# Improving the Efficiency of Homomorphic Encryption Schemes

by

Yin Hu

A Dissertation
Submitted to the Faculty
of the
WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the
Degree of Doctor of Philosophy
in
Electrical and Computer Engineering

---

May 2013

APPROVED:

| | |
|---|---|
| Professor Berk Sunar<br>Dissertation Advisor<br>ECE Department | Professor Lifeng Lai<br>Dissertation Committee<br>ECE Department |
| Professor Kathryn Fisler<br>Dissertation Committee<br>Computer Science Department | Professor Wenjing Lou<br>Dissertation Committee<br>Computer Science Department<br>Virginia Tech |

**Abstract**

In this dissertation, we explore different approaches to practical homomorphic encryption schemes. For partial homomorphic encryption schemes, we observe that the versatility is the main bottleneck. To solve this problem, we propose general approaches to improve versatility of them by either extending the range of supported circuits or extending the message space. These general approaches can be applied to a wide range of partial HE schemes and greatly increase the number of applications that they support. For fully homomorphic encryption schemes, the slow running speed and the large ciphertext are the main challenges. Therefore, we propose efficient implementations as well as methods to compress the ciphertext. In detail, the Gentry Halevi FHE scheme and the LTV FHE scheme are implemented and the resulting performance shows significant improvement over previous works. For ciphertext compression, the concept of scheme conversion is proposed. Given a scheme converter, we can convert between schemes with compact ciphertext for communication and homomorphic schemes for computation.

**Acknowledgements**

I would like to sincerely thank to my advisor: Professor Berk Sunar, who has guided me through my Ph.D career. Special thanks to Professor Kathryn Fisler, Professor Wenjing Lou and Professor Lifeng Lai for serving on my committee.

I wish to thank all my teachers for their help during the six years of my studies at WPI, especially Professor William J. Martin and Professor Xinming Huang.

I also would like to thank all my fellow students for their help and company, special thanks to Michael Moukarzel, Chenguang Yang, Yarkın Doröz, Ghaith Hammouri and Wei Wang.

Finally, I would like to express my deepest gratitude to the most important people in my life: my parents. Without their constant support and continuous trust, this dissertation can not be achieved.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

In recent years, distributed systems and especially cloud computing are developing at a high speed. The economic benefits achieved through resource sharing and the greater degree of flexibility in scaling resources have pushed the cloud into mainstream computing.

However, the cloud inherits most information security problems from traditional computing platforms. In addition, the distributed nature of the cloud enables many new types of attacks. There are several major problems that the cloud faces:

- **The cloud may be untrusted.** The cloud service provider (CSP) is not necessarily trusted. For example, a malicious Google employee may be able to setup back doors and bypass all the protection over the Google cloud services. In addition, some machines in the cloud may be mismanaged, making them vulnerable to attacks. Even further, some machines may belong to attackers.

- **Implementation bugs can be exploited.** Even if the CSPs are trusted and they provide isolation mechanisms such as sandboxing and virtualization. Bugs in the system of which more are discovered every day, may be exploited to circumvent any protection, e.g. [1]. As an example, in [2] the authors

show that an attacker could take control of the VMware and Xen virtualization software when moving a virtual machine from one physical computer to another.

- **Side channel attacks can bypass protection.** Even if the system is fully secure and the code is executing in a trusted environment, the side channel attacks may still compromise the security. For example, an attacker using the cold boot attack [3] is able to retrieve sensitive data from the unrefreshed DRAM after using a cold reboot to restart the machine. An attacker using the branch prediction attacks [4] can gather information about the encryption keys by simply monitoring the CPU time. These attacks typically require physical access to the machines, which is not an easy task traditionally. However, in cloud computing settings, it is possible that your code will be executed in a machine belongs to the attacker. In such cases, the attacker will be able to gain physical access to the machine easily.

Secure cloud computing protocols can help to overcome these challenges. As a result, the secure function evaluation (SFE) is gaining more and more significance since it may be employed to design computation delegation schemes and secure cloud computing protocols. SFE provides an important tool when designing protocols involving various parties performing computation together while each party still keeps some of their information secret. A simple setting of the SFE can be abstracted as follows. Alice has the data $x$ and Bob has the function $f$. Alice wants Bob to calculate $f(x)$ for her without revealing the input $x$ to Bob. Depend on the setting, Alice may or may not have access to the function $f$ and Bob may or may not have access to the result $f(x)$. Applying this simple setting to the cloud scenario generates a secure cloud computing protocols, i.e. the clients can evaluate functions

on the cloud without revealing the input to the cloud service provider.

Numerous SFE schemes have been introduced in the literature. One early solution was the garbled circuit construction proposed by Yao in [5]. This approach can work for virtually on any function $f$. However, the communication cost for this approach is high since the ciphertext size grows at least linearly in size of $f$. Sander, Young and Yung proposed another approach to evaluate any constant fan-in Boolean circuit in $NC^1$ [6]. Unfortunately, again the communication complexity is exponential in the depth of the circuit implementing $f$.

## 1.1 Homomorphic Encryption Schemes

The development of homomorphic encryption provides yet another clear-cut approach to build SFE protocols. Informally, a homomorphic encryption scheme allows computation directly on encrypted data. It is clear that a SFE protocol can also be build quite straightforward using HE. Alice can now encrypt the input $x$ and send the ciphertexts to Bob. Bob will compute $f(x)$ directly on the ciphertext and send back the encrypted result that only Alice can decrypt. In this way, Bob will not be able to learn anything about $x$ as long as the security of the homomorphic encryption scheme holds. Homomorphic properties of standard public key encryption schemes, e.g. RSA and ElGamal encryption, were recognized early on [7]. However they were largely viewed as a weakness rather than an asset. Applications where data is static typically require non-malleable encryption. However, the community has grown to trust the security of these schemes and, recently, the work of Gentry and others demonstrate that, when carefully employed, such homomorphic properties can be quite valuable. Indeed, a number of recent specific applications such as data aggregation in distributed networks [8, 9], electronic voting [10], biometrics [11] and privacy

preserving data mining [12] have led to reignited interest in homomorphic schemes. Many powerful HE schemes were proposed during the past decades. One of the earliest discoveries relevant here was the Goldwasser-Micali cryptosystem [13] whose security is based on the quadratic residuosity problem which allows homomorphic evaluation of a bitwise exclusive-or (XOR). Other additive homomorphic encryption schemes that provide semantic security are Benaloh [14], Naccache-Stern [15], Paillier [16], Damgård-Jurik [17], Okamoto-Uchiyama [18] and Boneh-Goh-Nissim [19]. Some additively homomorphic encryption schemes use lattices or linear codes [20, 21, 22, 23, 24]. These HE schemes only support homomorphic evaluation of certain classes of circuits. e.g. the Goldwasser-Micali Scheme only supports bitwise exclusive-or, Paillier's Scheme only supports additions. A number of techniques were proposed to expand the classes of circuits the schemes handle. For instance, the chaining encryption scheme introduced by Melchor, Gaborit and Herranz [23] allows homomorphic computation of functions expressible as $d$-operand products of terms, each of which is a sum of inputs. The products of sums are effectively CNF circuits. However, the ciphertext size of the chaining scheme will grow exponentially with $d$, which limits the circuits supported by this scheme.

One of the most significant developments in cryptography in the last few years has been the introduction of the first fully homomorphic encryption scheme by Gentry [25]. Since addition and multiplication on any non-trivial ring constitute a Turing-complete set of gates, a fully homomorphic encryption scheme – if made efficient – allows one to employ untrusted computing resources without risk of revealing sensitive data. Computation is carried out directly on ciphertexts and the true result of circuit evaluation is not revealed until the decryption stage. In addition to this powerful applicability, Gentry's lattice-based scheme appears to be secure and hence settles an open problem posed by Rivest et al. in 1978 [7].

Inspired by this milestone work, a variety of new schemes are proposed in recent years [26, 27, 28]. In the same time, implementation of proposed FHE schemes also achieved impressive performance [29, 30]. However, the achieved efficiency is still far from being enough to support practical applications.

## 1.2 A Brief Overview of this Thesis

The goal of this dissertation is to improve the efficiency of homomorphic encryption schemes and move them to practice. To achieve this goal, we propose varieties of ways to improve existing HE schemes. In detail, we observe common problems of the partial HE schemes and the FHE schemes and propose different possible solutions to them, including new protocols and more efficient implementations.

**Extending Message Space of HE Schemes.** Some HE schemes have high potential in practical applications however are limited by small message space. e.g. the ElGamal-type encryption schemes are homomorphic with respect to one algebraic operation, however, message recovery involves solving a discrete logarithm problem in the group, which means only small message can be efficiently recovered. The BGN [19] scheme which could be employed to evaluate 2-DNF formulas has the same problem. To solve this problem, we employ the CRT to replace one discrete logarithm problem in a large space by several similar problems in a more tractable search space while retaining full security. This work appears in the Industrial Track of the ACNS 2012 [31].

**Extending Supported Circuits of Partial HE Schemes.** Clearly, the main problem of the partial HE schemes is the range of the circuits that they support. Most partial HE schemes only support one type of operations (either addition or multiplication in most cases) while we need two types of operations for a large

number of practical applications. To solve this problem, we propose a special family of partial HE schemes and discuss ways to convert between different partial HE schemes that support different types of operations. In this way, we can convert to corresponding schemes when certain types of operations are required. This approach can extend the supported circuits of partial HE schemes significantly.

**Efficient Implementation of FHE schemes.** Speed is one of the major bottlenecks that stop existing FHE schemes being practical. To solve this problem, we propose efficient implementation for the Gentry-Halevi FHE scheme and the Lopez-Alt, Tromer and Vaikuntanathan (LTV) FHE scheme. In detail, for the Gentry-Halevi FHE scheme, we adopt NTT based multiplications via GPUs along with other optimizations and achieve about 174, 7.6 and 13.5 times faster than the original Gentry-Halevi code for encryption, decryption and recryption respectively. This result of this joint work that appears in the proceeding of 2012 IEEE HPEC [32]. The LTV FHE scheme is a recently proposed scheme without previous implementation results. We analyze the parameter selection of it and implement the scheme using the selected parameters. The implementation results show promising speed of millisecond level encryption, decryption and relinearization operations.

**Compressing Ciphertexts.** The large ciphertext size of existing FHE schemes is another major problem of the FHE schemes. It will generate large bandwidth requirement if the applications require transferring ciphertext through the network. To address this issue, we formalized the concept of scheme conversion between different encryption schemes originally mentioned in [33, 34, 35]. In addition, we provide efficient instantiation of the conversion between FHE schemes and other encryption schemes with small ciphertext sizes. This enables us converting to schemes with small ciphertext sizes for communication and to FHE schemes for homomorphic evaluation. In this way, we can "compress" the ciphertexts of the FHE schemes and

reduce the bandwidth requirement.

In the rest of the dissertation, we will present an overview of the existing HE schemes first, followed by the discussion of the challenges and possible approaches to overcome them. After that, we will discuss our solutions in detail.

# Chapter 2

# Overview of Homomorphic Encryption Schemes

In this chapter, we will briefly introduce some of the existing partial and fully homomorphic encryption schemes. After that, we will present the advantage of disadvantages of these two families of HE schemes respectively and discuss the approaches to improve them toward practical HE schemes.

## 2.1 Partial Homomorphic Encryption Schemes

Homomorphic properties of several standard public key encryption schemes were recognized early on [7]. Both the RSA and ElGamal encryption schemes were immediately seen to have homomorphic properties, but only with respect to one operation. Ironically, this aspect of these schemes was largely seen as a weakness rather than an asset. Applications where data is static typically require non-malleable encryption. However, the community has grown to trust the security of these schemes and, recently, the work of Gentry and others demonstrates that, when carefully em-

ployed, such homomorphic properties can be quite valuable. Indeed, a number of recent specific applications such as data aggregation in distributed networks [8, 9], electronic voting [10], biometrics [11] and privacy preserving data mining [12] have led to reignited interest in homomorphic schemes.

A list of prominent partially homomorphic schemes is presented in Table 2.1. One of the earliest discoveries relevant here was the Goldwasser-Micali cryptosystem [13] whose security is based on the quadratic residuosity problem and which allows homomorphic evaluation of a bitwise exclusive-or. This scheme has already been applied to the problem of securing biometric information [11]. Other additive homomorphic encryption schemes that provide semantic security are Benaloh [14], Naccache-Stern [15], Paillier [16], Damgård-Jurik [17], Okamoto-Uchiyama [18] and Boneh-Goh-Nissim [19]. Some additively homomorphic encryption schemes use lattices or linear codes [20, 21, 22, 23, 24]. For instance, the lattice-based encryption scheme introduced by Melchor, Gaborit and Herranz [23] allows homomorphic computation of functions expressible as $d$-operand products of terms, each of which is a sum of inputs.

Of particular interest to us is the Boneh-Goh-Nissim partially homomorphic encryption scheme [19], which allows evaluations of arbitrary 2-DNFs, i.e., functions whose evaluation requires one multiplication per term followed by an arbitrary number of additions of terms. The scheme is based on Paillier's earlier additive partially homomorphic scheme and bilinear pairing. As a consequence, the Boneh-Goh-Nissim scheme allows the secure evaluation of degree-two multivariate polynomials, with dot product computation being a particularly useful primitive arising as a special case. Paillier's scheme is the most efficient among currently known additively homomorphic schemes. Therefore, it is employed by some of our works as a building block or as a basis for comparison.

| Scheme | Homomorphism | Computation |
|---|---|---|
| Textbook RSA | Multiplicative | Mod. Exp. in $Z_{pq}$ |
| Textbook ElGamal | Multiplicative | Mod. Exp. in $GF(p)$ |
| Goldwasser Micali [13] | XOR | Mod. Exp. in $Z_{pq}$ |
| Benaloh [14] | Additive | Mod. Exp. in $Z_{pq}$ |
| Paillier Scheme [16] | Additive | Mod. Exp. in $Z_{(pq)^2}$ |
| Paillier ECC variations [36] | Additive | Scalar-point mult. in elliptic curves |
| Naccache-Stern [15] | Additive | Mod. Exp. in $Z_{pq}$ |
| Kawachi-Tanaka-Xagawa [21] | Additive | Lattice Algebra |
| Okamoto-Uchiyama [18] | Additive | Mod. Exp. in $Z_{p^2q}$ |
| Boneh-Goh-Nissim [19] | 2-DNF formulas | Mod. Exp. in $Z_{(pq)^2}$, Bilinear Map |
| Melchor-Gaborit-Herranz [23] | $d$-op. mult. | Lattice Algebra |

Table 2.1: Survey of partially homomorphic encryption schemes

**Limitation of Partial HE Schemes:** Clearly, partial HE schemes are useful in certain applications. In addition, the efficiency of some partial HE schemes is high enough for practical applications. E.g. the Paillier scheme can perform evaluations in milliseconds level. However, the drawbacks of this family of schemes are also clear.

The main problem of the partial HE schemes is the range of the circuits that they support. Most partial HE schemes only support one type of operation, e.g. additions for Paillier and multiplications for RSA. This draws a heavy restriction on the circuits that the HE schemes can evaluate homomorphically.

Some partial HE schemes supports more than one operation, however, restrictions still exist. The Boneh-Goh-Nissim scheme support one level of multiplications via bilinear maps. This feature enables it evaluating 2-DNF formulas which cannot be evaluated using single-operation partial HE schemes. However, only one level multiplication is supported. The Boneh-Goh-Nissim scheme cannot handle more complicated circuits.

**Somewhat Homomorphic Encryption Scheme:** FHE schemes without "refreshing" the noise can also be employed as partial HE schemes. These schemes

usually support a large number of additions and limited levels of multiplications. HE schemes with this property are usually referred to as *Somewhat Homomorphic Encryption Schemes* (SWHE). Although this type of partial HE schemes can support much more complicated circuits than the single-operation ones, it is still heavily restricted since the limitation on levels of multiplications will eventually be reached.

Some partial HE schemes have additional bottlenecks that prevent them from being employed for practical applications. For example, the Boneh-Goh-Nissim scheme requires a small message size to achieve tractable decryption efficiency, which imposes extra limitation to the scheme.

## 2.2 Fully Homomorphic Encryption Schemes

To support the efficient evaluation of an arbitrary function $f$ we may make use of a powerful class of homomorphic encryption schemes named "fully homomorphic encryption" (FHE) which support efficient homomorphic evaluation of any circuit[1]. Gentry proposed the first FHE scheme [25] based on lattices that supports addition and multiplication circuits for any depth. Since addition and multiplication on any non-trivial ring constitute a Turing-complete set of gates, this scheme – if made efficient – allows one to employ *any* untrusted computing resources without risk of revealing sensitive data. In [26], Marten van Dijk et al proposed a FHE scheme based on integers. In 2010, Gentry and Halevi [29] presented a variant of Gentry's FHE; this publication introduced a number of optimizations as well as the results of the world's first FHE implementation.

Although these earlier schemes have achieved full homomorphism, the performance of these schemes becomes the bottleneck. To address this problem, some

[1]The "efficient" here means in asymptotic. The actual running speed of existing FHE schemes is not considered.

newer FHE schemes were proposed in recent years. In [37, 38, 30] Gentry, Halevi and Smart, propose a customized LWE-based FHE scheme tailored for the efficient leveled evaluation of the AES block cipher without bootstrapping. In [27] Brakerski, Gentry, and Vaikuntanathan proposed a new FHE scheme (BGV) based on LWE problems. Instead of recryption, this new scheme uses other light weighted methods to refresh the ciphertexts. These methods cannot thoroughly refresh the ciphertexts as the recryption does, however they can limit the growth of the noise so that the scheme can evaluate much deeper circuits. The recryption process will serve as an optimization to deal with over complicated circuits instead of a necessary for most circuits. In [28], Lopez-Alt, Tromer and Vaikuntanathan adopted this idea to a modified NTRU [39] scheme from Stehle and Steinfeld [40] and developed a FHE scheme (LTV) that supports multiple public keys. The later FHE schemes significantly improved over earlier constructions in both time complexity and in ciphertext size. Still both the latency and the message expansion rates are roughly 2 orders of magnitude higher than those of public-key schemes.

**Limitation of Fully HE Schemes:** In contrast to the partial HE schemes, the FHE schemes can support both additions and multiplications for unlimited times. This enables them evaluating any Boolean circuits. However, the efficiency of existing FHE schemes is far from practical in terms of both computation speed and ciphertext size.

We use the first FHE implementation proposed by Gentry and Halevi [29] as an example. Despite the impressive array of optimizations proposed with the goals of reducing the size of the public-key and improving the performance of the primitives, encryption of one bit takes more than a second on a high-end Intel Xeon based server, while recryption primitive takes nearly half a minute for the lowest security setting. Furthermore, after every few bit-AND operations a recryption operation must be

applied to reduce the noise in the ciphertext to a manageable level. In addition to the computation efficiency, the Gentry-Halevi scheme requires a ciphertext of more than $780,000$ bits for encrypting a single bit. This huge cipher size creates bottlenecks on bandwidths required to transfer the ciphertexts.

## 2.3 Toward Practical Homomorphic Encryption Schemes

It is clear that one of the most important goals of the researches about the homomorphic encryption schemes is to make them closer to practical applications. In this section, we will discuss possible ways to achieve it. Before we can even discuss the practical HE schemes, we need to define the criteria for a scheme to be considered practical.

The first requirement we set is versatility of the scheme, i.e., the scheme should support a large range circuits. This is the feature that most partial HE schemes are missing. It is easy to see that FHE schemes are extremely powerful in terms of versatility. However, fully homomorphism is not a necessary. The BGV and LTV schemes give perfect example of this. Both schemes can perform recryption and achieve fully homomorphism, however, in most cases, the recryption is left as an optimization. In [30], the full AES rounds are evaluated using the BGV scheme without recryption. In other words, the BGV scheme is used as a partial HE scheme or Somewhat HE scheme instead of a "Fully" HE scheme in this case. In conclusion, the versatility is an important requirement for a HE scheme to be practical. The practical HE schemes that we are searching for may not necessarily be FHE schemes, however, it should support a large enough range of circuits.

Another requirement we set for the practical HE schemes is the efficiency. As we

discussed in previous sections, it is the low efficiency that stops existing FHE schemes from being employed in practical applications. In detail, neither the efficiency in terms of computation or cipher size is satisfactory. Since the approaches to improve the computation speed and to reduce the ciphertext size are quite different, we separate them into two categories. Hence, there are three basic requirements for a practical HE scheme:

- Versatility
- Speed
- Ciphertext Size

The obvious approach to achieve practical HE schemes is to design a new scheme that meets every requirement, i.e., a fast FHE scheme with small ciphertext size. Clearly, this is not an easy task. Alternatively, we can improve existing scheme to make them closer to practical. As we discussed in the previous sections, there are two families of HE schemes exit, i.e., the partial HE schemes and the FHE schemes. If we check the partial HE schemes and the FHE schemes against the three requirements we set, we get the result listed in Table 2.2.

| Type | Versatility | Speed | Ciphertext Size |
|---|---|---|---|
| Partial HE | Low | Fast | Small |
| Fully HE | High | Slow | Large |

Table 2.2: Partial HE schemes vs. FHE schemes

We can see clearly from the table that versatility is the main problem for the partial HE schemes and speed and cipher size are more of a problem for the FHE schemes. Therefore, we will focus on versatility when discussing improving the partial HE schemes and focus on speed and cipher size when discussing improving

the FHE schemes. In the rest of the dissertation, we will first discuss the approaches to improve the Partial HE schemes followed by the ones for FHE schemes.

