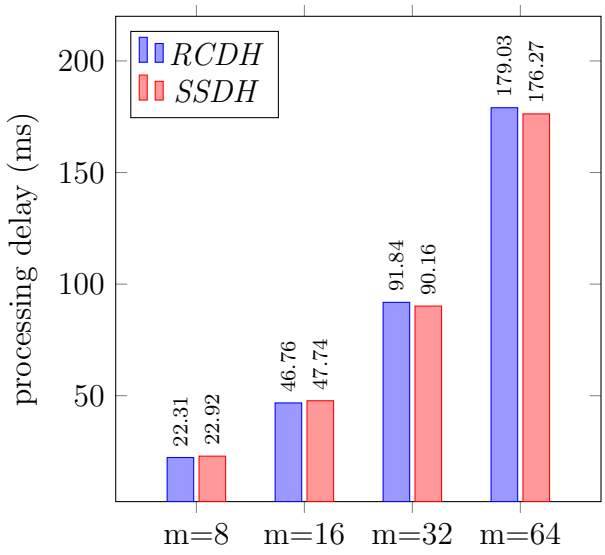

Figure 17: Combine Overhead(n=10)

Figure 16 shows that combine time doesn't depend on the value of n because *combine*() takes about 22ms($m = 8$) whatever the value of n is. In Figure 17, we notice that, in both schemes, the overhead doubles as the number of downloaded blocks m doubles; for instance, in *RCDH*, *combine*() costs 22.31ms when $m = 8$ and the time is doubled to 46.76ms when $m = 16$ and then to 91.84ms and 179.03ms when $m = 32$ and $m = 64$ respectively; this phenomenon can also be observed in *SSDH*. A common feature in Figure 16 and Figure 17 is that the *combine*() algorithms have approximately the same amount of processing delay when m and n are set to the same values. When $m = 64$ and $n = 10$, the *RCDH* schemes takes 179.03ms to complete one *combine*() operation; under the same conditions, the amount of time is 176.27ms in *SSDH* which is close to the value 179.03ms.

Analysis. When we examine the *combine*() algorithms in Section 4.3 and Section 4.4, we find that the computations in both constructions depend on the number of downloaded blocks m, while they don't depend on the dimension of information parts of the vectors n. Besides, the signature in *SSDH* contains two parts x and s; the signature in *RCDH* scheme doesn't contain the second part. As a result, in standard model, *combine*() needs to do the combination of s part additionally; and it turns out that these computations in $\mathbb{F}_p$ are so cheap that they can be neglected. Therefore, *combine*() will not bring an overhead difference to these two schemes in different oracle models.