# Chapter 3

# Improving Partial HE Schemes

## 3.1 Extending Message Space of HE Schemes

It is well known that ElGamal-type encryption schemes are homomorphic with respect to one algebraic operation. However, this feature has been widely dismissed as useless since in the additive context, message recovery involves solving a discrete logarithm problem in the group, and this is precisely the problem whose difficulty ensures security. Our solution is simple: we employ the CRT to replace one discrete logarithm problem in a large space by several similar problems in a more tractable search space while retaining full security. On the one hand, this yields for us a general form for two ElGamal variants which are homomorphic with respect to addition: one based on the ElGamal set-up in the multiplicative group $\mathbb{Z}_p^*$ and another using the group of $\mathbb{F}_q$-rational points on an elliptic curve. These schemes are homomorphic with respect to addition in $\mathbb{Z}$. This simple CRT expansion technique has a second application. We show that this technique solves an open problem in the paper of Boneh, et al. [19], alleviating message size limitations on the BGN encryption scheme which was shown to allow homomorphic evaluation of 2-DNF circuits.

### 3.1.1 The CRT-Based ElGamal Scheme

We now present, in generic form, our CRT-based ElGamal scheme $\mathcal{E}$ which we shall henceforth refer to as the CEG Scheme. The CEG scheme, with security parameter $\lambda$, is specified using four procedures: KeyGen, Encrypt, Decrypt and Eval as follows.

KeyGen: Choose a group $G$ with subgroup $H$ generated by $g \in G$ and require $|H|$ to have at least one large prime factor. Pick[1] $k \xleftarrow{\$} [0, 2^\lambda - 1]$. Compute $h = g^k$. Choose $d_i \in \mathbb{Z}^+$ for $i = 1, \ldots, t$ such that $d = \prod d_i < |H|$ and $\gcd(d_i, d_j) = 1$ for $i \neq j$. Set the message space as $\mathcal{M} = \{0, 1, \ldots, N\}$ where $N < d$. The secret key is $\mathcal{SK} = (k)$ and the public key is $\mathcal{PK} = (g, h, \langle d_1, \ldots, d_t \rangle)$.

Encrypt: A message $m \in \mathcal{M}$ is encrypted as the $t$-tuple of pairs

$$\mathsf{Encrypt}(m) = \langle (g^{\ell_i}, h^{\ell_i} g^{m_i}) \ , \ i = 1, \ldots, t \rangle \ ,$$

where $m_i = m \pmod{d_i}$ and $\ell_i \xleftarrow{\$} [0, 2^\lambda - 1]$.

Decrypt: A ciphertext $c = \langle (u_i, v_i) \ , \ i = 1, \ldots, t \rangle$ is decrypted as follows.

$$\mathsf{Decrypt}(c) = \mathrm{CRT}^{-1} \left( \langle \log_g(v_i u_i^{-k}) \ , \ i = 1, \ldots, t \rangle \right) \ ,$$

where $\mathrm{CRT}^{-1}(\langle m_i \ , \ i = 1, \ldots, t \rangle) = \sum_{i=1}^{t} m_i \frac{d}{d_i} \left( \frac{d}{d_i}^{-1} \bmod d_i \right) \bmod d$. The function $\log_g(\cdot)$ denotes the discrete logarithm with respect to generator $g$.

Eval: Given encryptions of $S$ values $m^{(1)}, \ldots, m^{(S)}$, it is straightforward to obtain an encryption of their sum. We simply perform componentwise multiplication in group $G$. The resulting $2t$-tuple lists $t$ pairs $(g^{L_i}, h^{L_i} g^{\sum m_i^{(r)}})$ where the sum

[1] The notation $k \xleftarrow{\$} \mathcal{U}$ stands for $k$ drawn uniformly at random from a universe $\mathcal{U}$.

of residues $m_i^{(r)} \equiv m^{(r)} \pmod{d_i}$ suffice to reconstruct the sum of the integers $m^{(r)}$ via $\mathrm{CRT}^{-1}$.

**Correctness.** The correctness of the scheme follows from the correctness of ElGamal encryption and the correctness of the CRT. The CRT will yield correct results as long as $d > N$. Let $c = \langle (u_i, v_i)\ ,\ i = 1, \ldots, t \rangle$ be the entrywise sum of valid ciphertexts $c_j$ whose corresponding plaintexts satisfy $\sum \mathsf{Decrypt}(c_j) < \prod d_i$. Then $\langle m_1, \ldots, m_t \rangle = \langle \log_{\mathrm{g}}(v_i u_i^{-k})\ ,\ i = 1, \ldots, t \rangle$ satisfies $g^{m_i} u_i^k = v_i$. So $m = \mathrm{CRT}^{-1}(\langle m_i\ ,\ i = 1, \ldots, t \rangle)$ satisfies $m \bmod d_i = m_i$ as long as $0 \leq m < \prod d_i$.

**Efficiency.** The CEG scheme converts one carefully constructed DLP in the subgroup $H := \langle g \rangle$ into a sequence of tractable discrete logarithm problems in the same group. The factors $d_i$ of the composite modulus $d = d_1 d_2 \ldots d_t$ are pairwise relatively prime and $[1, \max(d_i)]$ represents a tractable search space (for the DLP). If each $d_i$ is not too large (say, $w = 16$ bits each) Alice may retrieve each integer $m_i$, even if she must resort to exhaustive search. But she can exploit standard techniques to do better than this; we present some ideas on this step below. However she obtains these logarithms, Alice can then recover the message $m$ via a simple CRT inversion step.

- CEG-$\mathbb{Z}_p$: **Additive CRT-ElGamal Encryption using a Prime Field.** The integer specialization of the generic CEG will be denoted by CEG-$\mathbb{Z}_p$.

  KeyGen: Choose a large prime $p$ with generator $g$ for $\mathbb{Z}_p^*$; we require $p - 1$ to have at least one large prime factor. Pick a random $k$ as above and set $h = g^k$; the integer $\mathcal{SK} = (k)$ remains secret, while $\mathcal{PK} = (g, h, \langle d_1, \ldots, d_t \rangle)$ are made public, where, as in the general case, the message space is $\mathcal{M} = [0, N]$ and we have small coprime integers $d_i$ with $d = \prod d_i > N$.

Encrypt: A message $m \in \mathcal{M}$ is encrypted as

$$\mathsf{Encrypt}(m) = \langle (g^{\ell_i}, h^{\ell_i} g^{m_i}) \ , \ i = 1, \ldots, t \rangle \ ,$$

where $m_i = m \bmod d_i$.

Decrypt: A given ciphertext $c = \langle (u_i, v_i) \ , \ i = 1, \ldots, t \rangle$ is decrypted as

$$\mathsf{Decrypt}(c) = \mathrm{CRT}^{-1} \left( \langle \log_g(v_i u_i^{-k}) \ , \ i = 1, \ldots, t \rangle \right) \ .$$

The function $\log_g(\cdot)$ denotes the discrete logarithm with respect to generator $g$ in $\mathbb{Z}_p^*$.

- CEG-ECC: **Elliptic Curve Version of CRT-ElGamal Encryption:** The elliptic curve specialization of the above scheme will be denoted by CEG-ECC. For illustrative purposes, we concisely present the three procedures as they specialize to elliptic curve encryption.

  KeyGen: Choose an elliptic curve $\mathsf{E}$ with element $P \in \mathsf{E}$ such that $|\langle P \rangle|$ is a large prime. Pick $k \xleftarrow{\$} [0, 2^\lambda - 1]$. Compute $Q = kP$. Choose $d_i \in \mathbb{Z}$ for $i = 1, \ldots, t$ as in the general case. The secret key is again $\mathcal{SK} = (k)$ and the public key is $\mathcal{PK} = (P, Q, \langle d_1, \ldots, d_t \rangle)$. The message space is again $\mathcal{M} = [0, N]$.

  Encrypt: A message $m \in \mathcal{M}$ is encrypted as $\mathsf{Encrypt}(m) = \langle (\ell_i P, \ell_i Q + m_i P) \ , \ i = 1, \ldots, t \rangle$, where $m_i = m \bmod d_i$ and $\ell_i \xleftarrow{\$} [0, 2^\lambda - 1]$.

  Decrypt: Letting $\log_P(\cdot)$ denote the discrete logarithm in $\langle P \rangle$ with respect to $P$, ciphertext $c = \langle (A_i, B_i) \ , \ i = 1, \ldots, t \rangle$ is decrypted as $\mathsf{Decrypt}(c) = \mathrm{CRT}^{-1} \left( \langle \log_{\mathrm{P}}(B_i - kA_i) \ , \ i = 1, \ldots, t \rangle \right)$.

**Optimizations for the CEG-CRT Scheme**

For the sake of simplicity we focus our attention on the elliptic curve specialization CEG-ECC. Similar optimizations may be applied to the other specializations.

**Encryption.** There are number of optimization techniques we can utilize in the implementation of the CEG scheme. Recall that each component of the CEG ciphertext is in the form $(\ell P, \ell Q + mP)$. The encryption procedure for each component involves only three point-scalar multiplications and one point addition neglecting the generation of the random integer $\ell$. This simple approach improves the speed of the encryption procedure. In addition, the encryption can be further sped up using Shamir's Trick. Since the plaintext $m$ is much smaller than the random number $\ell$, the latency of the encryption procedure is dominated by the computation of $\ell P$ and $\ell Q$. For different CRT components, we would compute different $\ell_i P$ and $\ell_i Q$ however with the same $P$ and $Q$, this is ideally suited for Shamir's Trick [42]. With Shamir's Trick, the complexity of CEG encryption for all $t$ components would become $c \cdot \frac{t+2}{3t}$ instead of $ct$, where $c$ represents the time required for the encryption of a single component.

**Decryption.** The CEG decryption may use the Pollard Kangaroo Algorithm to solve the discrete logarithm problem. More specifically, in the CEG scheme, if we want to solve for $m$ given $C_0 = mP$, where $m \in [0, b]$, we generate random walks from $T_0 = bP$ with iterations defined as $T_i = x_i P + T_{i-1}, x_i = f(T_{i-1})$ where $f$ is a hash function. After a number of steps we place a *trap* $T = (b + \sum x_i)P$. Then we start a similar procedure from $C_0 = mP$ and get $C_i = y_j P + C_{j-1}, y_j = f(C_{j-1})$. If the trap is placed after sufficiently many steps, the second run (the kangaroo) will collide with the trap with a high probability. When this collision happens, we can then solve for $m$ from $m = b + \sum x_i - \sum y_j$. In our experiment the steps $x_i$ and $y_j$

are generated with an average size of $0.5\sqrt{b}$ and the trap is placed after $\sqrt{b}$ steps. It is clear that the expectation of the trap position would be $1.5b$. Therefore, $2\sqrt{b}$ steps are expected before the kangaroo is caught by the trap. The complexity of this algorithm is $O(\sqrt{b})$.

We can exploit the fact that we need to perform $t$ parallel DLP computations to reduce the overall decryption complexity. After the accumulation of $S$ operands with word size $w$ the upper limit of the value of $m$ would be $2^{\frac{w+log(S)}{2}}$. Therefore, the complexity of recovering each CRT component using the Pollard Kangaroo algorithm will be $O(2^{\frac{w+log(S)}{2}})$. However, in recovering the various CRT components, we are solving discrete logarithm problems in the same known group. Therefore, only one trap is required. Therefore, the first phase of the Pollard Kangaroo procedure needs only to be executed for the first CRT component; for all remaining components we need only find the collision. This approach saves about 1/3 of the steps. Therefore, the complexity for all $t$ components together can be reduced to $O(\frac{2t+1}{3} \cdot 2^{\frac{w+log(S)}{2}})$.

**Decryption with Precomputation.** Since we envision a scheme where the same group, the same generator and the same primes $d_i$ will be used repeatedly, we can significantly exploit precomputation techniques to speed up the Pollard Kangaroo Algorithm. For instance, the "trap" can be pre-computed as it is independent of the ciphertext. Also, when we are computing $C_i = y_j P + C_{j-1}, y_j = f(C_{j-1})$ during the second run, the costly multiplication $y_j P$ can be significantly speed up by using table lookup; $\sqrt{b}$ entries are required for such a table.

Moreover, since the search space of our scheme is relatively small, we can even get rid of the Pollard Kangaroo Algorithm and directly apply table lookup techniques on the DLP computation. For instance, a naive approach is to store all $b = S \cdot 2^w$ possible pairs $(i, g^i)$ in a table sorted by the second coordinate. At the expense of substantial storage this reduces the DLP to $t$ lookup operations requiring $tw \log_2(S)$

computational steps. A more reasonable compromise is to precompute a fraction of the search space, say $z$ out of $b$ evenly spaced points. The table lookup remains negligible while the search space is reduced to only $b/z$. After this many iterations $y \mapsto y \cdot g$ we are assured a collision and the table lookup completes the computation of our DLP. For example, for 160-bit CEG-ECC with $b = 2^{32}$ and $z = 2^{16}$, we will need a table of $z = 2^{16}$ rows with each row contains the 32-bit $i$ and the 160-bit $g^i$ (representing elliptic curve points by their $x$ coordinate). (only $x$ part of the points is more than enough.). Then the lookup table contains about $192 \cdot 2^{16}$ bits$\approx$ 1.6 Mbytes. [2] The number of components will not affect the storage overhead as all $t$ components can share the same table.

**Implementation Results**

To evaluate the performance of the CEG scheme, we implemented one CRT component and used the measured performance to estimate the addition, encryption and decryption speed of the full CEG scheme. We also implemented the standard version of Paillier's scheme (Page 7, [16]) for comparison. The implementation is realized on an Intel Core i5 2.4GHz CPU.

In the implementation for both our CEG schemes and Paillier's scheme we made use of the Crypto++ library [43]. The Crypto++ library is modestly optimized. Therefore, the performance of Paillier's scheme may be worse than some optimized implementations [44]. To be thorough we also present results for a more efficient implementation of Paillier's scheme using the MPIR library [45]. But we note that the first implementation may be the most natural one to compare against as our CEG scheme implementations are using the same optimization level.

Several parameter choices such as $w, W, S$ and $t$ will affect the performance of

[2] In practice, we do not need to store the entire operand, but sufficiently many bits, e.g. 64-bits, enough to uniquely identify the point with high confidence.

the CEG scheme. The efficiency of Paillier's scheme is not tied to these parameters. The performance for the original CEG-ECC scheme, CEG-$\mathbb{Z}_p$ scheme and Paillier's scheme for various numbers of summands with various word sizes is given in Table 3.1. We chose a 224-bit NIST curve for the CEG-ECC scheme and selected a Paillier scheme with roughly equivalent security level of 2048-bits [46]. Our CEG-$\mathbb{Z}_p$ scheme implementation also uses a 2048 bit prime number.

| Parameters | Addition | Encryption | | Decryption | |
|---|---|---|---|---|---|
| | | Normal | Shamir's trick | Pollard Kangaroo | Precomputation |
| CEG-$\mathbb{Z}_p$-2048 | | | | | |
| $w = 8, S = 2^{24}, t = 7$ | 0.22 ms | 115.71 ms | | 79.63 sec | 1.35 sec (0.75 Mbytes) |
| CEG-ECC-224 | | | | | |
| $w = 8,\ S = 2^{24}, t = 7$ | 0.22 ms | 12.37 ms | 5.30 ms | 99.61 sec | 2.76 sec (0.75 Mbytes) |
| $w = 16, S = 2^{24}, t = 4$ | 0.12 ms | 7.07 ms | 3.53 ms | 15.93 min | 25.17 sec (12 Mbytes) |
| $w = 8,\ S = 2^{40}, t = 9$ | 0.28 ms | 15.90 ms | 6.48 ms | 8.97 hours | 15.10 min (192 Mbytes) |
| Paillier - Crypto++ | | | | | |
| $n = 2048$ | 0.028 ms | 29.60 ms | | 28.10 ms | |
| Paillier - MPIR | | | | | |
| $n = 2048$ | 0.013 ms | 25.90 ms | | 24.90 ms | |

Table 3.1: Performance comparison of CEG with Paillier's Scheme

From Table 3.1 we can see that the CEG-ECC scheme is about 4 times faster than the Paillier scheme in encryption at comparable security levels. In this simplest approach, the decryption performance of CEG is significantly worse than that of Paillier. However, with precomputation the decryption performance may be improved. The precomputation tables were fixed to have $\sqrt{b} = \sqrt{S2^W}$ rows. For instance, for a very modest precomputation table of 0.75 Mbytes we can reduce the decryption time to less than 3 seconds[3]. In many applications of homomorphic encryption schemes the encryption and evaluation speeds matter more than the decryption speed since typically decryption is performed only once after the computations are completed.

[3]In the table, we assume that 64 bits is sufficient to uniquely identify the points in the precomputation table.

### 3.1.2 Applying CRT to the BGN homomorphic scheme

In this section, we first review the encryption scheme of Boneh, Goh and Nissim (BGN) [19]. We then discuss applying the same CRT approach to the BGN scheme.

#### The BGN homomorphic scheme

**Groups admitting Bilinear Pairing** The BGN scheme uses what is known as a bilinear pairing to effect the required multiplication in evaluating a 2-DNF formula. We use a notation similar to the one in [19] in order to facilitate easier comparison:

1. Let $\mathbb{G}$ and $\mathbb{G}_1$ be two (multiplicative) cyclic groups of finite order $n$;
2. let $g$ be a generator of $\mathbb{G}$;
3. let $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_1$ satisfy three conditions:
   (a) $e(\cdot, \cdot)$ is efficiently computable,
   (b) $e(g, g)$ is a generator of $\mathbb{G}_1$,
   (c) for all $u, v \in \mathbb{G}$ and $a, b \in \mathbb{Z}^+$, $e(u^a, v^b) = e(u, v)^{ab}$.

The map $e$ above is a special case of a *bilinear pairing*; one source of such pairings is Tate pairings and another is Weil pairings [47], but this beautiful and complex mathematics is beyond the scope of this dissertation; we need nothing more here than to know that such maps exist for various groups.

To abstract the construction process in the scheme, define an algorithm $\mathcal{G}$ that given a security parameter $\tau \in \mathbb{Z}^+$ outputs a tuple $(q_1, q_2, \mathbb{G}, \mathbb{G}_1, e)$ where $\mathbb{G}$,$\mathbb{G}_1$ are groups of order $n = q_1 q_2$ and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_1$ is a bilinear pairing. On input $\tau$, algorithm $\mathcal{G}$ works as follows:

1. Generate two random $\tau$-bit primes $q_1, q_2$ and set $n = q_1 q_2 \in \mathbb{Z}$;

2. Generate two groups $\mathbb{G}$ and $\mathbb{G}_1$ of order $n$ such that there exists a generator $g$ for $\mathbb{G}$ and a bilinear pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_1$;

3. Output $(q_1, q_2, \mathbb{G}, \mathbb{G}_1, e)$.

**The BGN Scheme** Using the definition of the bilinear pairings and the construction function $\mathcal{G}(\tau)$ presented above, the BGN scheme can be described as follows. (The presentation here differs slightly from that in [19] in order to match our notation and use.)

$\mathsf{KeyGen}(\tau)$: Given a security parameter $\tau \in \mathbb{Z}^+$, run $\mathcal{G}(\tau)$ to obtain a tuple $(q_1, q_2, \mathbb{G}, \mathbb{G}_1, e)$. Let $n = q_1 q_2$. Pick two random generators $g, u \stackrel{R}{\leftarrow} \mathbb{G}$ and set $h = u^{q_2}$. Then $h$ is a random generator of the subgroup of $\mathbb{G}$ of order $q_1$. The public key is $\mathcal{PK} = (n, \mathbb{G}, \mathbb{G}_1, e, g, h)$. The private key is $\mathcal{SK} = (q_1)$.

$\mathsf{Encrypt}(\mathcal{PK}, m)$: We assume the message space consists of integers in the set $\{0, 1, \ldots, T\}$ with $T < q_2$. To encrypt a message $m$ using public key $\mathcal{PK}$, pick a random $r \stackrel{R}{\leftarrow} \{0, 1, \ldots, n-1\}$ and compute $C = g^m h^r \in \mathbb{G}$. Output $C$ as the ciphertext.

$\mathsf{Add}(C^{(1)}, C^{(2)})$: Addition is quite straightforward in the scheme. To evaluate the sum of two messages homomorphically, pick a random $r \stackrel{R}{\leftarrow} \{0, 1, \ldots, n-1\}$ and compute

$$C = C^{(1)} C^{(2)} h^r = g^{m^{(1)}} h^{r^{(1)}} g^{m^{(2)}} h^{r^{(2)}} h^r = g^{m^{(1)} + m^{(2)}} h^{\tilde{r}}$$

Output $C$ as the resulting ciphertext.

$\mathsf{Mul}(C^{(1)}, C^{(2)})$: The bilinear map is used to perform the one multiplication computation. Set $g_1 = e(g, g)$, and $h_1 = e(g, h)$. Then $g_1$ is of order $n$ and $h_1$ is of

order $q_1$. Also, write $h = g^{\alpha q_2}$ for some unknown $\alpha \in \mathbb{Z}$. Then to evaluate the product of two messages homomorphically, pick a random $r \xleftarrow{R} \{0, 1, \ldots, n-1\}$ and compute

$$\begin{aligned} C &= e(C^{(1)}, C^{(2)})h_1^r = e(g^{m^{(1)}}h^{r^{(1)}}, g^{m^{(2)}}h^{r^{(2)}})h_1^r \\ &= g_1^{m^{(1)}m^{(2)}} h_1^{m^{(1)}r^{(2)}+m^{(2)}r^{(1)}+\alpha q_2 r^{(1)}r^{(2)}+r} = g_1^{m^{(1)}m^{(2)}} h_1^{\tilde{r}} \in \mathbb{G}_1 \end{aligned}$$

Output $C$ as the resulting ciphertext. Note that any number of outputs of multiplication gates (i.e., values produced by the bilinear pairing) will have the form $g_1^{m'}h_1^{r'}$ and these can all be viewed as secure encryptions of the corresponding plaintexts $m'$, but instead in the group $\mathbb{G}_1$, where the same additive homomorphic properties hold.

$\mathsf{Decrypt}(\mathcal{SK}, C)$: To decipher $C$ using private key $\mathcal{SK} = (q_1)$, observe that $C^{q_1} = (g^m h^r)^{q_1} = (g^{q_1})^m$. Let $\hat{g} = g^{q_1}$. To recover $m$, it suffices to compute the discrete log of $C^{q_1}$ base $\hat{g}$. Since $0 \leq m \leq T$ this takes expected time $\tilde{O}(\sqrt{T})$ using Pollard's lambda method [48]. It is clear that the message space of the BGN scheme is limited due to the complexity of this decryption.

### 3.1.3 Applying CRT to the BGN Scheme

As discussed above, the (output) message size of the BGN scheme is limited since decryption requires a discrete logarithm computation. In fact, the BGN paper [19] leaves the message size restriction as an open problem. We find that the application of CRT presents a solution to this problem, i.e. we employ CRT to break large messages into smaller pieces and then encrypt the smaller pieces using the BGN scheme. Since the BGN scheme has semantic security, the overall scheme will still be semantically secure even with smaller message sizes. We present the CRT-BGN

scheme first and discuss its efficiency later.

KeyGen($\tau$): Given a security parameter $\tau \in \mathbb{Z}^+$, run $\mathcal{G}(\tau)$ to obtain a tuple $(q_1, q_2, \mathbb{G}, \mathbb{G}_1, e)$. Let $n = q_1 q_2$. Pick two random generators $g, u \stackrel{R}{\leftarrow} \mathbb{G}$ and set $h = u^{q_2}$. Then $h$ is a random generator of the subgroup of $\mathbb{G}$ of order $q_1$. Suppose the message space consists of integers in the set $\{0, 1, \ldots, N\}$. Choose $d_i \in \mathbb{Z}$ for $i = 1, \ldots, t$ such that $d_i < T$, $d = \prod d_i > N$ and $\gcd(d_i, d_j) = 1$ for $i \neq j$. The public key $\mathcal{PK} = (n, \mathbb{G} - \!\!\!\succ\!\!\!\supset\!\!\!\ltimes, \mathbb{G}_1, e, g, h, \langle d_1, \ldots, d_t \rangle)$. The private key is $\mathcal{SK} = (q_1)$.

Encrypt($\mathcal{PK}, M$): We will encrypt the message with $t$ tuples of BGN scheme. The message space for each tuple will be $\{0, 1, \ldots, T\}, T < q_2$. To encrypt a message $m$ using public key $\mathcal{PK}$, pick random variables $r_i \stackrel{R}{\leftarrow} \{0, 1, \ldots, n-1\}$ and compute $t$-tuple of pairs

$$C = \langle C_i \ , \ i = 1, \ldots, t \rangle = \langle g^{m_i} h^{r_i} \in \mathbb{G} \ , \ i = 1, \ldots, t \rangle \ .$$

where $m_i = m \pmod{d_i}$. Output $C$ as the ciphertext.

Add($C^{(1)}, C^{(2)}$): The addition is done pairwise for the $t$-tuples. To evaluate the sum of two messages homomorphically, pick random variables $r_i \stackrel{R}{\leftarrow} \{0, 1, \ldots, n-1\}$ and compute

$$\begin{aligned} C &= \langle C_i^{(1)} C_i^{(2)} h^{r_i} \ , \ i = 1, \ldots, t \rangle = \langle g^{m_i^{(1)}} h^{r_i^{(1)}} g^{m_i^{(2)}} h^{r_i^{(2)}} h_i^r \ , \ i = 1, \ldots, t \rangle \\ &= \langle g^{m_i^{(1)} + m_i^{(2)}} h^{r_i^{(1)} + r_i^{(2)} + r_i} \ , \ i = 1, \ldots, t \rangle = \langle g^{m_i^{(1)} + m_i^{(2)}} h^{\tilde{r_i}} \ , \ i = 1, \ldots, t \rangle \ . \end{aligned}$$

Output $C$ as the resulting ciphertext. After the multiplication, the addition can be evaluated in the same way using $g_1$ and $h_1$.

$\mathsf{Mul}(C^{(1)}, C^{(2)})$: Similarly, the multiplication is done pairwise for the $t$-tuples. Set $g_1 = e(g, g)$, and $h_1 = e(g, h)$. Then $g_1$ is of order $n$ and $h_1$ is of order $q_1$. Also, write $h = g^{\alpha q_2}$ for some unknown $\alpha \in \mathbb{Z}$. Then to evaluate the product of two message homomorphically, pick random variables $r_i \xleftarrow{R} \{0, 1, \ldots, n-1\}$ and compute

$$\begin{aligned} C &= \langle e(C_i^{(1)}, C_i^{(2)}) h_1^{r_i} \ , \ i = 1, \ldots, t \rangle \\ &= \langle e(g^{m_i^{(1)}} h^{r_i^{(1)}}, g^{m_i^{(2)}} h^{r_i^{(2)}}) h_1^{r_i} \ , \ i = 1, \ldots, t \rangle \\ &= \langle g_1^{m_i^{(1)} m_i^{(2)}} h_1^{m_i^{(1)} r_i^{(2)} + m_i^{(2)} r_i^{(1)} + \alpha q_2 r_i^{(1)} r_i^{(2)} + r_i} \ , \ i = 1, \ldots, t \rangle \\ &= \langle g_1^{m_i^{(1)} m_i^{(2)}} h_1^{\tilde{r}_i} \in \mathbb{G}_1 \ , \ i = 1, \ldots, t \rangle \ . \end{aligned}$$

Output $C$ as the resulting ciphertext.

$\mathsf{Decrypt}(\mathcal{SK}, C)$: To decrypt a ciphertext $C$ using a private key $\mathcal{SK} = (q_1)$, for each tuple we have: $C_i^{q_1} = (g^{m_i} h^{r_i})^{q_1} = (g^{q_1})^{m_i}$ where the $g, h$ here could be $g_1, h_i$ if the multiplication is computed.

Let $\hat{g} = g^{q_1}$. To recover $m_i$, it suffices to compute $t$ tuples of the discrete log of $C_i^{q_1}$ base $\hat{g}$. Since $0 \leq m_i \leq T$ this takes expected time $\tilde{O}(\sqrt{T})$ using Pollard's lambda method. After recovering all the $m_i$ values, the plaintext can be reconstructed as $m = \mathrm{CRT}^{-1}(m_i \mod d_i \ , \ i = 1, \ldots, t)$.

**Correctness:** The correctness of the CRT-BGN scheme follows from the correctness of the BGN and CRT schemes. The correctness of each component in CRT comes from the property of the multiplicative cyclic group. As long as the sub-result of each component is smaller than $T$, the residues can be efficiently decrypted. After the recovery of each $m_i$, if the result of the computation satisfies $m < d = \prod d_i$, then $m$ can be correctly recovered by the definition of CRT.

**Conclusion:** In this section, we propose an approach to overcome the discrete logarithm impasse in the additive version of ElGamal encryption and in the decryption step of the BGN schemes. This new CRT-based technique holds promise for making the additively homomorphic schemes described above more practical. The technique also has potential as a building block for a variety of other protocols where one party needs an advantage over another in computing discrete logarithms.

## 3.2 Extending Supported Circuits of Partial HE Schemes

In previous chapters, we have highlighted several advantages of partially homomorphic encryption (partially HE) schemes over FHE schemes, such as efficiency and compact ciphertexts. Of course, these savings come at the cost of a severely limited suite of circuits that the scheme can evaluate homomorphically. The question then arises as to what sorts of circuits fall "in between" the two paradigms. For example, in [19] a scheme is proposed to homomorphically evaluate any 2-DNF circuit. More generally, a somewhat homomorphic encryption scheme (SWHE, which can evaluate arbitrary circuits of fixed limited depth) can be employed to homomorphically evaluate n-DNF circuits of limited degree and size. While a partially homomorphic scheme is not able to evaluate arbitrary n-DNF circuits, we will now see how a converter may be employed to piece together two partially HE schemes resulting in a merged scheme with much more computational power.

A direct approach would involve a converter from a partially HE scheme to an FHE, which can then perform the evaluation. But, in this case, we may as well use the FHE scheme to do the evaluation in the first place. We gain an advantage only if both schemes are more efficient in some way than the known FHE schemes. Consider

a pair of converters taking an additively HE scheme to a multiplicatively HE scheme and back again[4]. If we are able to convert unlimited numbers of ciphertexts between the two schemes, the resulting merged scheme becomes fully homomorphic! Whether such a pair of converters exists remains unknown. However, there do exist converters that can perform large-but-limited numbers of such conversions; the result of this construct is greatly expanded flexibility for partially HE schemes. For example, this construction makes it possible to evaluate DNF formulas homomorphically.

In the rest of this section, we will first introduce a special family of partial HE schemes and then discuss how these may be used to build a converter with the above-mentioned properties, thereby allowing us to perform homomorphic DNF-formula evaluation.

### 3.2.1 Probabilistic Gates

Given a a set of universal gates such as NAND or OR and AND gates it is possible to realize any boolean function. However, if we are given a *less* than universal set of gates then it becomes impossible to build a *deterministic circuit* to evaluate any boolean function. In contrast, here we present a technique that allows one to build probabilistic gates using a limited set of logic gates. Using the probabilistic gates as a building block we can realize several useful classes of functions. We construct these gates with the aid of a pseudo random permutation function and redundant encoding. The encoding is defined as follows:

**Definition 1 (Encoding Function)** *Let $C$ denote a binary linear code and denote its complement by $\bar{C}$. Then an encoding function $e : \{0,1\} \mapsto \{0,1\}^n$ is defined as $e_C(b) = (1-b)\mathbf{c} \oplus b\bar{\mathbf{c}}$ where $\mathbf{c} \leftarrow C$ and $\bar{\mathbf{c}} \leftarrow \bar{C}$. For completeness, define the*

[4]In practice, a converter from a SWHE that supports more additions to a SWHE that supports more multiplications also has the same effect and is, in fact, a more realistic goal.

*symmetric case as* $\bar{e}_C(b) = b\mathbf{c} \oplus (1-b)\bar{\mathbf{c}}$. *Here* $\oplus$ *is the usual mod 2 addition of binary n-tuples.*

Thus $e_C$ encodes each zero bit to a random element of $C$, and each one bit to a random element of $\bar{C}$. Note that when the context is clear we will drop the subscript for brevity and write $e_C = e$. Next we introduce a probabilistic gate.

**Definition 2 (Probabilistic Gate $\mathsf{G}_p$)** *Given an encoding function* $e_C : \{0,1\} \mapsto \{0,1\}^n$ *(or* $\bar{e}_C$*) a probabilistic gate* $\mathsf{G}_p : \{0,1\} \mapsto \{0,1\}^n$ *is explicitly defined as* $(a\ \mathsf{G}_p\ b) = a \oplus b$.

Here, the subscript $p$ denotes a *failure probability* for the gate $\mathsf{G}_p$. For instance, if $C \subseteq \{0,1\}^n$ and we interpret all tuples in $C$ to be valid encodings of zero and all other binary $n$-tuples to be valid encodings of one, we have a probabilistic $\mathsf{OR}$ gate with

$$\begin{aligned} p &= \Pr[\mathbf{c}_1 \oplus \mathbf{c}_2 \in \bar{C} \mid \mathbf{c}_1, \mathbf{c}_2 \leftarrow C] + 2\Pr[\mathbf{c}_1 \oplus \bar{\mathbf{c}}_2 \in C \mid \mathbf{c}_1 \leftarrow C; \bar{\mathbf{c}}_2 \leftarrow \bar{C}] \\ &\quad + \Pr[\bar{\mathbf{c}}_1 \oplus \bar{\mathbf{c}}_2 \in C \mid \bar{\mathbf{c}}_1, \bar{\mathbf{c}}_2 \leftarrow \bar{C}] \end{aligned}$$

which, for a linear code $C$ with $|C| \ll 2^n$, simplifies to $p = \Pr[\bar{\mathbf{c}}_1 \oplus \bar{\mathbf{c}}_2 \in C \mid \bar{\mathbf{c}}_1, \bar{\mathbf{c}}_2 \leftarrow \bar{C}]$. Likewise, for a binary linear $[n,k]$-code $C$, the encoding function $\bar{e}_C$ yields a probabilistic $\mathsf{AND}$ gate with failure probability $p$ satisfying $1 - p \approx \Pr[e(0) - e(0) \in C] = \frac{|\bar{C}|}{2^n} = 1 - 2^{k-n}$.

**Lemma 1 ($\mathsf{OR}_p$ Gate)** *An* $\mathsf{OR}_{2^{k-n}}$ *probabilistic gate can be constructed using an encoding function* $e_C$ *where* $C$ *is any binary linear* $[n,k]$*-code.*

**Lemma 2 ($\mathsf{AND}_p$ Gate)** *An* $\mathsf{AND}_{2^{k-n}}$ *probabilistic gate can be constructed using an encoding function* $\bar{e}_C$ *where* $C$ *is any binary linear* $[n,k]$*-code.*

**Example 1** *Assume $e_C$ is built using a binary $[4,1]$-code (4-bit repetition code). We have two codewords $C = \{(0000), (1111)\}$. We may then use $e_C : \{0,1\} \mapsto \{0,1\}^4$ given by $\mathbf{b} \leftarrow C$ when $b = 0$ and $\mathbf{b} \leftarrow \bar{C}$ when $b = 1$. Table 3.2 gives a few examples of how reliably the mod 2 addition of encodings represents the encoding of the logical* OR *of the two input bits.*

| Operands | Vector operation | Result | Comment |
|---|---|---|---|
| $\mathbf{0} \oplus \mathbf{0}$ | $(0000) \oplus (1111) = (1111)$ | $\mathbf{0}$ | |
| $\mathbf{0} \oplus \mathbf{1}$ | $(0000) \oplus (0101) = (0101)$ | $\mathbf{1}$ | Logic |
| $\mathbf{1} \oplus \mathbf{0}$ | $(0110) \oplus (1111) = (1001)$ | $\mathbf{1}$ | OR |
| $\mathbf{1} \oplus \mathbf{1}$ | $(0101) \oplus (0110) = (0011)$ | $\mathbf{1}$ | |
| $\mathbf{1} \oplus \mathbf{1}$ | $(0101) \oplus (0101) = (0000)$ | $\mathbf{0}$ | Error |

Table 3.2: A few sample computations with a probabilistic $\mathsf{OR}_p$ gate.

Clearly the gate fails only when both input bits are ones and the failure probability is only 1/254. Therefore, we have constructed a probabilistic OR gate with small failure probability using the repetition code. Clearly the failure probability drops exponentially as the length of the code grows. The idea of using repetition code to compute multiplications first appears in [6]. Our probabilistic gates built from arbitrary linear codes can be viewed as a generalization of that technique.

Consider any partially HE scheme $\mathcal{A}$ which evaluates (XOR) gates. Using the encoding just described, this scheme functions effectively as a partially HE scheme $\mathcal{HE}$ which evaluates probabilistic (AND or OR) gates; so we have achieved (allowing for a controllable failure probability) scheme conversion from an additively HE scheme to a multiplicatively HE scheme. This latter scheme, based on probabilistic gate $\mathsf{G}_p$, will now be denoted by $\mathcal{HG}$. The probabilistic scheme is limited in two respects:

- there exists a controllably small failure probability $p$ associated to every occurrence of gate $\mathsf{G}_p$ which can potentially lead to wrong results;

- only one of the two multiplicative operations is supported at a time; i.e., one may create a partially HE using either $\mathsf{OR}_p$ gates or $\mathsf{AND}_p$ gates. However it is impossible to evaluate $\mathsf{AND}$ operation directly on ciphertexts output by $\mathsf{OR}_p$ gates and vice versa.

The first restriction can be largely removed by increasing code length. Failure probability drops exponentially with code length while the overhead grows only linearly. Therefore, we can achieve exponential reduction in the failure probability with reasonable overhead. The second restriction is actually a common drawback for many partially HE schemes, and this is where the use of a converter has value.

### 3.2.2 Converters for Schemes with Limited Homomorphic Properties

Using the probabilistic gate based HE schemes, we now build a secure converter to achieve conversion from an additively homomorphic scheme to a multiplicatively homomorphic scheme. We introduce the concept of scheme converters to formalized this procedure. The same concept will also be used in later chapters. Here we will give a brief definition of the scheme converter. The details will be covered in later chapters.

**Definition 3 (Converter)** *Given two encryption schemes $\mathcal{A}$ and $\mathcal{B}$, a* **converter** *from $\mathcal{A}$ to $\mathcal{B}$ is an ordered pair $\mathcal{C} = (\mathsf{KeyGen}, \mathsf{Convert})$ such that*

$$\begin{aligned} & & s &= \mathcal{C}.\mathsf{KeyGen}(\mathcal{A}, \mathcal{B}) \\ \textit{satisfies} & & \mathcal{B}.\mathsf{Enc}(x, k_{\mathcal{B}}) &= \mathcal{C}.\mathsf{Convert}(\mathcal{A}.\mathsf{Enc}(x, k_{\mathcal{A}}), s) \ . \end{aligned}$$

*Here, the extra input $s$ required for $\mathcal{C}.\mathsf{Convert}$ is called the* **converter key**. *We then*

*say scheme $\mathcal{A}$ is* **convertible** *to scheme $B$, we refer to $C$ as a* **converter**, *and we refer to the overall process informally as* **scheme conversion**.[5]

Consider an additive HE scheme $\mathcal{HE}$ and a multiplicative HE scheme $\mathcal{HG}$ constructed from $\mathcal{HE}$, we will discuss conversion from $\mathcal{HE}$ to $\mathcal{HG}$ first and the converter for the opposite direction later.

**Converter from $\mathcal{HE}$ to $\mathcal{HG}$.** Designing a converter from $\mathcal{HE}$ to $\mathcal{HG}$ is straightforward. Recall the encoding process defined for probabilistic gates: $e_C(x) = (1-x)\mathbf{c} \oplus x\bar{\mathbf{c}}$. The converter from $\mathcal{HE}$ to $\mathcal{HG}$ can be simply viewed as homomorphically computing $\mathcal{HG}.\mathsf{Enc}(x) = \mathcal{HE}.\mathsf{Enc}(e_C(x))$ from $\mathcal{HE}.\mathsf{Enc}(x)$[6]. Clearly, the additive part in the encoding process can be evaluated homomorphically trivially since $\mathcal{HE}$ is additive. However, we also need to evaluate products, i.e. $(1-x)\mathbf{c}$ and $x\bar{\mathbf{c}}$ where $\mathbf{c}$ and $\bar{\mathbf{c}}$ are random elements of $C$ or $\bar{C}$ which can be viewed as vectors while $x$ and $1-x$ are single bits.

Multiplication is usually hard to evaluate in additive schemes. However, for the encoding process, $\mathbf{c}$ and $\bar{\mathbf{c}}$ are generated in plaintext form and $x$ and $1-x$ are single bits. Therefore, what we need is only bitwise scalar multiplication which can be achieved in the additive setting. For example, $\mathcal{HE}.\mathsf{Enc}(x\bar{\mathbf{c}}) = \mathcal{HE}.\mathsf{Enc}(x)\bar{\mathbf{c}}$ can be computed by checking each bit $\bar{\mathbf{c}}_i$ of $\bar{\mathbf{c}}$. If $\bar{\mathbf{c}}_i$ equals one, put $\mathcal{HE}.\mathsf{Enc}(x)$ the $i$th position of the result. If $\bar{\mathbf{c}}_i$ equals zero, put $\mathcal{HE}.\mathsf{Enc}(0)$ to the $i$th position of the result. With this approach, we can realize the desired products.

Combining the above multiplication approach and the additive homomorphic property of $\mathcal{HE}$, it is clear that a converter from $\mathcal{HE}$ to $\mathcal{HG}$ can be achieved by homomorphically evaluating the encoding process. The security of this converter is easily reduced to the security of $\mathcal{HE}$ since no extra information other than $\mathcal{HE}.\mathsf{Enc}(x)$ is

[5]This notion is not to be confused with the Fujisaki-Okamoto conversion scheme [49], with which it has no relationship.

[6]The encryption of a vector, i.e. $e_C(x)$, is defined as the bit by bit encryption of the $e_C(x)$.

used. If there exists any method to compromise the resulting $\mathcal{HG}.\mathsf{Enc}(x)$, an attacker may then compromise $\mathcal{HE}.\mathsf{Enc}(x)$ by converting it to $\mathcal{HG}.\mathsf{Enc}(x)$ then attacking $\mathcal{HG}$.

**Converter from $\mathcal{HG}$ to $\mathcal{HE}$.** We can construct a converter from $\mathcal{HG}$ to $\mathcal{HE}$ using the homomorphic decryption idea introduced in Section 4.3.2. Similar to the encoding process, the decoding process of the probabilistic gates can also be evaluated as bit operations, which can be easily carried out by $\mathcal{HE}$. However, due to the higher complexity of the decoding process, certain depth of homomorphic multiplication is required in most cases. This means that we have to use SWHE for $\mathcal{HE}$.