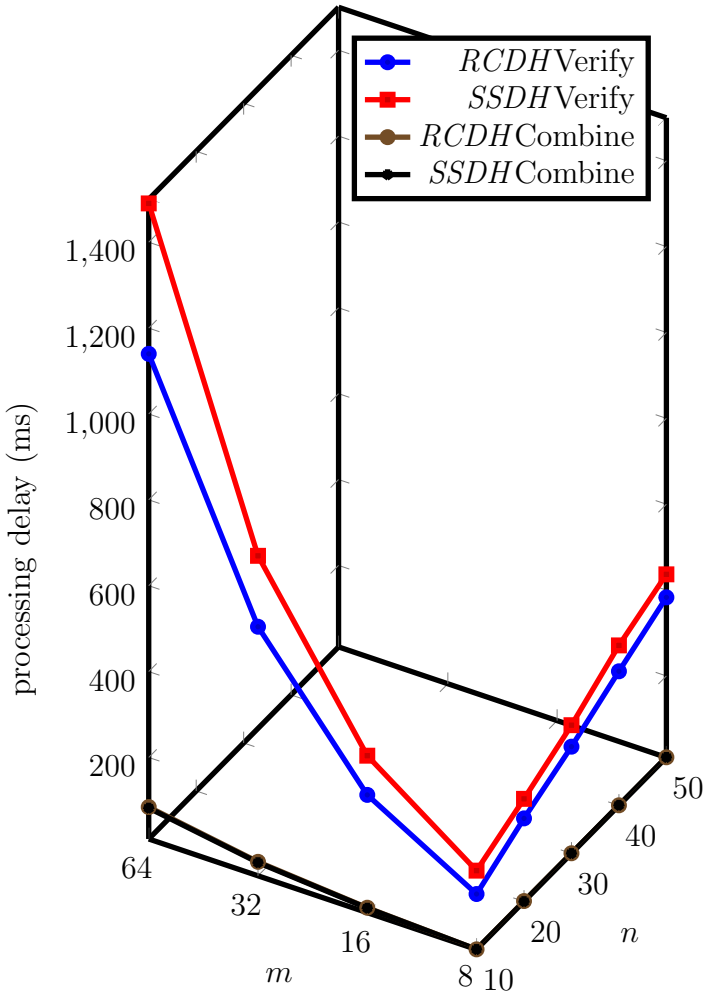

Figure 18: Processing Overhead

7.2.3 Summary and Comparison

From what we discussed above, it is noticeable that the overhead differences of two bilinear-based schemes in random oracle/standard model are primarily caused by the algorithm *vry_Sig*() since two schemes take approximately the same amount of time to complete one *vry_NC*() and one *combine*(). Figure 18 works as a combinational result of the above discussion. From Figure 18, we can see that the *RCDH*Combine line and the *SSDH*Combine line are overlapped meaning that both constructions take approximately the same amount of time to finish one combine operation; while the gap between the *RCDH*Verify line and the *SSDH*Verify line shows the time difference of these two *vry_Sig*() algorithms. Let us see the right part of the *RCDH*Verify and *SSDH*Verify lines which shows

the change when $m = 8$ and n is set to different values; these two parts of lines are parallel and this indicates that the *vry_Sig()* time gap doesn't change with the value of n. The left part illustrates the variation when $n = 10$ and m is set to different values; clearly, the gap between two lines becomes bigger when m increases.

In conclusion, two bilinear-based schemes are somewhat comparable in terms of the communicational overhead when m and n are small; whereas, they become incomparable when the values of m or n gets larger. This is because one of the four extra operations in bilinear groups, the exponentiation in $\mathbb{G}'$, has made the time difference of computational overhead of these two schemes incomparable, especially with larger m and n.

7.3 Summary

As presented, our purpose is to investigate the computational overhead of homomorphic network coding signature schemes in random oracle/standard model to see whether they are comparable. To achieve this goal, we choose four homomorphic schemes and separate them into two groups(RSAGroup and BilinearGroup) and compare the processing delay of the schemes in each group.

From the above analysis, we conclude that the computational costs on the intermediate nodes in different oracle models are somewhat comparable when the generation size and the security parameter k are small and the overhead becomes incomparable when the values of m, n and k become larger. The 'somewhat comparable' is under the condition that a certain amount of time difference between the random schemes and the standard schemes is tolerable. A more detailed explanation will be shown in Section 8.1.

To avoid using a random oracle, we know from the above analysis that the RSA-based scheme in standard oracle would be preferred in contrast to the bilinear-based scheme in standard model. This is because pairing computation is very expensive as well as the exponentiation with the base chosen from the bilinear group of large element size. However, *RRSA* can only be applied in the circumstances that the file is with a decent size and the security parameter k is moderate; the larger the file is, the worse the performance will be. For the question that of what size a file can be qualified as decent and what k is moderate, it depends on the specific homomorphic scheme used and the applications.

8 Conclusions and Evaluations

8.1 Conclusions

Linear network coding, as an alternative to traditional network coding, provides the advantages of high throughput and better network performance. However, linear network coding faces the challenge of pollution attack which may totally mess up the effort of sending files. To combat this attack, many countermeasures have been proposed, including information-theoretic approaches and cryptographic approaches. We focus on the homomorphic network coding signature which is one of the cryptographic approaches.

As mentioned, the difficulty of finding an appropriate hash function has made a theoretical secure cryptographic scheme in random oracle insecure when putting the scheme into practice. The challenge has led to the research of our project, that is, to figure out whether homomorphic signatures in standard model are as efficient as the ones in random oracle in terms of cryptographic overhead on the intermediate nodes; if this is the case, instead of the random oracle signature, the standard signature would be the first choice. To achieve the goal, we choose and compare the schemes over the same algebraic setting in different oracle models. From the existing signatures in the previous works, we select four schemes, the first two(the schemes in random oracle model [13] and standard model [11] respectively) are built on RSA assumption and the other two(the schemes in random oracle model [7] and standard model [11] respectively) are defined over bilinear groups.