For example, suppose the 64-bit repetition code $[64, 1]$ is used for the $\mathcal{HG}$, the decoding process will be simply checking whether all the bits are '1's. It can be done by $\mathsf{AND}$ (multiplying) them together. However, this will require a depth 6 multiplication circuit. If the SWHE scheme we selected for $\mathcal{HE}$ supports multiplication circuits deeper than this, we can then define a converter which simply decodes the probabilistic gate homomorphically. The security of this converter can be proven using an approach similar to the approach in Section 4.3.2. Now that we have conversion techniques in both directions, we can present concise definitions of all four of these converters. Given a somewhat homomorphic encryption scheme $\mathcal{HE}$, we can define $\mathcal{HG}_{\mathsf{AND}}$ and $\mathcal{HG}_{\mathsf{OR}}$ from some linear code $\mathbf{c}$ using the methods discussed in previous sections and define the converters as follows:

$\mathcal{HE}to\mathcal{HG}_{\mathsf{AND}}.\mathsf{Convert}$**:** Evaluate $x\mathbf{c} \oplus (1 - x)\bar{\mathbf{c}}$ homomorphically from $\mathcal{HE}.\mathsf{Enc}(x)$.

$\mathcal{HE}to\mathcal{HG}_{\mathsf{OR}}.\mathsf{Convert}$**:** Evaluate $(1 - x)\mathbf{c} \oplus x\bar{\mathbf{c}}$ homomorphically from $\mathcal{HE}.\mathsf{Enc}(x)$..

$\mathcal{HG}_{\mathsf{AND}}to\mathcal{HE}.\mathsf{Convert}$**:** Decode $\mathcal{HG}_{\mathsf{AND}}.\mathsf{Enc}(x)$ homomorphically.

$\mathcal{HG}_{\mathsf{OR}}to\mathcal{HE}.\mathsf{Convert}$**:** Decode $\mathcal{HG}_{\mathsf{OR}}.\mathsf{Enc}(x)$ homomorphically.

Note that for the above process to work, the number of multiplications supported by $\mathcal{HE}$ must be sufficient to homomorphically decode $\mathbf{c}$. Therefore, this $\mathcal{HE}$ alone is capable of evaluate circuits with large amount of additions and certain depth of multiplications. However, with proper parameters, the hybrid $\mathcal{HE}$-$\mathcal{HG}$ scheme will be able to support a larger range of circuits. We will explain this in the next section using an $n$-DNF formula evaluation scheme as example.

**Security** If we merge $\mathcal{HE}$ and $\mathcal{HG}$ before and after the conversion and view them as a single merged scheme. The resulting merged scheme can be proved to be IND-CPA if the $\mathcal{HE}$ is IND-CPA. The detail will be discussed in Section 4.3.2.

### 3.2.3 Application: Evaluating $n$-DNF Formula

DNF formula can be used to describe complicated circuits. Evaluating DNF formulas homomorphically enables varies of applications, such as homomorphic database operations. In [19], the authors proposed a method to homomorphically evaluate 2-DNF formulas, which supports only one level of AND operation before the OR operations. In this section, we will propose an approach to homomorphically evaluate n-DNF formulas which supports $n$ levels of AND operation.

For simplicity, we assume that the $n$-DNF formulas do not require inversing inputs. Clearly, inversion can be easily evaluated homomorphically with the additive homomorphic property in our setting. Suppose we describe an $n$-DNF formula by $F = (a_1, a_2, \ldots, a_n, a_{n+1}, \ldots, a_{2n}, a_{2n+1}, \ldots, a_{sn})$. Then the result we are looking for can be computed by:

$$result = \bigvee_{j=1}^{s} \left( \bigwedge_{i=1}^{n} a_{(j-1)n+i} \right).$$

Given security parameter $\tau$, using the $\mathcal{HE}$ and $\mathcal{HG}$ and the converters discussed in last section, we can define the homomorphic $n$-DNF formula evaluation procedure

as Figure 3.1. As we discussed above, if the $\mathcal{HE}$ supports $N$ multiplications, the $\mathcal{HE}$ alone can evaluate certain DNF formulas. However, clearly the size of the DNF formulas that $\mathcal{HE}$ supports is limited by[7] $sn \leq N$. For our scheme, the decoding process will consume certain number of multiplications. However, as we can see from Figure 3.1, the decoding process is independent of the input formula. Therefore, the overhead from the decoding is independent of the size of $n$-DNF formula to be evaluated. In addition, the homomorphic multiplications of $\mathcal{HG}$ ciphertexts is evaluated by additions of $\mathcal{HE}$ ciphertexts. Assume that the decoding process requires constant $c$ multiplications and additions generate logarithmic level noise [8], the size of the DNF formulas that our scheme supports is limited by $\log(sn) + c \leq N$. Recall that the same limitation of $\mathcal{HE}$ alone is $sn \leq N$. Clearly our scheme is able to evaluate much larger DNF formulas with proper parameters. Note that if $\mathcal{HE}$ allows more multiplications, we can apply the converter more times to evaluate more complicated circuits. However, the number of times that a given converter can be applied in practice is heavily restricted by the costly decoding process. Therefore, though much larger range of circuits are supported, our new scheme is still partial homomorphic. An interesting question to consider is whether there exists any secure converter that can process an unlimited number of ciphertexts from an additive scheme to a multiplicative scheme. Clearly, the discovery of such a converter gives us a new approach to the construction of FHE schemes.

[7] Here we ignore noise accumulation from additions.
[8] Most of the existing SWHE schemes exhibit this behavior.

**Homomorphic evaluating** $F = (a_1, a_2, \ldots, a_n, a_{n+1}, \ldots, a_{sn})$:

**Setup** :

Run $\mathsf{KeyGen}_{\mathcal{HE}}(\tau)$.

**Encrypt** :

Given $F = (a_1, a_2, \ldots, a_n, a_{n+1}, \ldots, a_{2n}, a_{2n+1}, \ldots, a_{sn})$, compute

$$\begin{aligned} F' &= (a'_1, a'_2, \ldots, a'_{sn}) \ , \\ & \quad where \\ a'_i &= \mathcal{HE}.\mathsf{Enc}(\mathcal{PK}_{\mathcal{HE}}, a_i). \end{aligned}$$

**Evaluate** :

Given $F' = (a'_1, a'_2, \ldots, a'_{sn})$, run the following steps:

1. Compute $\mathsf{AND}$:
   For $j = 1, 2, \ldots, s$
   $b_j = \mathcal{HE}to\mathcal{HG}_{\mathsf{AND}}.\mathsf{Convert}(a'_{(j-1)n+1})$;
   For $i = 2, 3 \ldots, n$
   $b_j = \mathcal{HG}_{\mathsf{AND}}.\mathsf{Eval}(b_j, \mathcal{HE}to\mathcal{HG}_{\mathsf{AND}}.\mathsf{Convert}(a'_{(j-1)n+i}))$
   $c_j = \mathcal{HG}_{\mathsf{AND}}to\mathcal{HE}.\mathsf{Convert}(b_j)$
2. Compute $\mathsf{OR}$:
   $d = \mathcal{HE}to\mathcal{HG}_{\mathsf{OR}}.\mathsf{Convert}(c_1)$;
   For $j = 2, 3, \ldots, s$
   $d = \mathcal{HG}_{\mathsf{AND}}.\mathsf{Eval}(d, \mathcal{HE}to\mathcal{HG}_{\mathsf{OR}}.\mathsf{Convert}(c_j))$
3. Output $d$ as the result.

**Decrypt** :

Decrypt each bit of $d$ and decode the result according to $\mathsf{OR}$ probabilistic gate.

Figure 3.1: n-DNF Formula Evaluation Scheme

# Chapter 4

# Improving FHE Schemes

As we discussed in previous chapters, the low speed and large ciphertext size of existing FHE schemes prevent them from being employed in practical applications. In this section, we will present our efficient implementation of the Gentry-Halevi scheme and the LTV scheme, followed by our scheme conversion approach to compress the ciphertext of FHE schemes.

## 4.1 Implementing the Gentry-Halevi Scheme

### 4.1.1 The Gentry-Halevi FHE Scheme

The first FHE was proposed by Gentry in [25, 50]. However, this preliminary implementation is far too inefficient to be used in any practical applications. The Gentry-Halevi FHE variant with a number of optimizations and the results of a reference implementation were presented in [29]. Here we only present a high-level overview of the primitives and the details can be referred to the original work in [29].

Encrypt: To encrypt a bit $b \in \{0, 1\}$ with a public key $(d, r)$, Encrypt first generates a

random "noise vector" $u = \langle u_0, u_1, \ldots, u_{n-1} \rangle$, with each entry chosen as 0 with some probability $p$ and as $\pm 1$ with probability $(1-p)/2$ each. Clearly, $p$ will determine the hamming weight of the random noise $u$. Gentry showed in [29] that $u$ can contain a large number of zeros without impact the security level, i.e., $p$ could be very large.

Then the message bit $b$ is encrypted by computing

$$c = [b + u\,(r)]_d = \left[ b + 2 \sum_{i=1}^{n-1} u_i r^i \right]_d \tag{4.1}$$

where $d$ and $r$ is part of the public key.

Eval: When encrypted, arithmetic operations can be performed directly on the ciphertext with corresponding modular operations. Suppose $c_1 = \mathsf{Encrypt}(m_1)$ and $c_2 = \mathsf{Encrypt}(m_2)$, we have:

$$\begin{aligned} \mathsf{Encrypt}(m_1 + m_2) &= (c_1 + c_2) \bmod d \\ \mathsf{Encrypt}(m_1 \cdot m_2) &= (c_1 \cdot c_2) \bmod d\ . \end{aligned} \tag{4.2}$$

Decrypt: An encrypted bit can be recovered from a ciphertext $c$ by computing

$$m = [c \cdot w]_d \bmod 2 \tag{4.3}$$

where $w$ is the private key and $d$ is part of the public key.

Recrypt: The Recrypt process is realized by homomorphically evaluating the decryption circuit on the ciphertext. However, due to the fact that we can only encrypt a single bit and that we can only evaluate a limited number of arithmetic operations, we need an extremely shallow decryption method. In [29], the authors discussed a practical way to re-organize the decryption process to make this possible. Informally, the private key is divided into $s$ pieces that satisfy $\sum^s w_i = w$. Each $w_i$

is further expressed as $w_i = x_i R^{l_i} \bmod d$ where $R$ is constant, $x_i$ is random and $l_i \in \{1, 2, \ldots, S\}$ is also random. The recryption process can then be expressed as:

$$\begin{aligned} m &= [c \cdot w]_d \bmod 2 \\ &= \left[\sum^S c x_i R^{l_i}\right]_d \bmod 2 \\ &= \left[\sum^S c x_i R^{l_i}\right]_2 - \left[\left\lfloor (\sum^S c x_i R^{l_i})/d \right\rfloor \cdot d\right]_2 \\ &= \left[\sum^S c x_i R^{l_i}\right]_2 - \left[\left\lfloor \sum^S (c x_i R^{l_i}/d) \right\rfloor\right]_2 . \end{aligned} \tag{4.4}$$

The Recrypt process can then be divided into two parts. First we compute the sum of $cx_iR^{l_i}$ for each "block" $i$. To further optimize this process, encode $l_i$ to a $0-1$ vector $\{\eta_1^{(i)}, \eta_2^{(i)}, \ldots, \eta_n^{(i)}\}$ where only two elements are "1" and all other elements are "0"s. Suppose the two positions are labeled as $a$ and $b$. We write $l(a, b)$ to refer to the corresponding value of $l$. Alternatively we can obtain $cx_iR^{l_i}$ from

$$cx_i R^{l_i} = \sum_a \eta_a^{(i)} \sum_b \eta_b^{(i)} cx_i R^{l(a,b)} . \tag{4.5}$$

Obviously, only when $\eta_a^{(i)}$ and $\eta_b^{(i)}$ are both "1", the corresponding $cx_iR^{l(a,b)}$ is selected. In addition, if we encode $l$ in a way that each iteration only increases it by 1, the next factor $cx_iR^{l(a,b)}$ can be easily computed by multiplying $R$ to the result of the previous computation.

After applying these modifications, all operations involved in this formulation of decryption become bit operations realizable by sufficiently shallow circuits. Thus we can evaluate this process homomorphically. The parameters $\eta_i$ are stored in encrypted form and incorporated into the public key.

### 4.1.2 Fast Multiplications on GPUs

### 4.1.3 The Schönhage-Strassen FFT Multiplication

Large integer multiplication is the most time consuming operation in the FHE primitives. Therefore, it becomes the main target for acceleration. In [51], Strassen described a multiplication algorithm based on Fast Fourier Transform (FFT), which offers a good solution for effectively parallel computation of the large-number multiplication as shown in Fig. 4.1. The algorithm uses Fast Fourier transforms in rings with $2^{2^n}+1$ elements, i.e. a specialized number theoretic transform. Briefly, the Strassen FFT algorithm can be summarized as follows:

1. Break large numbers $A$ and $B$ into a series of words $a(n)$ and $b(n)$ given a base $b$, and compute the FFT of the $A$ and $B$ series by treating each word as an sample in the time domain.

2. Multiply the FFT results, component by component: set $C[i] = FFT(A)[i] * FFT(B)[i]$.

3. Compute the inverse fast Fourier transform: set $c(n) = IFFT(C)$.

4. Resolve the carries: when $c[i] \geq b$, set $c[i+1] = c[i+1] + (c[i] \operatorname{div} b)$, and $c[i] = c[i] \bmod b$.

### 4.1.4 Emmart and Weems' Approach

In [52], Emmart and Weems implemented the Strassen FFT based multiplication algorithm on GPUs. Specifically, they performed the FFT operation in finite field $Z/pZ$ with a prime $p$ to make the FFT exact. In fact, they chose the $p = 0\text{xFFFFFFFF00000001}$ from a special family of prime numbers which are called

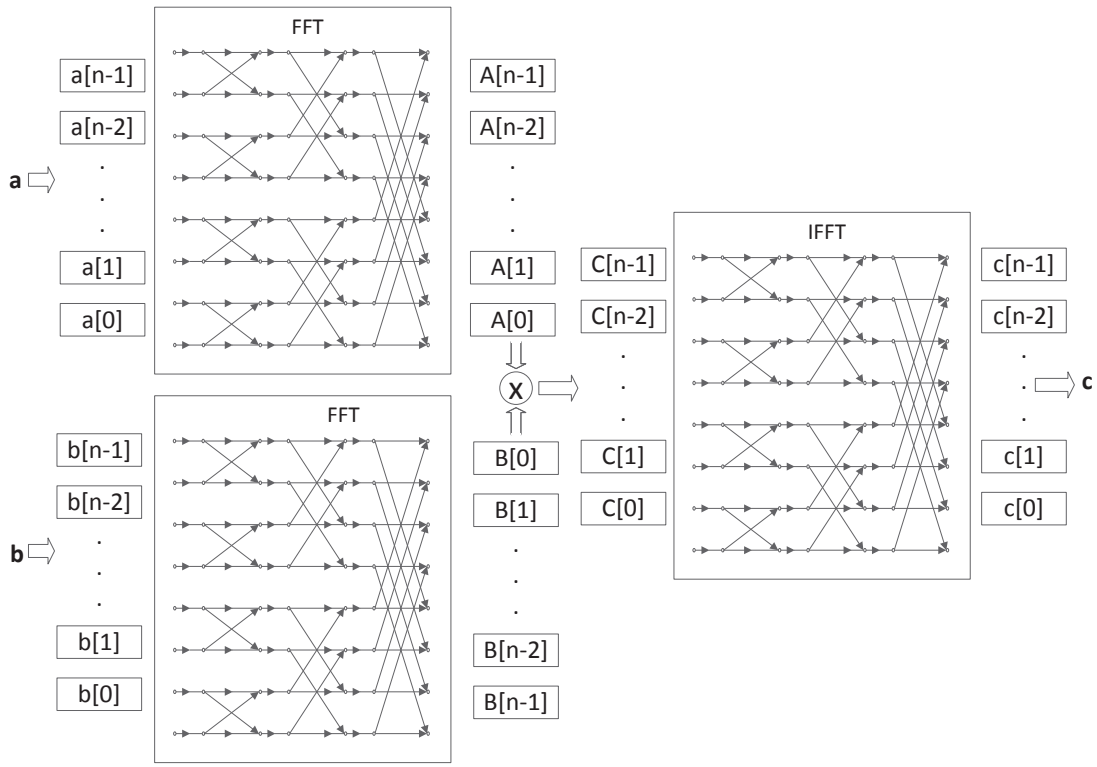


Figure 4.1: Strassen's FFT Multiplication Algorithm

Solinas Primes [53]. Solinas Primes support high efficiency modulo computations and this $p$ especially is ideal for 32-bit processors, which has also been incorporated into the latest GPUs. In addition, an improved version of Bailey's FFT technique [54] is employed to compute the large size FFT. The performance of the final implementation is very promising. For the operands up to $16,320K$ bits, it shows a speedup factor of up to 16.7 when comparison with multiplication on the CPUs of the same technology generation.

We follow the implementation in [52] and test it on the GPU. As we can see from Table 4.1, the actual speedup factors are slightly different from [52]. Nevertheless, it is a significant speedup over the implementations achieved on CPUs. Therefore, we employ this specific instance of the Strassen FFT based multiplication algorithm in the FHE implementation.

We note however, the optimizations we later introduce virtually eliminate the need for FFT conversions except at the very beginning or the very end of the computation chains. The only exception is the decryption primitive which is implemented using a single modular multiplication. Therefore, while still necessary, the efficiency of the FFT operation has a negligible impact on the overall performance of encryption and recryption.

| Size in K bits | On CPU | On GPU | Speedup |
|---|---|---|---|
| 1024 x 1024 | 8.5 ms | 0.583 ms | 14.6 |
| 2048 x 2048 | 15.1 ms | 1.085 ms | 13.9 |
| 4096 x 4096 | 30.4 ms | 2.351 ms | 12.9 |
| 8192 x 8192 | 63.1 ms | 4.850 ms | 13.0 |
| 16384 x 16384 | 137.3 ms | 8.835 ms | 16.7 |

Table 4.1: Performance comparison of multiplication on CPUs vs. GPUs

### 4.1.5 Modular Multiplication

Efficient modular multiplication is crucial for the decryption primitive. The other primitives only use modular reduction at the very end of the computations only once. Many cryptographic software implementations employ the Montgomery multiplication algorithm, cf. [55, 56]. Montgomery multiplication replaces costly trial divisions with additional multiplications. Unfortunately, the interleaved versions of the Montgomery multiplication algorithm generates long carry chains with little instruction-level parallelism. For the same reason, it is hard to realize Montgomery's algorithm on parallel computing friendly GPUs. For example, a Montgomery multiplication implementation on GeForce 9800GX2 card was presented In [57]. The speedup factor of GPU decreased from 2.6 to 0.6 when the operand size increases from 160-bit to 384-bit, which showed little speedup if any can be achieved with large operand sizes. In addition, the underlying large integer multiplication algo-

rithm we use is FFT based and optimized for very large numbers. Therefore, there does not seem to be any easy way to break it into smaller pieces. In conclusion, we implement modular multiplications without integrating the multiplication and reduction steps, but instead by executing them in sequence.

### Modular Reduction

The most popular algorithms for modular reduction are the Montgomery reduction [58] and the Barrett reduction algorithms [59]. As mentioned earlier, the interleaved Montgomery reduction algorithm cannot exploit the parallel processing on GPUs. The Barrett approach has a simpler structure and thus lends itself better for further optimizations. Therefore, we select the Barrett method to realize modular reductions.

Given two positive integers $t$ and $M$, the Barrett modular reduction approach computes $r = t \bmod M$. A version of Barrett's reduction algorithm is shown in Figure 4.2.

```
1: procedure BARRETT(t, M)          ▷ Output: r = t mod M
2:     q ← 2⌈log2(M)⌉                    ▷ Precomputation
3:     μ ← ⌊2^q / M⌋                     ▷ Precomputation
4:     r ← t − M ⌊tμ/2^q⌋
5:     while r ≥ M do
6:         r ← r − M
7:     end while
8:     return r                                ▷ r = t mod M
9: end procedure
```

Figure 4.2: Barret reduction algorithm

Note that code from line 5 to line 7 is a loop. However, it can be shown that the initial $r$ for this loop is smaller than $3M - 1$. Therefore, this loop can finish quickly. In addition, the value $\mu = \lfloor \frac{2^q}{M} \rfloor$ ($q = 2\lceil \log_2(M) \rceil$) can be pre-computed to speed up

the process. If multiple reductions are to be computed with the same modulus $M$. Then this value can be reused for all reductions, which is exactly the case we have.

In addition, it would be advantageous to apply truncations only at multiples of the word size $w$ of the multiplier hardware (usually 32 bits) rather than at the original bit positions. In this case, we require $q$ to be a multiple of the word size $w$. With this approach, the division by $2^q$ can be easily implemented by discarding the least significant $q/w$ words.

### 4.1.6 Optimization of FHE Primitives

The FHE algorithm consists of four primitives: KeyGen, Encrypt, Decrypt and Recrypt. The KeyGen is only called once during the setup phase. Since keys are generated once and then preloaded to the GPU, the speed of KeyGen is not as important. Therefore we focus our attention to optimizing the other three primitives.

For the Decrypt primitive, we perform the computation as in 4.3. The flow is shown in Fig. 4.3. Obviously, the time spent in the primitive is equivalent to the time it takes to compute a single modular multiplication with large operands. Applying the FFT based Strassen algorithm and Barrett reduction, which we discussed earlier, yields significant speedup for the Decrypt operation.

### 4.1.7 Optimizing Encrypt

To realize the Encrypt primitive, we need to evaluate a degree-$(n-1)$ polynomial $u$ at point $r$. In [29], a recursive approach for evaluating the 0-1 polynomial $u$ of degree $(n-1)$ at root $r$ modulo $d$. The polynomial $u(x) = \sum_{i=0}^{n-1} u_i r^i$ is split into a "bottom half" $u^{bot}(r) = \sum_{i=0}^{n/2-1} u_i r^i$ and a "top half" $u^{top}(r) = \sum_{i=0}^{n/2-1} u_{i+d/2} r^i$. Then $y = r^{n/2} u^{top}(r) + u^{bot}(r)$ can be computed. The same procedure repeats until the remaining degree is small enough to be computed directly.

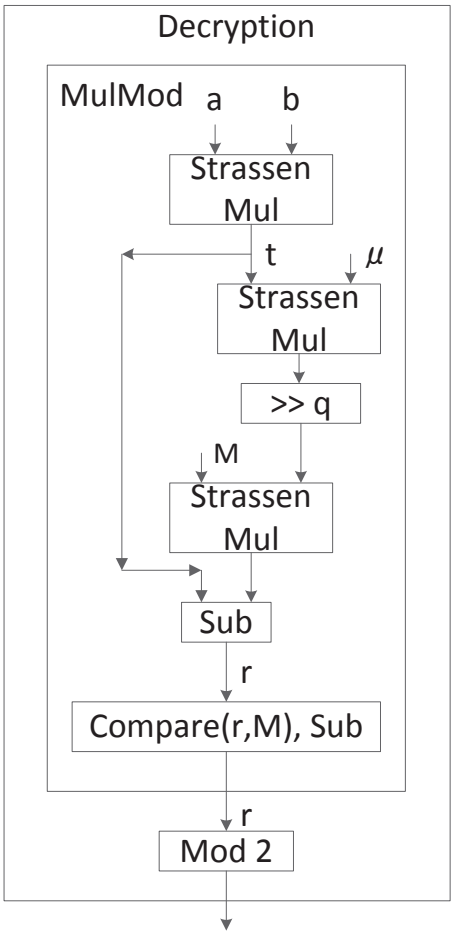


Figure 4.3: Decryption Procedure

In our implementation, to fully exploit the power of pre-computation, we use a direct approach for polynomial evaluations. Specifically, we apply the sliding window technique to compute the polynomial. Suppose the window size is $w$ and we need $t = n/w$ windows, we compute:

$$\sum(u_i r^i) = \sum_{j=0}^{t-1} [r^{w \cdot j} \cdot \sum_{i=0}^{w-1} (u_{i+wj} r^i)]. \tag{4.6}$$

Here all additions and multiplications are evaluated modulo $d$. After organizing the computation as described above, we can introduce pre-computation to speed up the process. As $r$ is determined during KeyGen and therefore known apriori, the $r^i$,

$i = 0, 1, \ldots, w$ values can be precomputed. In order to further reduce the overhead caused by the relatively slow communication between the CPU and the GPU, these precomputed values can be preloaded into GPU memory before the Encrypt process starts. Clearly, larger window size $w$ leads to fewer multiplications but an increased memory requirement. Hence, we have a trade-off between speed and memory use. In addition, as mentioned in previous sections, the majority of the coefficients of $u$ are zeros. It is possible that all the coefficients a window are zeros. In this case, we can skip the multiplication to further speed up the process.

In addition, as we use the FFT based algorithm to compute the multiplications, these pre-computed values can also be saved in FFT form. Since the FFT form is linear, we can directly evaluate additions in FFT domain. Therefore, the whole computation before the final reduction can be performed in FFT domain:

$$\sum (u_i r^i) = IFFT(\sum_{j=0}^{31} [R^{64 \cdot j} \cdot \sum_{i=0}^{63} (u_{i+64j} R^i)]) \bmod d, \tag{4.7}$$

where $R^i$ is the precomputed FFT form of corresponding $r^i$. With this reformulation we eliminated almost all of the costly FFTs and IFFTs and modular reductions. Also as a side-benefit of staying in the FFT domain, carry propagations among words normally performed during addition operations are eliminated, which also contributes to the speed up.

In our implementation with dimension $n = 2048$, we choose the window size as $w = 64$. With this parameters, the CPU implementation of Encrypt runs in 1.08 seconds while our implementation on GPU the run time is significantly reduced to only 6.2 ms.

### 4.1.8 Implementing Recrypt

The Recrypt primitive is significantly more complicated. As mentioned earlier, Recrypt can be divided into two steps: processing of $S$ blocks and the computation of their sum. In the first step, the most time-consuming computation is as follows

$$cx_i R^{l_i} = \sum_a \eta_a^{(i)} \sum_b \eta_b^{(i)} cx_i R^{l(a,b)} \ . \tag{4.8}$$

Here $\eta_i$ is part of the public key. If we encode the $l$ in a proper way such that each iteration it only increases by one, the next factor $cx_i R^{l(a,b)}$ can be easily computed by multiplying $R$ with the result of the previous iteration. Here we refer to $cx_i R^{l(a,b)}$ as the *factor* for each iteration. In each iteration, we update $factor \cdot R \bmod d$ and determine whether we should sum $\eta_b$ or not. Since in this process $R$ is a small constant, the computation may even be performed on the CPU without any noticeable loss of efficiency in the overall scheme. Therefore, the CPU is used to compute the new *factor* value while the GPU is busy computing the additions from previous iteration. This approach allows us to run the CPU and the GPU concurrently and therefore harnessing the full computational power of the overall system.

The constants used in Recrypt are part of the public key. They can be precomputed to further speed up the computation. Similar to Encrypt, the public keys can be pre-loaded into the GPU memory to eliminate the latency incurred in CPU-GPU communications. In our implementation we targeted the small (security) setting, where the public key is about 140MB. The public key can perfectly fit into the GPU memory of the latest graphic cards. In fact, the public key will even fit into the GPU memory in the large setting, whose public key is about 2.25GB [29].

Furthermore, the majority of the computation in the Recrypt can also be represented as some "add-mul-add" chain as in the Encrypt. Therefore, the similar opti-

mizations can be applied. The computation before the reduction can be performed in the FFT domain, reducing the number of expensive FFT and IFFT operations significantly. However, this optimization will also cause growth in public key size and make it impossible to store the whole public key in the GPU memory in the high dimension case. Fortunately, with such high dimension settings, the computation time is long enough to dwarf this extra communication overhead.

### 4.1.9 Implementation Results

We realized the Encrypt, Decrypt and Recrypt primitives of the Gentry-Halevi FHE scheme with the proposed optimizations on a machine with Intel Core i7 3770K running at 3.5 GHz with 8 GB RAM and a NVIDIA GTX 690 running at 1.02 GHz with 4GB memory. Only one GPU is used in this implementation. Shoup's NTL library [60] is used for high-level numeric operations and GNU's GMP library [61] for the underlying integer arithmetic operations. A modified version of the code from [52] is used to perform the Strassen FFT multiplication on GPU.

We implemented the scheme with small and medium parameter setting, respectively dimension 2,048 and 8192. We also recompiled the code provided by Gentry and Halevi for the CPU implementation [29] on the same computer for comparison. The performance results are summarized in Table 4.2.

As we can see clearly from the table, our implementation for the small case is about 174, 7.6 and 13.5 times faster than the original Gentry-Halevi code for encryption, decryption and recryption, respectively [29]. The impressive speedup of encryption is due to the fact that encryption benefits significantly from precomputation. For the medium case with dimension 8192, we also achieved a speed up of 442, 9.7 and 11.7. Note that the encryption process enjoys even more speedup as the dimension grows.

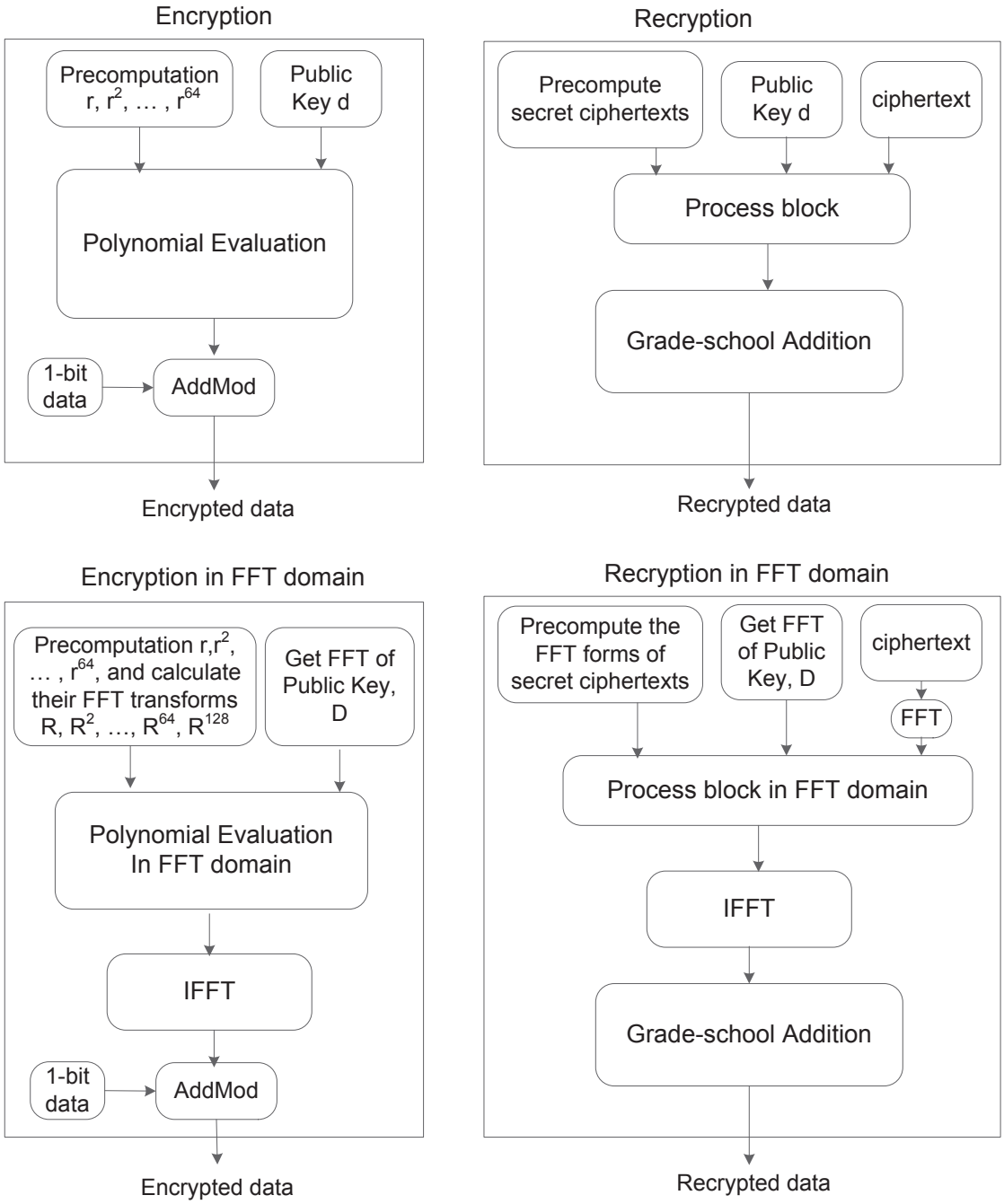


Figure 4.4: Proposed Encrypt and Decrypt in the FFT domain

To explore the effect of our optimizations, we also broke down and evaluated the time consumption for the Recrypt primitive. Use the small case as example, if we look into the 1.32 seconds of time it takes to compute the Recrypt, we discover

| Small Setting: Dimension 2048 | | | |
|---|---|---|---|
| Operation | CPU | GPU | Speedup |
| Encrypt | 1.08 sec | 6.2 msec | 174 |
| Decrypt | 14 msec | 1.84 msec | 7.6 |
| Recrypt | 17.8 sec | 1.32 sec | 13.5 |
| Medium Setting: Dimension 8192 | | | |
| Operation | CPU | GPU | Speedup |
| Encrypt | 10.6 sec | 24 msec | 442 |
| Decrypt | 70 msec | 7.2 msec | 9.7 |
| Recrypt | 96.3 sec | 8.4 sec | 11.5 |

Table 4.2: Performance of FHE primitives with proposed optimizations

that it takes about 0.87 seconds for processing blocks and 0.46 second for grade-school addition. Further inspection of the block processing part reveals that the GPU multiplications and additions take about 0.54 second. In the meantime, it takes the CPU about 0.6 second to compute the *factor*. Clearly, the sum of this two latencies amounts to more than 0.87 seconds. This is due to the fact that the CPU and the GPU are working in parallel. For comparison, if we implement the Recrypt by only realizing the multiplications and additions via GPU, i.e. without out our further optimization, the same break down shows the multiplications and additions will take about 2.3 seconds in total. This shows that our optimization speedup the block processing part by a factor of 2.64.

For the medium case, the GPU memory is not large enough to hold the whole public key. Therefore, the keys are loaded when required, which introduces an overhead of about 0.6 seconds for Recrypt. However, as the computation for the medium case consumes much more time, the impact of this overhead to the overall performance is limited.

**Optimizing the grade-school additions** The grade-school addition part is relatively light-weighted and therefore of less important for optimization. Thus, for the previous discussed implementation, no optimization is allied to it other than

employing GPU multiplications. However, as the block processing becomes faster, the overhead of the the grade-school addition part generates a new bottle neck. Therefore, we further optimized the grade-school addition part by employing "three to two" adders. The resulting grade-school addition module only requires about 0.1 seconds to finish the additions. After applying this optimization, the Recrypt process can be evaluated in less than 1 second.

## 4.2 Impelementing the LTV FHE Scheme

### 4.2.1 Lopez-Tromer-Vaikuntanathan FHE

In [28], Lopez-Alt, Tromer and Vaikuntanathan propose a multi-key homomorphic encryption scheme (LTV-FHE) based on a modified NTRU [39] scheme previously introduced by Stehle and Steinfeld [40]. Striving to obtain implementations that are truly practical, in this section we adapt their presentation to arrive at a streamlined single-key formulation.

We require the ability to sample from a probability distribution $\chi$ on $B$-bounded polynomials in $R_q := \mathbb{Z}_q[x]/(x^n + 1)$ where a polynomial is "$B$-bounded" if all of its coefficients lie in $[-B, B]$. For example, we can sample each coefficient from a discrete Gaussian with mean 0 and discard samples outside the desired range. The basic idea of encryption is to cover each message $m$ with two masks, one which can be removed with the private key and the other which vanishes upon reduction modulo two. Simply put, if $f$ is the private key satisfying $f \equiv 1 \pmod 2$, we generate a random polynomial $g$ from $\chi$ and publish $h = 2gf^{-1}$ as our public key. A bit $m$ is then encrypted as $c = hs + 2e + m$ where $s$ and $e$ are random polynomials also sampled from distribution $\chi$. The decryption is simply multiply the cipher to the private key: $cf = 2gsf^{-1}f + 2ef + mf \equiv 1 \pmod 2$.

With a single key in use, we immediately gain some economy over the LTV-FHE scheme. Since the norm of a sum of two polynomials is bounded by the sum of their respective norms, we may implement an XOR gate by simply adding the two ciphertexts encrypting the two inputs. Each AND gate in the circuit incurs a modulus reduction step that has the effect of reducing the magnitude of the noise in the output of the gate. As well, a Relinearization step is employed to effectively re-encrypt the output of previous gates under the new encryption modulus. So we have a decreasing sequence of odd prime moduli $q_0 > q_1 > \cdots > q_d$ where $d$ is the depth of the decryption/application circuit. In this way, the key (public and evaluation keys) can become quite large and it remains a practical challenge to manage the size of this data and handle it efficiently. As well, the parameters $n$ and the $q_i$ must be chosen very carefully to balance efficiency and wraparound control while still maintaining a trusted level of security in the face of known attacks.

- KeyGen: We choose a decreasing sequence of primes $q_0 > q_1 > \cdots > q_d$ and a polynomial $\phi(x) = x^n + 1$. For each $i$, we sample $u^{(i)}$ and $g^{(i)}$ from distribution $\chi$, set $f^{(i)} = 2u^{(i)} + 1$ and $h^{(i)} = 2g^{(i)} \left(f^{(i)}\right)^{-1}$ in ring $R_{q_i} = \mathbb{Z}_{q_i}[x]/\langle\phi(x)\rangle$. (If $f^{(i)}$ is not invertible in this ring, re-sample.) We then sample, for $i = 0, \ldots, d$ and for $\tau = 0, \ldots, \lfloor \log q_i \rfloor$, $s_\tau^{(i)}$ and $e_\tau^{(i)}$ from $\chi$ and publish evaluation key $\left\{\zeta_\tau^{(i)}(x)\right\}_{tau}^{i}$ where $\zeta_\tau^{(i)}(x) = h^{(i)} s_\tau^{(i)} + 2e_\tau^{(i)} + 2^\tau \left(f^{(i-1)}\right)^2$ in $R_{q_{i-1}}$.

- Encrypt: To encrypt a bit $b \in \{0, 1\}$ with a public key $(h^{(0)}, q_0)$, Encrypt first generates random samples $s$ and $e$ from $\chi$ and sets $c^{(0)} = h^{(0)} s + 2e + b$, a polynomial in $R_{q_0}$.

- Decrypt: To decrypt the ciphertext $c$ with the corresponding private key $f^{(i)}$, Decrypt multiplies the ciphertext and the private key in $R_{q_i}$ then compute the message by modulo two: $m = c^{(i)} f^{(i)} \pmod 2$

- Eval: We assume we are computing a leveled circuit with gates alternating between XOR and AND. Arithmetic operations are performed directly on ciphertexts as follows: Suppose $c_1^{(0)} = \mathsf{Encrypt}(b_1)$ and $c_2^{(0)} = \mathsf{Encrypt}(b_2)$. Then XOR is effected by simply adding ciphertexts: $\mathsf{Encrypt}(b_1 + b_2) = c_1^{(0)} + c_2^{(0)}$. Polynomial multiplication incurs a much greater growth in the noise, so each multiplication step is followed by a modulus switching. First, we compute $\tilde{c}^{(0)}(x) = c_1^{(0)} \cdot c_2^{(0)} \pmod{\phi(x)}$ and then perform Relinearization, as described below, to obtain $\tilde{c}^{(1)}(x)$ followed by modulus switching $\mathsf{Encrypt}(b_1 \cdot b_2) = \lfloor \frac{q_1}{q_0} \tilde{c}^{(1)}(x) \rceil_2$ where the subscript 2 on the rounding operator indicates that we round up or down in order to make all coefficients equal modulo 2. The same process hold for evaluating with $i$th level ciphertexts, e.g. computing $\tilde{c}^{(i)}(x)$ from $c_1^{(i-1)}$ and $c_2^{(i-1)}$.

- Relinearization: We will show the general process that computing $\tilde{c}^{(i)}(x)$ from $\tilde{c}^{(i-1)}(x)$. We expand $\tilde{c}^{(i-1)}(x)$ as an integer linear combination of 1-bounded polynomials $\tilde{c}^{(i-1)}(x) = \sum_\tau 2^\tau \tilde{c}_\tau^{(i-1)}(x)$ where $\tilde{c}_\tau^{(i-1)}(x)$ takes its coefficients from $\{0, 1\}$. We then define $\tilde{c}^{(i)}(x) = \sum_\tau \zeta_\tau^{(i)}(x) \tilde{c}_\tau^{(i-1)}(x)$ in $R_{q_i}$.

To see why this works, observe that simple substitution gives us

$$\begin{aligned}
\tilde{c}^{(i)}(x) &= h^{(i)}(x) \left[ \sum_{\tau=0}^{\lfloor \log q_i \rfloor} s_\tau^{(i)}(x) \tilde{c}_\tau^{(i-1)}(x) \right] + 2 \left[ \sum_{\tau=0}^{\lfloor \log q_i \rfloor} e_\tau^{(i)}(x) \tilde{c}_\tau^{(i-1)}(x) \right] \\
&\quad + \left[ f^{(i-1)} \right]^2 \sum_{\tau=0}^{\lfloor \log q_i \rfloor} 2^\tau \tilde{c}_\tau^{(i-1)}(x) \\
&= h^{(i)}(x) S(x) + 2E(x) + \left[ f^{(i-1)} \right]^2 \tilde{c}^{(i-1)}(x) \\
&= h^{(i)}(x) S(x) + 2E(x) + \left[ f^{(i-1)} c_1^{(i-1)}(x) \right] \left[ f^{(i-1)} c_2^{(i-1)}(x) \right] \\
&= h^{(i)}(x) S(x) + 2E'(x) + m_1 m_2
\end{aligned}$$

modulo $q_{i-1}$ for some pseudorandom polynomials $S(x)$ and $E'(x)$. This ensures that the output of each gate takes the form of a valid fresh encryption of the product $m_1 m_2$ of plaintexts.

### 4.2.2 Parameter Selection in the LTV-FHE

A significant – yet unresolved – problem holding researchers back from implementing and improving LTV-FHE is parameter selection. In the original reference [28] the security analysis is mostly given in asymptotic by reduction to the related learning with error (LWE) problem [40]. In this section we summarize the results of our preliminary work on parameter selection.

There are three parameters that will affect security, the dimension N, the coefficient size $|q|$ and the error bound $B$. The $|q|$ will mainly affect the number of multiplications the scheme supports. The value of it largely depends on the complexity of the application. e.g. a full ten round AES will require at least $|q| = 1024$. For our implementation, we select $|q| = 256$ for small scale experiments and $|q| = 1024$ for AES implementation. The error in the LTV scheme must provide enough randomness to defend against brute force attack. If the dimension is high enough, even $B = 1$ will be enough to cover brute force attacks. If we assume the RLWE reduction and consider the lattice attacks, then the smaller the B the lower the security level. Therefore we assume a small B, i.e. $B = 1$, so that the corresponding dimension requirement will provide enough security for any larger $B$. Given the selection of $|q|$ and $B$, we will discuss the relation between the dimension $N$ and the security level.