We observe that Catalano *et al.* briefly compare the efficiency of these four schemes from two theoretical aspects in [11]. One is the comparison of the communicational overhead of the schemes; the signature scheme in random oracle has cheaper communicational overhead than the scheme in standard model as the former one has public key of constant size, however, the size of the public key in the latter case is linear in the dimension of blocks. The other one is to compare the processing overhead of two algorithms(signing and verification algorithms) in those schemes; as shown, the time needed for these two operations is comparable in random oracle/standard models in some senses.

Our work focuses on two operations(the verification of signatures and the combination) in four schemes from an experimental perspective. The results output from the four implementations are used as statistical evidences for the comparisons. Based on the statistics, we have found that (1) two RSA-based schemes are not comparable because of the combine operation and (2) two

bilinear-based schemes are not comparable because of the verification of signatures; besides, we have seen that the time difference is getting significant when the values of m, n or k increase. To emphasize, the number of a file is divided into m and the magnitude of the coordinates in initial vectors M are established by the specific network coding application; once the application is chosen and the size of the file to be delivered is known, we can adjust the value of n to let m and M in the predefined domains. Normally, in a network coding scheme, if the number of the blocks stored on the intermediate nodes exceeds the number of blocks a file is divided into, the nodes will discard extra incoming blocks that have the same file identifier since the extra blocks will not pass the verification of linear independence.

There are three aspects we need to mention which concern about how the random oracle/standard models affect the performance of network coding schemes.

First, strictly speaking, the homomorphic network coding signature schemes in random oracle/standard models are not comparable especially with larger m, n and k(m here is the number of downloaded blocks and n and k are as defined above). We have seen that the time difference of the schemes in the same group(*RSAGroup* or *BilinearGroup*) increases linearly with the number of downloaded blocks m; in an application with a large number of the blocks a file is divided into, the number of downloaded blocks can be very large. To transmit a large file in a certain application, the dimension of the information part of the blocks n has to be set to a relatively large value to let M meet the condition; this will greatly lead to the increase of time difference since the operations that depend on n will become costly. In a secure-critical system, we need to raise the security parameter k to achieve a higher security level which will cause a growth in the size of other parameters and then raise the processing time on the intermediate nodes.

Second, the signature schemes in random oracle/standard models are somewhat comparable with small m and n and moderate k. The condition 'small m and n' is saying that the number of downloaded blocks on the intermediate nodes and the dimension of the information vectors are small. As mentioned, to limit the number of downloaded blocks, we can choose an application that set the number of blocks a file is divided into to a small number. For the value of n, when a small file is to be transmitted, we can play with n and set it to a small value provided the condition of M(as defined above) is met. Clearly, to achieve a higher security level, more computations are required and this will lead to worse performance. Therefore, for an application that is not secure-critical and is used to transmit small files, the signature schemes in standard model(*SRSA* and

SSDH) would be preferred; among *SRSA* and *SSDH*, *SRSA* is preferable since it is time-saving to achieve the same security level as the *SSDH*.

Finally, a more customized implementation should be introduced to reduce the processing overhead. From the above two aspects, there are circumstances in which the schemes in random oracle/standard models are somewhat comparable. To design a network coding application, we should carefully analyse the situation where network coding signature schemes are applied and then decide whether to use the schemes in random oracle or the schemes in standard model. To this end, elements like the size and the security level of the files should be taken into consideration; and it is also worthwhile to consider the upper bound of each coordinates in the blocks etc.. For instance, in a network coding application used to send small or large files, the number of blocks a file is divided into is fixed to a small value; either *SRSA* or *SSDH* can be applied since their time difference doesn't depend on the value of n and is insignificant with small m; if the application set M into large value, then the processing time in each scheme will then be greatly reduced since n will decrease. However, under these circumstances, two RSA-based schemes are only comparable when the file is small since their time difference depends on the values of m and n; if the file is larger, the time difference will markedly extend.

Those are three perspectives concluded from what we have done to find out whether the standard schemes perform as well as the random oracle schemes. Besides those, we also find that the homomorphic signatures are far more expensive than the vanilla network coding schemes. Though homomorphic signatures can achieve a high security level to combat pollution attacks, efficiency is to be sacrificed since additional and expensive computations are done to achieve extra security.