The LTV-FHE scheme is developed from a modified version of NTRU [39] proposed by Stehle and Steinfeld [40], which can be reduced to the Ring-LWE (RLWE) problem. Specifically, the security reduction is obtained through a hybrid argument:

1. Recall that for the LTV-FHE scheme, the public key is of the form $h = 2gf^{-1}$

where $g, f$ chosen from a Gaussian distribution $D$ where $f$ is kept secret. The DSPR problem is to distinguish polynomials of the form $h = 2gf^{-1}$ from samples $h'$ picked uniformly at random from the ring $R_q$. If the DSPR problem is hard, we can replace $h = 2gf^{-1}$ by some uniformly sampled $h'$.

2. Once $h$ is replaced by $h'$, the encryption $c = h's + 2e + m$ takes the form of the RLWE problem and we can replace the challenge cipher by $c' = u + m$ with a uniformly sampled $u$, thereby ensuring security.

Stehle and Steinfeld have shown that the DSPR problem is hard even for unbounded adversaries with their parameter selection. However, the new LTV-FHE scheme will require different parameters to support homomorphic evaluation. The impact of the new parameter settings to the security level is largely unknown and requires careful research. However, even if we assume that the DSPR problem is hard for typical LTV-FHE parameter selection, concrete parameters are still hard to chose. The RLWE problem is still relatively new and lacks thorough security analysis. A common approach is to *assume* that RLWE follows the same behavior as the LWE problem [30]. Under this assumption only, we can select parameters. If we omit the noise, given the prime number $q$ and $k$-bit security level, the dimension is bounded as in [30] as $N \leq \log(q)(k + 110)/7.2$ .

For example, given a 256-bit prime $q$, an 80-bit security level will require dimension $N = 6756$. However, this large estimate is actually an upper bound and assumes that the LTV-FHE scheme can be reduced to the RLWE problem. It is not clear whether the reverse is true, i.e. whether attacks against the RLWE problem apply to the LTV-FHE scheme. For instance, the standard attack on the LWE problem requires many samples generated with the *same* secret $s$. However, in the LTV-FHE scheme, the corresponding samples are ciphertexts of the form

$c = h's + 2e + m$, where the $s$ polynomials are randomly generated and independent. This difference alone suggests that standard attacks against LWE problems cannot be directly applied to the LTV-FHE scheme. However, as a modified version of NTRU, the LTV-FHE scheme suffers from the same attack as the original NTRU. We can follow a similar approach as in the original NTRU paper [39] to find the secret $f$: Consider the following $2N$ by $2N$ "NTRU" lattice where the $h_i$ are the coefficients of $h = 2gf^{-1}$. Let $L$ be the lattice generated by the matrix. Clearly, $L$ contains the vector $a = (f, 2g)$ which is small. Now the problem is transformed to searching for short lattice vectors.

$$\left(\begin{array}{c|cccc} & h_0 & h_1 & \cdots & h_{N-1} \\ & -h_{N-1} & h_0 & \cdots & h_{N-2} \\ \mathbf{I} & \vdots & \vdots & \ddots & \vdots \\ & -h_1 & -h_2 & \cdots & h_0 \\ \hline \mathbf{0} & \multicolumn{4}{c}{q\mathbf{I}} \end{array}\right)$$

In [62], Gama and Nguyen proposed a useful approach to estimate the hardness of the SVP in an $N$-dimensional lattice $L$ using the Hermite factor $\delta^N = ||b_1||/\det(L)^{1/N}$ where $||b_1||$ is the length of the shortest vector or the length of the vector for which we are searching. The authors also estimate that, for larger dimensional lattices, a factor $\delta^N \leq 1.01^N$ would be the feasibility limit for current lattice reduction algorithms. In [63], Lindner and Peikert gave further experimental results regarding the relation between the Hermite factor and the break time as $t(\delta) := \log(T(\delta)) = 1.8/\log(\delta) - 110$. For instance, for $\delta^N = 1.0066^N$, we need about $2^{80}$ seconds on the platform in [63].

For the LTV-FHE scheme, we can estimate the $\delta$ of the NTRU lattice and thus the time required to find the shortest vector. Clearly, the NTRU lattice has

dimension $2N$ and volume $q^N$. However, the desired level of approximation, i.e. the desired $||b_1||$, is unclear. In [62], Gama and Nguyen use $q$ as the desired level for the original NTRU. However, for the much larger $q$ used in the LTV-FHE scheme, this estimate won't apply. In particular, Minkowski tells us that $L$ has a nonzero vector of length at most $\det(L)^{1/t}\sqrt{t}$ where $t$ is the dimension. There will be exponentially (in $t$) many vectors of length $poly(t)\det(L)^{1/t}$. In our case, this size would be $(q^N)^{1/2N}poly(N) = poly(N)\sqrt{q}$. Since $N \approx \log(q)$ for the LTV-FHE scheme, we can see that the approximate level $q$ will tend to be obscured by these exponentially many vectors. In other word, the analysis in [62] for NTRU does not apply to the LTV scheme. Therefore, we choose to follow the experimentation approach in [64].

We ran a large number of experiments to determine the time required to search for this shortest vector for a number of dimensions following the same approach as in [64]. We generated different LTV keys with with coefficient size $|q| = 256$[1] and different dimensions, constructed the NTRU lattices as above and used the Block-Korkin-Zolotarev (BKZ) [65] lattice reduction functions provided in Shoup's NTL Library [60] to search for the shortest vectors. More specifically, we set Schnorr's pruning constant to 0 as experiments did not show that it would improve the running time. Then we set the LLL constant to 0.99 and ran the program with different block sizes. We started with a small block size for each run and kept increasing the block size until we found the target vector. The observed results showed the same pattern as in [64]. The block size needed increases roughly linearly with the dimension, and the running time grows exponentially with the block size.

After getting the time required to break different dimension instance of LTV keys, we can estimate the time required for higher dimensions and estimate the

[1]Clearly, the larger $|q|$, the larger the required dimension will be to keep the LTV scheme secure. However, other than lower the number of iterations required to break the system, larger $|q|$ will also slow down the reduction algorithm significantly. Experiments show that for $|q| = 1024$, the actual time required to break the LTV scheme for given dimension is close to the $|q| = 256$ case.

corresponding bit security levels using the approach in [66]. Table 4.2.2 summarizes the results and the estimated bit security level for higher dimensions. The results are collected on a latest 3.5 GHz Intel i7 machine using the approaches discussed above. The newly proposed BKZ2.0 algorithm in [66] will affect our parameter selection. Unfortunately, we do not have access to it and cannot run it to determine the performance. However, we may estimate the impact on the security level according to the timing results reported in [66]. The dimension $N = 768$ will be appropriate to keep enough security margin against BKZ2.0. Following the time to bit conversion approach of BKZ2.0 we estimated the security level as 90-bits with BKZ (80-bits against BKZ2.0).

| Dimension | Bit Security |
|---|---|
| 140 | 42 |
| 160 | 46 |
| 256 | 52 (est.) |
| 384 | 63 (est.) |
| 512 | 74 (est.) |
| 768 | 90 (est.) |

Table 4.3: Security level of the LTV scheme; a small subset of our experiments and estimates

### 4.2.3 LTV-FHE in Software

To form a baseline for performance evaluation, we implemented the LTV primitives using Shoup's NTL library [60]. The LTV FHE algorithm consists of four functions: KeyGen, Encrypt, Decrypt and Eval. The KeyGen is only called once during the setup phase. Therefore the speed of KeyGen is not as important. In addition, the computation of Encrypt and Decrypt is quite simple. Experimental results also show that Encrypt and Decrypt have satisfactory performance even without any optimization other than those already included in the NTL library. Thus, we focus

on the optimization of the Eval primitive.

The Eval operation can be divided into three steps: the actual evaluation (addition or multiplication), Relinearization and modulus switching. Among these, the actual evaluation and the modulus switching operations require negligible time. The most computation intensive operation in the Eval is Relinearization. Recall that the Relinearization process of the LTV scheme involves calculating $\tilde{c}^{(i)}(x) = \sum_\tau \zeta_\tau^{(i)}(x)\tilde{c}_\tau^{(i-1)}(x)$ where $\tilde{c}_\tau^{(i-1)}(x)$ takes its coefficients from $\{0,1\}$ and $\zeta_\tau^{(i)}$ is part of the public evaluation key. In other word, we are computing the sum of product of polynomials for the Relinearization. In addition, since $\tilde{c}_\tau^{(i-1)}(x)$ takes its coefficients from $\{0,1\}$, the product can also be easily calculated via additions. Clearly, that the performance of the Relinearization process heavily relies on the efficiency of polynomial additions.

**Further Optimizations**. Relinearization process can be further improved via pre-computation. Recall that the Relinearization computation is expressed by $\tilde{c}^{(i)}(x) = \sum_\tau \zeta_\tau^{(i)}(x)\tilde{c}_\tau^{(i-1)}(x)$ where $\zeta_\tau^{(i)}(x) = 2^\tau f_i$ and $\tilde{c}_\tau^{(i-1)}(x)$ is the corresponding bits of the input ciphertext. However, the same computation can also be carried out more than one bit a time. i.e. let $\zeta_\tau^{(i)}(x) = 2^{k\tau} f_i$ and $\tilde{c}_\tau^{(i-1)}(x)$ take its coefficients from $\{0, 1, ..., 2^k - 1\}$. In this way, the Relinearization can be done in $1/k$ iterations. However, each iteration will then require multiplication of the $\zeta_\tau$ and the corresponding $2^k$ bit coefficient instead of one bit coefficient, which will slow down the process. To solve this problem, we pre-compute all possible multiples of $\zeta_\tau$ so that the multiplication can be computed via table lookup. In this way, we may speed up the Relinearization process at the expense of increased storage space. Table 4.4 shows the speedup factor and corresponding evaluation key sizes for various selections of bits per iteration. Note that, although we need only $1/k$-th iterations if we process $k$ bits a time, the speedup is not $k$-fold since, if we process one bit per iteration,

half of the iterations can be skipped due to zero coefficients of $\tilde{c}_\tau^{(i-1)}(x)$ . However, if we process $k$ bits per iteration, only the fraction of $1/2^k$ of the iterations can be skipped. In addition to the pre-computation, we also wrote our own code for

| Bits per Iteration | Speedup Factor | Size of Evaluation Key |
|---|---|---|
| 1 | 1 | 4 MBytes |
| 2 | 1.33 | 6 MBytes |
| 3 | 1.71 | 9.3 MBytes |
| 4 | 2.13 | 15 MBytes |

Table 4.4: Speed-Space Trade off

polynomial additions to eliminate the overhead from the NTL library lower level functions and to improve the memory accesses. After applying all these optimizations, we achieved more than ten times speed up over the direct implementation using NTL. The performance measured on a 3.5 GHz Intel i7 machine with 8 GB RAM is shown in Table 4.5. We determined that a pre-computation level of 4-bits per iteration provides a good speedup while still maintaining a reasonable evaluation key size. As shown in Table 4.5, we achieved millisecond level Encrypt and Decrypt and the Relinearization takes about 38 msecs. Compared to the 4.2 secs Recrypt time reported in [41] for the Gentry-Halevi scheme, this result shows much promising efficiency[2]. It will also be interesting to compare our implementation to the FHE implementation in [30]. Therefore, we estimate the time required for a homomorphic AES encryption using the LTV FHE scheme.

**Homomorphic AES Encryption.** Assume that the messages are encrypted bit by bit using the LTV FHE scheme, we want to homomorphically compute the AES rounds. Recall that for an AES round, we have four steps: SubBytes, ShiftRows,

[2]As an operation that is required after every multiplication, the Relinearization process in the LTV FHE scheme plays a similar role as the Recrypt in the Gentry-Halevi scheme. Therefore, the performance of these two operations offers a reasonable reflection of the efficiency of the two schemes.

| Operation | $N = 512$ $\lvert q\rvert = 256$ | $N = 512$ $\lvert q\rvert = 1024$ | $N = 768$ $\lvert q\rvert = 1024$ |
|---|---|---|---|
| Encrypt | 1.1 msec | 3.2 msec | 5.9 msec |
| Decrypt | 2.3 msec | 6.7 msec | 23 msec |
| Relinearization | 522 msec | n/a | n/a |
| Relinearization (optimized) | 38.1 msec | 635 msec | 1413 msec |

Table 4.5: Speed of the LTV FHE scheme

MixColumns and AddRoundKey. Among them, the ShiftRows step just switches bit positions and will not require any homomorphic operations. The AddRoundKey step simply XOR the round keys to the state values. If we pre-compute and encrypt the round keys beforehand, we will only need homomorphic additions in this step. The MixColumns step involves multiplications by constants and additions. Multiplications by constants in the finite field can be carried out via shift and additions[3], which will not generate much noise. The SubBytes step or the SBox is the only place where we require homomorphic multiplications and Relinearization operations.

An SBox lookup in AES correspond to a finite field inverse computation followed by the application of an affine transformation. The later operation is realized by simply multiplying the inverse by a $\{0, 1\}$ matrix and by XOR-ing the result with another constant vector. Therefore, if we can estimate the number of Relinearizations required for homomorphically computing the inverse of the input byte, we can estimate the number of Relinearizations required for the homomorphic AES encryption. In [67], the authors introduced a compact design for computing the inverse. The input byte in $GF(2^8)$ is converted using an isomorphism into a tower field representation which allows much more efficient inversion. We analyzed the design

[3]Multiplication by 2 can be evaluated by shifting. If the leftmost bit is 1, then XOR the result by (00011011). It can also be carried out without branching via $(b_7b_6b_5b_4b_3b_2b_1b_0) \cdot 2 = (b_6b_5b_4b_3b_2b_1b_00) \oplus (000b_7b_70b_7b_7)$. Multiplication by 3 can be computed by multiplying the input by 2 and adding the input. Clearly, these operations can be evaluated with only additions.

described in [67] and determined that the inversion can be evaluated with 4 levels of bit multiplications and 46 relinearizations in total. Converting to and from the sub-field representation involve only multiplication with $\{0, 1\}$ matrices and thus will only requires additions. Therefore, for the SubBytes step, we only need 4 levels of bit multiplications and 46 Relinearizations.

Since we will need 16 separate SBox substitution for one round, for the full 10 round 128 bit-AES block homomorphic encryption evaluation we need 40 levels of multiplications and 7,360 Relinearization operations. A 1024 bit coefficient size is sufficient to support 40 levels of multiplications. Therefore, we use 1024 bit coefficient size, which will slow down the Relinearization by roughly 16 times. If we omit the time required for operations other than Reliniearization, the time required for a full AES encryption can be estimated as about 74 minutes [4].

To verify our estimation, we implemented the AES using the LTV scheme with $|q| = 1024$ and $N = 512$. The results show that it takes about 85 minutes for the whole 10 round AES. We estimate that with $N = 768$, this time will increase to about 4 hours. In comparison, the AES implementation in [30] can evaluate the full AES rounds in 47 hours with 54 blocks evaluated together. A more aggressive implementation can evaluate 720 blocks together in 122 hours.

## 4.3 Compressing Ciphertexts

FHE schemes are powerful and allow arbitrary evaluations. On the other hand symmetric key schemes such as the block and stream ciphers have excellent performance in terms of ciphertext size. It would be ideal if we can take advantage of the desir-

[4] This is a rough estimate. The larger coefficient sizes may require the use of a larger dimension which will slow down Relinearization. In contrast, after each modulus switching operation, the Relinearization process becomes faster due to the drop in the coefficient size.

able features of the schemes *when* we need them. For instance, we may transfer data using block or stream ciphers with compact ciphertexts and then *convert* the ciphertext to the homomorphic one for further evaluation and then *convert* the ciphertext back into a scheme with short ciphertexts for transmission. This is precisely the problem we confront. Similar ideas were proposed in [33, 34, 35].

The concept of the scheme conversion is also employed in previous chapters for extending the supported circuits of partial HE schemes. In this section, we will formalize the notion of *secure converters* and provide efficient instantiation to reduce the bandwidth requirement. A secure converter allows an untrusted party to efficiently convert a ciphertext of one scheme to a ciphertext of another scheme, without affecting the data or gaining any significant information. We introduce a construction and show that a secure converter exists whenever the target scheme may homomorphically evaluate the decryption circuit of the initial scheme. Moreover, we show how to use converters to achieve bandwidth reduction in outsourced computation applications of FHE.

### 4.3.1 Secure Converters for Cryptographic Schemes

**Definition 4 (Converter)** *Given two encryption schemes $\mathcal{A}$ and $\mathcal{B}$, a* **converter** *from $\mathcal{A}$ to $\mathcal{B}$ is an ordered pair $\mathcal{C} = (\mathsf{KeyGen}, \mathsf{Convert})$ such that*

$$\begin{aligned} & & s &= \mathcal{C}.\mathsf{KeyGen}(\mathcal{A}, \mathcal{B}) \\ \textit{satisfies} & & \mathcal{B}.\mathsf{Enc}(x, k_{\mathcal{B}}) &= \mathcal{C}.\mathsf{Convert}(\mathcal{A}.\mathsf{Enc}(x, k_{\mathcal{A}}), s) \ . \end{aligned}$$

*Here, the extra input $s$ required for $\mathcal{C}.\mathsf{Convert}$ is called the* **converter key**. *We then say scheme $\mathcal{A}$ is* **convertible** *to scheme $B$, we refer to $C$ as a* **converter**, *and we*

*refer to the overall process informally as* **scheme conversion**.[5]

Note that the parameter $s$ in the definition of a converter can be generated from either $\mathcal{A}$ or $\mathcal{B}$ or by some algorithm involving both; we make no specific assumptions about how $s$ is constructed. It can even contain the keys of both schemes. We use $\mathsf{KeyGen}(\mathcal{A}, \mathcal{B})$ to abstract this process. A naïve construction is to design a converter that simply decrypts $\mathcal{A}.\mathsf{Enc}(x, k_{\mathcal{A}})$ first under $\mathcal{A}$ then encrypts again under scheme $\mathcal{B}$ to obtain $\mathcal{B}.\mathsf{Enc}(x, k_{\mathcal{B}})$. In this case, the converter key will be $s = (sk_{\mathcal{A}}, pk_{\mathcal{B}})$ or $s = (sk_{\mathcal{A}}, sk_{\mathcal{B}})$ in the symmetric key case. However, this toy approach is completely insecure as the plaintext is revealed during conversion.

Clearly the main challenge is to design efficient converters that will not leak any significant information. While $\mathcal{C}$ is used only for conversion from one ciphertext form to another, to capture the security of the overall scheme we need to consider the combination of schemes $\mathcal{A}, \mathcal{B}$ in conjunction with the converter $\mathcal{C}$. To simplify the security analysis we first define a merged scheme. The *merged scheme* $\mathcal{M} = (\mathcal{A}, \mathcal{B}, \mathcal{C})$ models, as a single encryption scheme, the overall process a message goes through as it is first encrypted using $\mathcal{A}$ and then converted using $\mathcal{C}$ to a valid ciphertext for scheme $\mathcal{B}$. (See Figure 4.5.) Rather than defining $\mathcal{M}.\mathsf{Enc}(\cdot, [pk_{\mathcal{A}}, s]) = \mathcal{C}.\mathsf{Convert}(\mathcal{A}.\mathsf{Enc}(\cdot, pk_{\mathcal{A}}), s)$ we choose to merge the role of $\mathcal{C}$ into the decryption process and define $\mathcal{M}.\mathsf{Dec}(\cdot, [sk_{\mathcal{B}}, s]) = \mathcal{B}.\mathsf{Dec}(\mathcal{C}.\mathsf{Convert}(\cdot, s), sk_{\mathcal{B}})$. This formalism will allow us to analyze the end-to-end security of the overall scheme.

Below, we show why the converter $\mathcal{C}$ is considered secure precisely when the merged scheme $\mathcal{M}$ is semantically secure. The notion of semantic security is well-known to be equivalent to indistinguishability under chosen plaintext attack (IND-CPA). In our proof (see appendix), we model IND-CPA using the game suggested

[5]This notion is not to be confused with the Fujisaki-Okamoto conversion scheme [49], with which it has no relationship.

**Merged Scheme** $\mathcal{M} = (\mathcal{A}, \mathcal{B}, \mathcal{C})$

**KeyGen:** Given security parameter $\tau$:

1. $(pk_{\mathcal{A}}, sk_{\mathcal{A}}) \leftarrow \mathcal{A}.\mathsf{KeyGen}(\tau); (pk_{\mathcal{B}}, sk_{\mathcal{B}}) \leftarrow \mathcal{B}.\mathsf{KeyGen}(\tau)$.
2. $s \leftarrow \mathcal{C}.\mathsf{KeyGen}(\mathcal{A}, \mathcal{B}, \tau)$.
3. Return $pk_{\mathcal{A}}$ and $s$ as the public key. Return $sk_{\mathcal{B}}$ as the secret key.

**Enc:** Given message $m$:

1. Set $c \leftarrow \mathcal{A}.\mathsf{Enc}(m, pk_{\mathcal{A}})$.
2. Return $c$ as the ciphertext.

**Dec:** Given a ciphertext $c$:

1. Compute $c' = \mathcal{C}.\mathsf{Convert}(c, s)$.
2. Return $a \leftarrow \mathcal{B}.\mathsf{Dec}(c', sk_{\mathcal{B}})$

Figure 4.5: Merged scheme modeling the two schemes and the converter as a single encryption scheme.

in [68]. In this game, the adversary is a pair of probabilistic algorithms $\mathcal{E} = (\mathcal{E}_1, \mathcal{E}_2)$. In the first step of the game, Algorithm $\mathcal{E}_1$ takes the public key as input and outputs a triple $(x_0, x_1, s)$ where $x_0$ and $x_1$ should of the same length and $s$ represents state information which may also include the public key. Next, one of $x_0$ and $x_1$ is selected at random and encrypted to form the challenge $y$. Finally, Algorithm $\mathcal{E}_2$ is given input $x_0$, $x_1$, $y$ and $s$ and determines which one among $x_0$ and $x_1$ was selected. More formally, we will have the following definition.

**Definition 5 (IND-CPA for public key schemes** [68]**)** *Let $\mathcal{A}$ be an encryption scheme and let $\mathcal{E} = (\mathcal{E}_1, \mathcal{E}_2)$ represent an adversary. For given security parameter $\tau$, let*

$$\boldsymbol{Adv}_{\mathcal{E}}(\tau) \stackrel{\text{def}}{=} |Pr\left[\mathcal{E}_2(x_0, x_1, s, y) = b \mid (pk_{\mathcal{A}}, sk_{\mathcal{A}}) \leftarrow \mathcal{A}.\mathsf{KeyGen}(\tau);\ (x_0, x_1, s) \leftarrow \mathcal{E}_1(pk_{\mathcal{A}});\right.$$
$$\left. b \leftarrow \{0,1\};\ y \leftarrow \mathcal{A}.\mathsf{Enc}(x_b, pk_{\mathcal{A}})\right] - 0.5 \mid ,$$

*where* $|x_0| = |x_1|$. *We say that* $\mathcal{A}$ *is IND-CPA secure if* $\boldsymbol{Adv}_{\mathcal{E}}(\cdot)$ *is negligible whenever* $\mathcal{E}$ *is a polynomial-time adversary.*

We are now ready to formally define a secure converter.

**Definition 6** *Given two IND-CPA public key schemes* $\mathcal{A}$, $\mathcal{B}$ *and a polynomial time converter* $\mathcal{C}$, *if the merged scheme* $\mathcal{M} = (\mathcal{A}, \mathcal{B}, \mathcal{C})$ *is IND-CPA, then we then say* $\mathcal{A}$ *is* **securely convertible** *to* $B$ *and call* $C$ *a* **secure converter** *of encryption schemes.*

Note that in this definition both $\mathcal{A}$ and $\mathcal{B}$ are public key schemes since this is the most readily useful case in homomorphic encryption. However, a similar definition can be given when one or both of $\mathcal{A}$ and $\mathcal{B}$ is/are symmetric key.

### 4.3.2 Using homomorphic encryption for secure converters

We finish this section by introducing a class of secure converters constructed using homomorphic evaluation. Specifically, we prove that whenever $\mathcal{B}$ is a homomorphic encryption scheme that is able to homomorphically evaluate the decryption circuit $\mathcal{A}.\mathsf{Dec}$, then $\mathcal{A}$ is securely convertible to $\mathcal{B}$. The converter $\mathcal{C}$ in this case has a key generation procedure[6] which encrypts the secret key of $\mathcal{A}$ using the public key of $\mathcal{B}$; the convert procedure belonging to $\mathcal{C}$ simply homomorphically decrypts the message.

**Theorem 1** *Given two semantically secure schemes* $\mathcal{A}$ *and* $\mathcal{B}$ *where* $\mathcal{B}$ *is capable of homomorphically decrypting ciphertexts of* $\mathcal{A}$ *with encrypted keys* $\mathcal{B}.\mathsf{Enc}(sk_{\mathcal{A}})$, *we*

[6]In practice, $\mathcal{B}$ may require more than just an encrypted copy of $sk_{\mathcal{A}}$ to perform the homomorphic decryption. For instance, if $\mathcal{A}$ is stateful, such as in a stream cipher, state information will also need to be included (in encrypted form if sensitive) alongside the secret key of $\mathcal{A}$.

*have that $\mathcal{A}$ is securely convertible to $B$. In this case, the converter $\mathcal{C}$ is defined as follows:*

$$\mathcal{C}.\mathsf{KeyGen}(\mathcal{A}, \mathcal{B}, \tau) = s = (pk_{\mathcal{B}}, \mathcal{B}.\mathsf{Enc}(sk_{\mathcal{A}}, pk_{\mathcal{B}})) \ ,$$

$$\mathcal{C}.\mathsf{Convert}(c, s) = \mathcal{B}.\mathsf{Eval}(\mathcal{B}.\mathsf{Enc}(c, pk_{\mathcal{B}}), \mathcal{A}.\mathsf{Dec}, \mathcal{B}.\mathsf{Enc}(sk_{\mathcal{A}}, pk_{\mathcal{B}})) \ .$$

We now restate Theorem 1 in terms of indistinguishability.

**Theorem 2** *Assume we are given two IND-CPA algorithms $\mathcal{A}$ and $\mathcal{B}$, where $\mathcal{B}$ can homomorphically evaluate $\mathcal{A}.\mathsf{Dec}$ and a converter $\mathcal{C}$ defined as in Theorem 1. Then the merged scheme $\mathcal{M} = (\mathcal{A}, \mathcal{B}, \mathcal{C})$ is IND-CPA.*

**Proof 1** *For the merged scheme $\mathcal{M} = (\mathcal{A}, \mathcal{B}, \mathcal{C})$ with security parameter $\tau$, suppose there exist an adversary $\mathcal{E} = (\mathcal{E}_1, \mathcal{E}_2)$ that can break the scheme in the IND-CPA game with a non-negligible advantage of $\boldsymbol{Adv}_{\mathcal{E}}$. Then we have*

$$\begin{aligned}
\boldsymbol{Adv}_{\mathcal{E}}(\tau) &= |Pr[\mathcal{E}_2(x_0, x_1, s, y) = b \ \mid \ (pk_{\mathcal{A}}, sk_{\mathcal{A}}) \leftarrow \mathcal{A}.\mathsf{KeyGen}(\tau); \\
&\quad (pk_{\mathcal{B}}, sk_{\mathcal{B}}) \leftarrow \mathcal{B}.\mathsf{KeyGen}(\tau); (x_0, x_1, s) \leftarrow \mathcal{E}_1(pk_{\mathcal{A}}, pk_{\mathcal{B}}, \mathcal{B}.\mathsf{Enc}(sk_{\mathcal{A}}, pk_{\mathcal{B}})); \\
&\quad b \leftarrow \{0,1\}; y \leftarrow \mathcal{A}.\mathsf{Enc}(x_b, pk_{\mathcal{A}})] - 0.5| \\
&\geq \epsilon(\tau) \ .
\end{aligned}$$

*Then we will show that we can use $\mathcal{E}$ to break either $\mathcal{A}$ or $\mathcal{B}$. Setup two games for $\mathcal{A}$ and $\mathcal{B}$. In the game for $\mathcal{A}$, the adversary $\mathcal{G}_{\mathcal{A}}$ has access to $pk_{\mathcal{A}}$ only. Then $\mathcal{G}_{\mathcal{A}}$ prepares a game for $\mathcal{E}$ as follows:*

1. $(pk'_{\mathcal{A}}, sk'_{\mathcal{A}}) \leftarrow \mathcal{A}.\mathsf{KeyGen}(\tau); (pk_{\mathcal{B}}, sk_{\mathcal{B}}) \leftarrow \mathcal{B}.\mathsf{KeyGen}(\tau);$
2. *Substitute $pk'_{\mathcal{A}}$ with the input $pk_{\mathcal{A}}$. Note that we use the exact same* KeyGen *function. Thus the distribution of the generated keys will be identical.*
3. $(x_0, x_1, s) \leftarrow \mathcal{E}_1(pk_{\mathcal{A}}, pk_{\mathcal{B}}, \mathcal{B}.\mathsf{Enc}(sk'_{\mathcal{A}}, pk_{\mathcal{B}}))$.

4. *Select* $b \xleftarrow{\$} \{0,1\}; y \leftarrow \mathcal{A}.\mathsf{Enc}(x_b, pk_{\mathcal{A}})$.

5. $\mathcal{G}_{\mathcal{A}}$ *treats the* $x_0, x_1, s, y$ *generated as the output of the its first stage* $\mathcal{G}_{\mathcal{A}1}$. $\mathcal{G}_{\mathcal{A}2}$ *will then call* $\mathcal{E}_2(x_0, x_1, s, y)$ *to guess* $b$.

*We call the advantage gained from this game* $\boldsymbol{Adv}_{\mathcal{A}}$. *Note that Steps 3-5 of this game actually form a standard game of* $\mathcal{E}$ *with an unmatched pair of* $s$ *and* $pk_{\mathcal{A}}$. *Obviously* $\boldsymbol{Adv}_{\mathcal{A}} \leq \boldsymbol{Adv}_{\mathcal{E}}$.

*In the game for* $\mathcal{B}$, *the adversary* $\mathcal{G}_{\mathcal{B}}$ *has access to* $pk_{\mathcal{B}}$ *only. Then* $\mathcal{G}_{\mathcal{B}}$ *prepares a game for* $\mathcal{E}$ *as the follows:*

1. $(pk_{\mathcal{A}}, sk_{\mathcal{A}}) \leftarrow \mathcal{A}.\mathsf{KeyGen}(\tau); (pk'_{\mathcal{B}}, sk'_{\mathcal{B}}) \leftarrow \mathcal{B}.\mathsf{KeyGen}(\tau)$;

2. *Substitute* $pk'_{\mathcal{B}}$ *with* $pk_{\mathcal{B}}$. *Note that we call the exact same* KeyGen *function, the distribution of* $pk_{\mathcal{B}}$ *and* $pk'_{\mathcal{B}}$ *will be identical.*

3. *Call* $\mathcal{A}.\mathsf{KeyGen}(\tau)$ *again to get another secret key* $sk'_{\mathcal{A}}$. $sk'_{\mathcal{A}}$ *will follow the same distribution of* $sk_{\mathcal{A}}$.

4. *Set* $x_0 = sk_{\mathcal{A}}$, $x_1 = sk'_{\mathcal{A}}$, $b \xleftarrow{\$} \{0,1\}$, $y \leftarrow \mathcal{B}.\mathsf{Enc}(x_b, pk_{\mathcal{B}})$. $s \leftarrow (pk_{\mathcal{A}}, pk_{\mathcal{B}})$.

5. $\mathcal{G}_{\mathcal{B}}$ *treats the* $x_0, x_1, s, y$ *generated in this way as the output of the its first stage* $\mathcal{G}_{\mathcal{B}1}$ *and forward them to* $\mathcal{G}_{\mathcal{B}2}$. *Note that* $s$ *only contain the public keys of* $\mathcal{A}$ *and* $\mathcal{B}$. *It is a valid piece of state information.*

6. $\mathcal{G}_{\mathcal{B}2}$ *calls* $(x'_0, x'_1, s') \leftarrow \mathcal{E}_1(pk_{\mathcal{A}}, pk_{\mathcal{B}}, y)$, *generate* $b'$ *and* $y' = \mathcal{B}.\mathsf{Enc}(x'_{b'}, pk_{\mathcal{B}})$ *accordingly.*

7. $\mathcal{G}_{\mathcal{B}2}$ *then calls* $\mathcal{E}_2(x'_0, x'_1, s', y')$ *to guess* $b'$.

8. *If the guess is correct,* $\mathcal{G}_{\mathcal{B}2}$ *guesses* $b = 0$ *else* $b = 1$.

*In this game, if $b = 0$, the selected $x_0$ is a valid encryption of $sk_{\mathcal{A}}$ that matches the public key of $\mathcal{A}$ contained in $s$. Steps 6-7 will be identical to a standard game for $\mathcal{E}$. Then $\mathcal{E}$ will have a non-negligible advantage of $\boldsymbol{Adv}_{\mathcal{E}}$. If $b = 1$, the selected $x_1$ is an encryption of $sk'_{\mathcal{A}}$ which does not match the public key of $\mathcal{A}$. In this case, the whole process will be identical to the game we designed for $\mathcal{A}$. We assume that the advantage of game $\mathcal{A}$ is $\boldsymbol{Adv}_{\mathcal{A}}$ (it may be negligible or not). Then we can compute the overall advantage of $\mathcal{G}_{\mathcal{B}}$ as*

$$\begin{aligned}
\boldsymbol{Adv}_{\mathcal{B}} &= Pr[\textit{Correct guess}] - 0.5 \\
&= Pr[b = 0 \mid \textit{Guess } 0] + Pr[b = 1 \mid \textit{Guess } 1] - 0.5 \\
&= 0.5 \cdot (0.5 + \boldsymbol{Adv}_{\mathcal{E}}) + 0.5 \cdot (0.5 - \boldsymbol{Adv}_{\mathcal{A}}) - 0.5 \\
&= 0.5\boldsymbol{Adv}_{\mathcal{E}} - 0.5\boldsymbol{Adv}_{\mathcal{A}}
\end{aligned}$$

*Now we can check the advantage of the two games together. It is clear that if $\boldsymbol{Adv}_{\mathcal{A}}$ is negligible, we will have*

$$\begin{aligned}
\boldsymbol{Adv}_{\mathcal{B}} &= 0.5\boldsymbol{Adv}_{\mathcal{E}} - 0.5\boldsymbol{Adv}_{\mathcal{A}} \\
&\approx 0.5\boldsymbol{Adv}_{\mathcal{E}}
\end{aligned}$$

*which is non-negligible. Therefore, either $\boldsymbol{Adv}_{\mathcal{A}}$ or $\boldsymbol{Adv}_{\mathcal{B}}$ is non-negligible.*

Note that Theorem 2 only covers the cases that both $\mathcal{A}$ and $\mathcal{B}$ are public key schemes. However, in the cases where $\mathcal{A}$ is symmetric, the converter which simply homomorphically decrypts will still be secure. The IND-CPA of the merged scheme in such cases can be proven using the same approach as we discussed above. More specifically, the merged scheme built from a symmetric $\mathcal{A}$ will also be symmetric. Therefore, we substitute the public key of $\mathcal{A}$ with an encryption oracal $\mathcal{O}_{\mathcal{A}}$ in the

proof as symmetric schemes do not have public keys. In this way, the $\mathcal{G}_{\mathcal{A}}$ part of the proof will become a standard IND-CPA game for symmetric schemes and the $\mathcal{G}_{\mathcal{B}}$ part will not be affected as $\mathcal{B}$ remains a public key scheme. In addition, parts of the process can still form valid games for $\mathcal{E}$. Therefore, the relation between $\mathbf{Adv}_{\mathcal{A}}$, $\mathbf{Adv}_{\mathcal{B}}$ and $\mathbf{Adv}_{\mathcal{E}}$ will still hold and so as the security reduction.

**Secure Converter for Symmetric Schemes**

Following similar approach, we can define a secure converter for symmetric schemes and the security proof given here can be easily modified to cover that scenario as well. First, we need a definition of IND-CPA for symmetric schemes. The definition also follows the game defined in [68]. The form is slightly different from the public case as we will have to rely on oracles for encryption.

**Definition 7 (IND-CPA for symmetric scheme)** *Let $\mathcal{A}$ be a symmetric encryption scheme and let $\mathcal{E} = (\mathcal{E}_1, \mathcal{E}_2)$ be an adversary. For given security parameter $\tau$, let*

$$\begin{aligned} \boldsymbol{Adv}_{\mathcal{E}}(\tau) \quad \stackrel{\text{def}}{=} \quad & |Pr[\mathcal{E}_2(x_0, x_1, s, y) = b \mid (sk_{\mathcal{A}}) \leftarrow \mathcal{A}.\mathsf{KeyGen}(\tau); \\ & (x_0, x_1, s) \leftarrow \mathcal{E}_1(\mathcal{O}); b \leftarrow \{0,1\}; y \leftarrow \mathcal{A}.\mathsf{Enc}(x_b, \mathcal{O})] - 0.5| \ , \end{aligned}$$

*where $|x_0| = |x_1|$ and $\mathcal{O}(\cdot)$ is an oracle that encrypts messages under the corresponding keys. The oracle used by $\mathcal{E}$ and oracle used to generate the challange are the same one. We say that $\mathcal{A}$ is IND-CPA secure if $\mathcal{E}$ is polynomial-time and $\boldsymbol{Adv}_{\mathcal{E}}(\cdot)$ is negligible.*

Following an approach to the one taken here, we may define the symmetric analogue of a merged scheme and a secure converter.

**Merged Scheme** $\mathcal{M} = (\mathcal{A}, \mathcal{B}, \mathcal{C})$

KeyGen: Given security parameter $\tau$:

1. $(sk_{\mathcal{A}}) \leftarrow \mathcal{A}.\mathsf{KeyGen}(\tau); (pk_{\mathcal{B}}, sk_{\mathcal{B}}) \leftarrow \mathcal{B}.\mathsf{KeyGen}(\tau)$.

2. $s \leftarrow \mathcal{C}.\mathsf{KeyGen'}(\mathcal{A}, \mathcal{B}, \tau)$.

3. Return $s$ as the public key. Return $sk_{\mathcal{A}}, sk_{\mathcal{B}}$ as the secret key.

Enc: Given message $m$:

1. Set $c \leftarrow \mathcal{A}.\mathsf{Enc}(m, sk_{\mathcal{A}})$.

2. Return $c$ as the ciphertext.

Dec: Given ciphertext $c$:

1. Compute $c' = \mathcal{C}.\mathsf{Convert}(c, s)$.

2. Return $a \leftarrow \mathcal{B}.\mathsf{Dec}(c', sk_{\mathcal{B}})$

Not that this merged scheme is effectively a secret key scheme. However, it employs a public key $s$ for the conversion. As in the public key case, the converter can either be merged into either the encryption or the decryption process without affecting the security level.

Likewise, we have the definition of a secure converter for the symmetric case.

**Definition 8** *Given an IND-CPA symmetric scheme $\mathcal{A}$, an IND-CPA public key scheme $\mathcal{B}$ and a polynomial time converter $\mathcal{C}$, if the merged scheme $\mathcal{M} = (\mathcal{A}, \mathcal{B}, \mathcal{C})$ is IND-CPA. We then say $\mathcal{A}$ is* **securely convertible** *to $B$ and call $C$ a* **secure converter**.

Now consider a symmetric $\mathcal{A}$, a homomorphic $\mathcal{B}$ and conversion effected by homomorphic decryptio. We have:

**Theorem 3** *Given an IND-CPA symmetric scheme $\mathcal{A}$ and an IND-CPA public key scheme $\mathcal{B}$ where $\mathcal{B}$ is capable of homomorphically decrypting ciphertexts of $\mathcal{A}$ with*

*encrypted keys* $\mathcal{B}.\mathsf{Enc}(sk_{\mathcal{A}})$*, then* $\mathcal{A}$ *is securely convertible to* $B$*. In this case, the converter* $\mathcal{C}$ *is defined as follows:*

$$\mathcal{C}.\mathsf{KeyGen}(\mathcal{A}, \mathcal{B}, \tau) = (pk_{\mathcal{B}}, \mathcal{B}.\mathsf{Enc}(sk_{\mathcal{A}}, pk_{\mathcal{B}})) \ ,$$

$$\mathcal{C}.\mathsf{Convert}(c, s) = \mathcal{B}.\mathsf{Eval}(\mathcal{B}.\mathsf{Enc}(c, pk_{\mathcal{B}}), \mathcal{A}.\mathsf{Dec}, (pk_{\mathcal{B}}, \mathcal{B}.\mathsf{Enc}(sk_{\mathcal{A}}, pk_{\mathcal{B}})) \ .$$

**Proof 2** *For the merged scheme* $\mathcal{M} = (\mathcal{A}, \mathcal{B}, \mathcal{C})$ *with security parameter* $\tau$*, suppose there exist an adversary* $\mathcal{E} = (\mathcal{E}_1, \mathcal{E}_2)$ *that can break the scheme in the IND-CPA game with a non-negligible advantage of* $\boldsymbol{Adv}_{\mathcal{E}}$*. Then we have:*

$$\begin{aligned}
\boldsymbol{Adv}_{\mathcal{E}}(\tau) &= |Pr[\mathcal{E}_2(x_0, x_1, s, y) = b \mid sk_{\mathcal{A}} \leftarrow \mathcal{A}.\mathsf{KeyGen}(\tau); \\
&\quad (pk_{\mathcal{B}}, sk_{\mathcal{B}}) \leftarrow \mathcal{B}.\mathsf{KeyGen}(\tau); (x_0, x_1, s) \leftarrow \mathcal{E}_1(\mathcal{O}, pk_{\mathcal{B}}, \mathcal{B}.\mathsf{Enc}(sk_{\mathcal{A}}, pk_{\mathcal{B}})); b \leftarrow \{0,1\}; \\
&\quad y \leftarrow \mathcal{A}.\mathsf{Enc}(x_b, pk_{\mathcal{A}})] - 0.5| \\
&\geq \epsilon(\tau) \ ,
\end{aligned}$$

*where* $\mathcal{O}$ *is the oracle that encrypts message for* $\mathcal{A}$*. Note that we can also pass some public information other than the oracle to* $\mathcal{E}_1$*.*

*Then we will show that we can use* $\mathcal{E}$ *to break either* $\mathcal{A}$ *or* $\mathcal{B}$*.*

*Then we will show that we can use* $\mathcal{E}$ *to break either* $\mathcal{A}$ *or* $\mathcal{B}$*. Setup two games for* $\mathcal{A}$ *and* $\mathcal{B}$*. In the game for* $\mathcal{A}$*, the adversary* $\mathcal{G}_{\mathcal{A}}$ *has access to the oracle* $\mathcal{O}$*. Then* $\mathcal{G}_{\mathcal{A}}$ *prepares a game for* $\mathcal{E}$ *as follows:*

1. $(sk'_{\mathcal{A}}) \leftarrow \mathcal{A}.\mathsf{KeyGen}(\tau); (pk_{\mathcal{B}}, sk_{\mathcal{B}}) \leftarrow \mathcal{B}.\mathsf{KeyGen}(\tau);$
2. *The game for* $\mathcal{E}$ *needs an oracle* $\mathcal{O}'$ *using* $sk'_{\mathcal{A}}$*. Substitute it with the input oracle* $\mathcal{O}$*. Note that we use the exactly same* KeyGen *function. Thus the distribution of the ciphertexts generated by* $\mathcal{O}$ *and* $\mathcal{O}'$ *will be identical.*
3. $(x_0, x_1, s) \leftarrow \mathcal{E}_1(\mathcal{O}, pk_{\mathcal{B}}, \mathcal{B}.\mathsf{Enc}(sk'_{\mathcal{A}}, pk_{\mathcal{B}}))$.