8.2 Critical Evaluations

The project is to compare the computational overhead of homomorphic network coding signatures in random oracle and standard models to see whether the random homomorphic signatures are comparable to the standard homomorphic signatures. The purpose is fulfilled by implementing four secure network coding signature schemes and comparing them group by group(*RSAGroup* and *Bilinear-Group*). These choices are viewed as the schemes with a better performance than other existing public-key homomorphic network coding signatures over the same algebraic setting.

We illustrate three operations(*vry_NC*(), *vry_Sig*() and *combine*()) that will lead to the cryptographic overhead on the intermediate nodes according to the

communication model of homomorphic signatures in Section 5.2; and then we show that the *vry_NC*() will not result in the time difference by demonstrating the implementation details of *vry_NC*() in Section 6.1; these lead to the result that our comparison concentrates on two operations, *vry_Sig*() and *combine*().

Normally, in a network coding application, security parameter k, the upper bound of each coordinate in the initial vectors M and the number of blocks a file is divided into m are application-specific and they are fixed once a certain network coding application is chosen. Notice that when m and M are fixed and the generation size(the size of a file) is known, the value of n is determined. In our implementation, each measurement is based on m blocks and, for simplicity, we assume there is no download delay on the intermediate nodes; that is, we assume that the number of downloaded blocks is equal to the number of blocks a file is divided into. In *RSAGroup* and *BilinearGroup*, we measure how the processing time of the verification and combination algorithms changes with the number of downloaded blocks m or the dimension of each block n respectively. The experimental results are shown in bar charts or three-D line charts. These figures give a vivid illustration of the variation tendency of the processing delay when m or n increases and also how the time difference of cryptographic overhead of the schemes in each group varies with m or n. The experimental results are used to analyse the computational overhead in the signature schemes and can be used to predict the changing tendency when m and n increase together or m or n keeps growing to larger values. In addition, in *RSAGroup*, we examine how the computational overhead changes when the security parameter k is doubled; to achieve a higher security level, expensive computational overhead has to be traded as well as the significant communicational overhead.

We have shown the precise implementation details including how we choose the parameters as defined in the schemes in Chapter 6. The results are concluded from the figures constructed from the experimental data, and in accordance with each result, the analysis is discussed in detail from a theoretical point of view and closely related to the definitions of the schemes.

Based on the implementation results of the four chosen schemes, we are able to answer the question we pose at the very beginning of the project. Strictly speaking, homomorphic signatures in standard model are not comparable to the schemes in random oracle model when the network coding applications are used to transmit large files and require a high security level; however, if only small files are transferred and the files needn't to be at a very high security level, the signature schemes in random oracle/standard models are somewhat comparable.

To conclude, the purpose of the project has been achieved by showing (1) what operations will result in computational overhead, (2) how to measure and compare each operation, (3) a detailed illustration of the experimental results which are associated with theoretical analyses and (4) a precise conclusion. The project is useful for further research on the performance of homomorphic signatures in random oracle and standard models and can serve as guidelines for choosing homomorphic signatures to apply to certain applications.

8.3 Possible Extensions

This section is going to discuss four possible extensions of the project. The first three possible extensions are aimed at improving the performance of the signature schemes and reducing the time difference of the schemes in random oracle/standard models; the last extension recommends a more practical way of comparing the schemes in different oracle models.

Optimized Pairing. To achieve a better performance of pairing operation, a good knowledge of pairing is required to select an elliptic curve to optimize the operations that reach a certain security level. There is no doubt that this is a complex task. In our project, PBC library is used to implement the pairing operation; as mentioned, the advantage of this library is that it allows the programmer to do the implementation of pairing with little knowledge of cyclic groups and the properties of pairing. Our choice of pairing is based on the descriptions of the document of PBC library; as shown, type D pairing which is based on the curve generated by applying complex multiplication method is the fastest type(offered by the library) to generate asymmetric pairing. Though type D pairing is faster than other types offered in the library, it has the drawback that the element size in $\mathbb{G}'$ is so large that it has resulted in the expensive exponentiation with the base number chosen from $\mathbb{G}'$. If an optimized elliptic curve can be chosen to reduce the processing time of pairing operation as well as other operations with the bilinear groups involved, the performance of two bilinear-based schemes will be improved to a certain extent, and the time difference of their computational overhead will become less significant.

Optimized Implementation. For the optimization of implementation, there are two aspects we will mention. First, it would be beneficial to take into account the implementation of certain operations. Our implementation just follows the definitions of the schemes and the equations in the schemes are realized step by step. Considerations like the optimization of multi-exponentiation are not in the schedule of our project. As mentioned, the main operation that causes

the time difference in two RSA-based schemes is one multi-exponentiation in the combine algorithm; if the time for multi-exponentiation can be reduced, then the condition that two RSA-based schemes are comparable is going to be more flexible. Besides, for the realization of the random oracle, we use the SHA2 hash function in OpenSSL. Hash values are calculated on the intermediate nodes in random oracle schemes; this will cause the processing overhead. It is wise to think about choosing an efficient and secure hash function or calling a more efficient SHA2 function in other libraries since the choice of hash function is the key point to implement the random oracle schemes. The optimization of the implementation should be aimed at achieving a better performance as well as reaching the required security level.

Batch verification. We have mentioned that three operations($vry\_NC()$, $vry\_Sig()$ and $combine()$) will lead to the computational overhead on the intermediate nodes. Among those three operations, the verification of signatures $vry\_Sig()$ is the most expensive one; so to reduce the processing overhead, we can focus on the optimization of the $vry\_Sig()$ algorithm by applying batch verification. Batch verification[4] is an important technique to speed up block verification. When an intermediate node receives a number of blocks, say μ, instead of calling the verification function μ times on each block, it first combines μ blocks into a new coded one and then verifies the new block; if the new block passes the verification check, this means that μ blocks pass the verification check. Batch verification has greatly reduced the frequency of verification from μ to 1; the verification time has improved $1 - \frac{1}{\mu}$. To apply batch verification, four signature schemes in our project will speed up, especially two bilinear-based schemes which require two pairing operations in one $vry\_Sig()$; furthermore, the time difference in two bilinear-based schemes will be significantly reduced(since their time difference is mainly caused by $vry\_Sig()$).

Simulation. It is meaningful to compare the signature schemes in random oracle/standard models by putting into practice the signature schemes using simulation technique. The comparison in the project focuses on the computational overhead of the schemes in different oracle models on the intermediate nodes; the download delay is assumed to be zero and the communicational overhead is not taken into consideration either. To do the comparison in a more practical way, we need to implement the schemes in certain network scenarios and compare them by taking into account the communicational overhead and the download delay. There are many applications for building network simulators, such as OMNeT++[22] and QualNet[24]; they offer many network frameworks to support wireless ad-hoc networks, sensor networks or photonic networks along with the network emulation

and real-time simulation. Besides, the simulation will serve as a method to check the feasibility of implementation by examining the throughput of the network and the decoding probability(the probability to recover the original file) at the targets.

The above four possible extensions are proposed based on the project and they are meaningful points that future works can be focused on.

References

[1] R. Ahlswede, Ning-Cai, S. Li, and R.W. Yeung. Network information flow. *IEEE Transactions on Information Theory*, 46(4):1204–1216, 2000.

[2] Nuttapong Attrapadung and Benoit Libert. Homomorphic network coding signatures in the standard model. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *PKC 2011: 14th International Workshop on Theory and Practice in Public Key Cryptography*, volume 6571 of *Lecture Notes in Computer Sciene*, pages 17–34, Taormina, Italy, March 6-9 2011. Springer, Berlin, Germany.

[3] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of primie order. In *Selected Areas in Cryptography*, pages 319–331, 2005.

[4] M. Bellare, J. Garay, and T. Rabin. Fast batch verification for modular exponentiation and digital signatures. In *Proc. Advances in Cryptology(EUROCRYPT'98),LNCS*, volume 1403, pages 236–250, 1998.

[5] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A pradigm for designing efficient protocols. In *ACM Conference on Computers and Communication Security*, pages 62–73, Nov. 1993.

[6] Dan Boneh and Xavier Boyen. Short signatures without random oracles and the sdh assumption in bilinear groups. *Journal of Cryptology*, 21(2):149–177, April 2008.

[7] Dan Boneh, David Freeman, Jonathan Katz, and Brent Waters. Signing a linear subspace: Signature schemes for network coding. In Stanislaw Jarecki and Gene Tsudik, editors, *PKC 2009: 12th International Conference on Theory and Practice of Public Key Cryptography*, volume 5443 of *Lecture Notes in Computer Science*, pages 68–87, Irvine, CA, USA, March 18-20 2009. Spring, Berlin, Germany.

[8] Dan Boneh and David Mandell Freeman. Homomorphic signatures for polynomial functions. In Kenneth G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, Tallinn, Estonia, May 15-19 2011. Springer, Berlin, Germany.

[9] Dan Boneh and David Mandell Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *PKC 2011:*

14th International Workshop on Theory and Practice in Public Key Cryptography, volume 6571 of *Lecture Notes in Computer Science*, pages 1–16, Taormina, Italy, March 6–9 2011. Springer, Berlin, Germany.

[10] Ran Canetti, Oded Goldreich, and shai Halevi. The random oracle methodology, revisited. In *30th Annual ACM Symposium on Theory of Computing*, pages 209–218, Dallas, May 1998.

[11] Dario Catalano, Dario Fiore, and Bogdan Warinschi. Efficient network coding signature in the standard model. In *PKC 2012*. Cryptology ePrint Archive http://eprint.iacr.org/2011/696.

[12] Dario Catalano, Dario Fiore, and Bogdan Warinschi. Adaptive psedo-free groups and applications. In Kenneth G. Paterson, editor, *Advances in Cryptology - EUPROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 207–223, Tallinn, Estonia, May 15-19 2011. Springer, Berlin, Germany.

[13] Rosario Gennaro, Jonathan Katz, Hugo Krawczyk, and Tal Rabin. Secure network coding over the integers. In Phong Q. Nguyen and David Pointcheval, editors, *PKC 2010: 13th International Conference on Theory and Practice of Public Key Cryptography*, volume 6065 of *Lecture Notes in Computer Science*, pages 142–160, Paris, France, May 26-28 2010. Springer, Berlin, Germany.

[14] T. Ho, R. Koetter, M. Medard, D. Karger, and M. Effros. The benefit of codding over routing in a randomized setting. In *Proc. of International Symposium on Information Theory (ISIT)*, page 442, 2003.

[15] T. Ho, B. Leong, R. Koetter, M. Medard, M. effros, and D. Karger. Byzantine modification detection in multicast networks using randomized network coding. In *Proc. of International Symposium on Informatino Theory (ISIT)*, pages 144–152, 2004.

[16] T. Ho, M. Medard, R. Koetter, D. Karger, M. Effros, J. Shi, and B. Leong. A random linear network coding approach to multicast. *IEEE Transactions on Information Theory*, 52:4413–4430, 2006.

[17] Dennis Hofheinz and Eike Kiltz. Programmable hash functions and their applications. In David Wagner, editor, *Advances in Cryptology - CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 21–38, Santa Barbara, CA, USA, August 17–21 2008. Springer, Berlin, Germany.

[18] S. Jaggi, M. Langberg, S. Katti, T. Ho, D. Katabi, M. Medard, and M. Effros. Resilient network coding in the presence of byzantine adversaries. *IEEE Transactions on Information Theory*, 54:2596–2603, 2008.

[19] J. Katz and Y. Lindell. Introduction to modern cryptography. In *Chapman and Hall/CRC Cryptography*, page 275, 2008.

[20] GMP Library. http://gmplib.org/.

[21] PBC Library. http://crypto.stanford.edu/pbc/.

[22] OMNeT++. http://www.omnetpp.org/.

[23] OpenSSL. http://www.openssl.org/.

[24] QualNet. http://www.scalable-networks.com/content/.

[25] Shuo-Yen Robert-Li, Raymond Y. Yeung, and Ning Cai. Linear network coding. *IEEE Transactions on Information Theory*, 49(2):371–381, 2003.

[26] Brent Waters. Dual system encryption: Realizing fully secure ibe and hibe under simple assumptions. In Shai Halevi, editor, *Cryptology - CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 619–636, CA, USA, Auguest 16-20 2009. Springer, Berlin, Germany.