4. $b \leftarrow \{0,1\}; y \leftarrow \mathcal{A}.\mathsf{Enc}(x_b, \mathcal{O})$.

5. $\mathcal{G}_{\mathcal{A}}$ *treats the* $x_0, x_1, s, y$ *generated in this way as the output of the its first stage* $\mathcal{G}_{\mathcal{A}1}$. $\mathcal{G}_{\mathcal{A}2}$ *will then call* $\mathcal{E}_2(x_0, x_1, s, y)$ *to guess* $b$.

*We call the advantage gained from this game* $\boldsymbol{Adv}_{\mathcal{A}} \geq \epsilon_{\mathcal{A}}$. *Note that Steps 3-5 of this game actually form a standard game of* $\mathcal{E}$ *with an unmatched pair of* $s$ *and* $pk_{\mathcal{A}}$. *Obviously* $\boldsymbol{Adv}_{\mathcal{A}} \leq \boldsymbol{Adv}_{\mathcal{E}}$.

*In the game for* $\mathcal{B}$, *the adversary* $\mathcal{G}_{\mathcal{B}}$ *has access to* $pk_{\mathcal{B}}$ *only. Then* $\mathcal{G}_{\mathcal{B}}$ *prepares a game for* $\mathcal{E}$ *as the follows:*

1. $sk_{\mathcal{A}} \leftarrow \mathcal{A}.\mathsf{KeyGen}(\tau); (pk'_{\mathcal{B}}, sk'_{\mathcal{B}}) \leftarrow \mathcal{B}.\mathsf{KeyGen}(\tau)$;

2. *Substitute* $pk'_{\mathcal{B}}$ *with* $pk_{\mathcal{B}}$. *Note that we call the exact same* KeyGen *function, the distribution of* $pk_{\mathcal{B}}$ *and* $pk'_{\mathcal{B}}$ *will be identical.*

3. *Call* $\mathcal{A}.\mathsf{KeyGen}(\tau)$ *again to get another secret key* $sk'_{\mathcal{A}}$. $sk'_{\mathcal{A}}$ *will follow the same distribution of* $sk_{\mathcal{A}}$. *The two coresponding oracles* $\mathcal{O}$ *and* $\mathcal{O}'$ *will also generate ciphertexts following the same distribution.*

4. *Set* $x_0 = sk_{\mathcal{A}}$, $x_1 = sk'_{\mathcal{A}}$, $b \leftarrow \{0,1\}$, $y \leftarrow \mathcal{B}.\mathsf{Enc}(x_b, pk_{\mathcal{B}})$. $s \leftarrow (pk_{\mathcal{A}}, pk_{\mathcal{B}})$.

5. $\mathcal{G}_{\mathcal{B}}$ *treats the* $x_0, x_1, s, y$ *generated in this way as the output of the its first stage* $\mathcal{G}_{\mathcal{B}1}$ *and forward them to* $\mathcal{G}_{\mathcal{B}2}$. *Note that* $s$ *only contain the public keys of* $\mathcal{A}$ *and* $\mathcal{B}$. *It is a valid piece of state information.*

6. $\mathcal{G}_{\mathcal{B}2}$ *calls* $(x'_0, x'_1, s') \leftarrow \mathcal{E}_1(\mathcal{O}, pk_{\mathcal{B}}, y)$, *generate* $b'$ *and* $y' = \mathcal{B}.\mathsf{Enc}(x'_{b'}, pk_{\mathcal{B}})$ *accordingly.*

7. $\mathcal{G}_{\mathcal{B}2}$ *then calls* $\mathcal{E}_2(x'_0, x'_1, s', y')$ *to guess* $b'$.

8. *If the guess is correct,* $\mathcal{G}_{\mathcal{B}2}$ *guess* $0$ *for* $b$.

*In this game, if $b = 0$, the selected $x_0$ is an encryption of the $sk_{\mathcal{A}}$ that matches the used oracle of $\mathcal{A}$. Steps 6-7 will be identical to a standard game for $\mathcal{E}$. Then $\mathcal{E}$ should have a non-negligible advantage of $\boldsymbol{Adv}_{\mathcal{E}}$. If $b = 1$, the selected $x_1$ is an encryption of $sk'_{\mathcal{A}}$ which does not match the used oracle of $\mathcal{A}$. In this case, the whole process will be identical to the game we designed for $\mathcal{A}$, we assume that the advantage of game $\mathcal{A}$ is $\epsilon_{\mathcal{A}}$ (it may be negligible or not). Then we can calculate the overall advantage of $\mathcal{G}_{\mathcal{B}}$:*

$$\begin{aligned}
\boldsymbol{Adv}_{\mathcal{B}} &= Pr[\textit{Guess correct}] \\
&= Pr[b = 0 \mid \textit{Guess } 0] + Pr[b = 1 \mid \textit{Guess } 1] \\
&= 0.5 \cdot (0.5 + \boldsymbol{Adv}_{\mathcal{E}}) + 0.5 \cdot (0.5 - \boldsymbol{Adv}_{\mathcal{A}}) - 0.5 \\
&= 0.5\boldsymbol{Adv}_{\mathcal{E}} - 0.5\boldsymbol{Adv}_{\mathcal{A}}
\end{aligned}$$

*Now we can check the advantage of the two games together. It is clear that if $\boldsymbol{Adv}_{\mathcal{A}}$ is negligible, we will have*

$$\begin{aligned}
\boldsymbol{Adv}_{\mathcal{B}} &= 0.5\boldsymbol{Adv}_{\mathcal{E}} - 0.5\boldsymbol{Adv}_{\mathcal{A}} \\
&\approx 0.5\boldsymbol{Adv}_{\mathcal{E}}
\end{aligned}$$

*which is non-negligible. Therefore, either $\boldsymbol{Adv}_{\mathcal{A}}$ or $\boldsymbol{Adv}_{\mathcal{B}}$ is non-negligible.*

Here are a few examples of converters:

- As per Theorem 1, conversion from any efficiently computable encryption scheme $\mathcal{A}$ to any FHE $\mathcal{F}$ will permit a converter where the conversion scheme

is defined as the homomorphic evaluation of the decryption circuit of scheme $\mathcal{A}$ using secret keys encrypted under $F$.

- It is possible to think of a *bootstrappable* fully homomorphic scheme as a scheme $\mathcal{A}$ which is convertible to itself.

- A less trivial example is obtained by considering the homomorphic encryption scheme of Boneh, Goh and Nissim [19] (BGN) which uses a bilinear pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_1$ to effect the required multiplication in evaluating a 2-DNF formula. To encrypt a message $m$ pick a random $r \overset{R}{\leftarrow} \{0, 1, \ldots, n-1\}$ and compute $C = g^m h^r \in \mathbb{G}$. During homomorphic evaluation messages are mapped to a new group $\mathbb{G}_1$. We can view the BGN encryption scheme defined over domain $\mathbb{G}$ as the initial scheme $A$, and the BGN scheme defined over domain $\mathbb{G}_1$ as the target scheme $B$, and the pairing operator $e(.,.)$ as the converter.

### 4.3.3 Bandwidth Reduction via Secure Scheme Converters

| FHE Scheme | Ciphertext size per message bit |
|---|---|
| Gentry-Halevi [29] | 100 KBytes |
| BGV w/o batching [27] | 8.5 MBytes |
| BGV w batching [27] | 8.5 KBytes |
| LTV [28] | 16 KBytes |

Table 4.6: Ciphertext sizes for FHE schemes.

In the previous section we introduced the notion of secure converters and introduced a construction using the homomorphic properties of encryption schemes. We now show how a secure converter can greatly reduce bandwidth requirements in a two-party secure function evaluation setting shown in Figure 4.6 (left). Assuming the encryption function allows homomorphic computation, the server can evaluate a public function directly on the ciphertext and return the resulting ciphertext back

to the client without gaining any significant information about the original plaintext. Unfortunately, most homomorphic encryption schemes suffer from a basic flaw which prevents real-life deployment, i.e. bandwidth. As seen in Table 4.6, for FHEs the message expansion is quite severe. Fortunately, by utilizing the secure converter introduced in Section 4.3.1 we can reduce the bandwidth overhead drastically for communications between a client and a server as shown in Figure 4.6 (right). In both directions of the communication we may employ secure converters to achieve ciphertext compression[7].

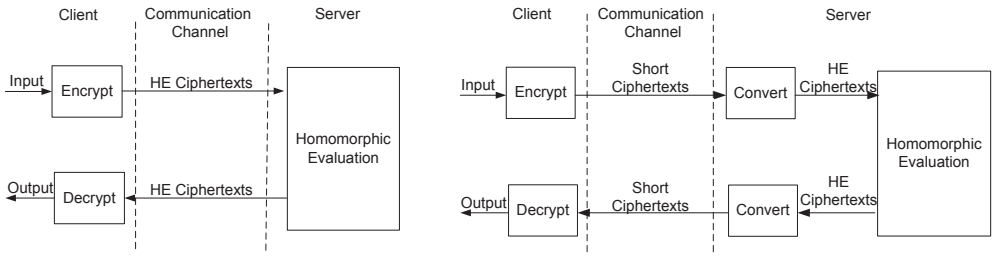


Figure 4.6: Standard Client-Server homomorphic function evaluation setting and bandwidth optimized scheme with converters.

### Ciphertext Compression at the Client Side

The client's ultimate goal is to encrypt the plaintext using a homomorphic encryption scheme to enable server side computation. Instead the client encrypts the plaintext using an ordinary symmetric key scheme with short ciphertexts. Once the ciphertext is received, the server may now employ a secure converter to uncompress the short ciphertext into a shorter one for further homomorphic evaluation.

In practice, the client may simply use a block cipher to encrypt the data before communication. As discussed earlier, an easy way to construct a secure converter

[7]We should note however that the technical requirements and the schemes are quite different since we assume the client side holds the decryption key while the server is only allowed to carry out computations on data in encrypted form.

at the server side is to decrypt the received ciphertext homomorphically using a FHE scheme. However, when we take performance into account, this approach does not seem feasible. For instance, [30] presents a technique and implementation to evaluate an AES circuit homomorphically. On average it takes more than 10 minutes to process a single AES block (128-bits) if the encryptions are batched. Instead, we find that stream ciphers are more suitable to facilitate efficient secure scheme conversion. Most stream ciphers encrypt messages on a bit by bit basis. To homomorphically decrypt the ciphertext, one can homomorphically generate the keystream bit by bit. Then decryption will be simply a homomorphic XOR addition of the key stream bits to the ciphertext bits. Furthermore, stream ciphers allow pre-computation of the keystream, which can hide the homomorphic evaluation latency, and significantly boost burst performance.

**Client Side Compression using Trivium and LTV.** Here we outline a concrete example of bandwidth reduction achieved by employing the Trivium stream cipher and assuming the homomorphic scheme employed is the one proposed by Lopez-Alt, Tromer and Vaikuntanathan (LTV).

Lopez-Alt, Tromer and Vaikuntanathan proposed a lattice based FHE which supports multikey computation. The security of this scheme can be reduced to Ring Learning with Error Problems (RLWE) with an assumption[8]. What makes the LTV FHE scheme desirable for the envisioned application, is it's simple decryption process. Very briefly, the LTV scheme works in a polynomial ring $\mathbb{Z}[x]/\langle x^k+1\rangle$. The ciphertext lies in the ring $R_q = R/qR$. The LTV scheme samples *small* polynomials $f', g$ from a $B$-bounded distribution $\chi$. [9] Then the scheme sets $f = 2f' + 1$ and

[8]The scheme can be reduced to RLWE through the Decisional Small Polynomial Ratio Problem (DSPR). In [40] Stehle and Steinfeld proved that DSPR is hard even for unbounded adversaries with certain parameter limitations. However, the scheme will lose the homomorphic property with this parameter selection. Therefore for the LTV scheme, the authors assume that the DSPR problem is still hard for their parameter selection.

[9]$B$-bounded means that the absolute value of the largest coefficient is smaller than the bound

$h = 2gf^{-1}$ and outputs $h$ as the public key and $f$ as the private key. To encrypt a message bit $m$, the LTV scheme samples some small polynomials $s, e$ again from $\chi$ and computes the ciphertext as $c = hs + 2e + m \in R_q$. To recover the message from the ciphertext, we first note that $\mu = fc = fhs + 2fe + fm = 2gs + 2fe + fm = 2gs + 2fe + (2f' + 1)m$ . Therefore $m = \mu \bmod 2 = fc \bmod 2$. It is clear that the majority of the steps in decryption involves only polynomial multiplication operations. This property makes the LTV scheme ideal to use with converters.

**Client Side Compression Using Trivium.** Ciphertext compression on the client side is rather easy to achieve. All that is needed is a compact (possibly symmetric) scheme whose decryption circuit is relatively easy to evaluate on the server side. Since we already assume that homomorphic encryption is already supported on the server side, the symmetric scheme selection is mostly a matter of efficiency. Therefore, for client side encryption we chose a stream cipher, i.e. Trivium, which features a simple decryption circuit which can be homomorphically decrypted rather efficiently on the server side[10].

Trivium is a synchronous stream cipher proposed by De Cannière and Preneel for the eSTREAM competition [70]. It has been selected as part of the portfolio 2 by the eSTREAM project. Trivium uses a total of 288-bits in three shift registers as the internal state. In each round, a bit is shifted into each of the three shift registers derived using a non-linear combination of some of the state bits and one output bit is produced from the state. Part of the IV is served as the key. Representing the three shift register bits by $a_i, b_i, c_i$ and '+' for an XOR and '$\cdot$' for an AND we can express the internal state update function of a Trivium round and the keystream

$B$.

[10]Indeed any stream cipher with a shallow circuit formulation will work here. However, one needs to be careful in selecting such ciphers due to algebraic attacks, cf. [69].

output bit $r_i$ as follows

$$\begin{aligned} a_i &= c_{i-66} + c_{i-111} + c_{i-110} \cdot c_{i-109} + a_{i-69} \quad , \\ b_i &= a_{i-66} + a_{i-93} + a_{i-92} \cdot a_{i-91} + b_{i-78} \\ c_i &= b_{i-69} + b_{i-84} + b_{i-83} \cdot b_{i-82} + c_{i-87} \\ r_i &= c_{i-66} + c_{i-111} + a_{i-66} + a_{i-93} + b_{i-69} + b_{i-84} \ . \end{aligned}$$

What it is important here is to realize that a keystream bit can be computed by just evaluating a second degree shallow circuit.

**Secure Converter from Trivium to LTV.** For decryption of Trivium we need to first encrypt the ciphertext bit using the FHE scheme and homomorphically evaluate the output bit $r_i$. A simple homomorphic XOR of $r_i$ and the ciphertext bit suffices to realize homomorphic decryption and thereby conversion to the FHE scheme. If LTV is used as the FHE, then we we can simply omit the initial encryption step from our converter, since addition of an unencrypted bit is equivalent to the addition of an encrypted bit in the LTV scheme which stores the message bit in the parity of the ciphertext. On the other hand, for the evaluation of $r_i$, as easily seen from the recurrence relations given above, only one layer of multiplications is needed for each round and the state bits are not used until after 64 rounds after being generated. This means that each bit is affected by at most one multiplication after about 64 rounds. If we use a FHE which will generate more noise from multiplication than addition (as most FHE schemes do), to run Trivium homomorphically, we will only need to evaluate the costly multiplication once every many rounds. In addition, as a common advantage of the stream ciphers, the online performance of the converter can be further improved by employing precomputation. Later inwe present implementation results.

### Ciphertext Compression at the Server Side

After homomorphic evaluation the server wants to communicate a compressed ciphertext back to the client. Clearly the server can convert the ciphertext to another FHE scheme with smaller ciphertext size. Homomorphically decrypting the ciphertext using another FHE scheme is straightforward. However, the gain in ciphertext reduction will be limited. Therefore, we need to turn to partially homomorphic encryption schemes. Although severely limited in their homomorphic properties, as long as they are capable evaluate the FHE decryption circuit, they yield secure converters. In addition, some partial homomorphic schemes support a large message space with reasonable ciphertext sizes[11].

The realize the converter we can homomorphically decrypt the cipher using the new partial HE scheme and generate a ciphertext under the partial HE. However, after reviewing partial HE and FHE schemes we could not find *any* FHE schemes that could be homomorphically decrypted by any of the existing PHE schemes. This is not surprising since what we are trying to achieve here is akin to *bootstrapping*, but with weaker homomorphic properties. One way to overcome this problem is to only partially decrypt the ciphertext. The protocol works correctly as long as the client can extract the actual final result from the sub-results.

**Server Side Compression Using Paillier's Scheme.** In this section we provide a concrete construction, and show that it is possible to homomorphically *nearly* decrypt ciphertexts in the LTV FHE scheme using Pailler's HE scheme. We now very briefly review both and highlight the features of these schemes that will enable secure ciphertext compression.

Paillier [16] proposed an additive homomorphic encryption scheme based on a

[11] Here the comparison is made against ciphertext sizes of fully homomorphic encryption schemes.

decision problem[12] closely related to the problem of deciding $n^{\text{th}}$ residues in the ring $\mathbb{Z}_{n^2}$. Let $p$ and $q$ be large primes and set $n = pq$; the group of units in the ring $\mathbb{Z}_n$ has exponent $\lambda = \lambda(n) = \text{lcm}(p-1, q-1)$. The $n^{\text{th}}$ roots of unity in $\mathbb{Z}_{n^2}$ are $u = 1 + kn$ $(0 \leq k < n)$ and $k$ is recovered as $L(u) = \frac{u-1}{n}$. Paillier's scheme requires a base $g$ satisfying $\gcd(L(g^\lambda \bmod n^2), n) = 1$. Primes $p$ and $q$ and the integer $\lambda$ are kept private while the public key is $(n, g)$. To encrypt a message $m \in \mathbb{Z}_n$, Alice generates a random $r$ and computes $\mathsf{Enc}(m) = g^m r^n \pmod{n^2}$. Paillier [16] gives evidence that recovering $m$ from $c = \mathsf{Enc}(m)$ without the knowledge of $p, q$ or $\lambda$ is hard. However, with the knowledge of private key $\lambda$ (or, equivalently, the factorization $n = pq$), $m$ may be easily recovered from $c$ by computing $m = \mathsf{Dec}(c) = \frac{L(c^\lambda \bmod n^2)}{L(g^\lambda \bmod n^2)} \pmod{n}$ . The encryption function is additively homomorphic and the scheme supports scaler multiplications to constants in plaintext, i.e., $\mathsf{Enc}(m_1) \cdot \mathsf{Enc}(m_2) = g^{m_1} r_1^n g^{m_2} r_2^n \pmod{n^2} = \mathsf{Enc}(m_1 + m_2)$ , and $[\mathsf{Enc}(m_1)]^a = (g^{m_1} r_1^n)^a \pmod{n^2} = \mathsf{Enc}(a \cdot m_1)$.

**Homomorphic Converters.** The decryption of the LTV FHE scheme requires multiplication of two polynomials in $R_q$ and then a reduction modulo 2. Since the message is kept in the parity of the constant term, we can further simplify the process by ignoring other terms. Suppose we have the ciphertext $c = c_0 + c_1 x + \ldots + c_{k-1} x^{k-1}$ and the secret key $f = f_0 + f_1 x + \ldots + f_{k-1} x^{k-1}$. The decryption computation $b = a_0 \pmod 2$ can be expressed as

$$b = [c_0 f_0 - \sum_{i=1}^{k-1} c_i f_{k-i} \bmod q\,] \pmod 2 = [c_0 f_0 + \sum_{i=1}^{k-1} c_i (0 - f_{k-i}) \bmod q\,] \pmod 2 .$$

If we further simplify the computations by postponing the modular reduction and the modulo 2 reduction, the polynomial multiplication can be evaluated with only

[12] In [16], Paillier defines the "Composite Residuosity Class Problem" and gives compelling evidence for its hardness.

one level of multiplications. Note that we can postpone the reductions, since Paillier is additively homomorphic and that reductions modulo $q$ (and later modulo 2) can be simply realized as subtractions — an operation compatible with the homomorphic property of Paillier's Scheme. Postponing the reductions allows us to evaluate a significant portion of the decryption by homomorphically evaluating a sum of products computation.

We should also point out that normally the first step in a converter is to encrypt using the target scheme $\mathcal{B}$. However, here $s_i$ is part of the secret key and has to be kept in encrypted form. The $c_i$ terms, on the other hand, are part of the ciphertext and thus can be processed in plaintext form. Since Paillier's scheme can evaluate a scalar multiplication computations the where we can compute the encrypted scalar product where one operand can be in plaintext form. Therefore, we can completely avoid the initial encryption step of the converter and focus only on the decryption procedure.

All operations involved in the decryption process are scalar multiplications and additions, which can be evaluated homomorphically using Paillier's scheme. Let $\mathsf{ScalarMul}$ represent the scalar multiplication process of Paillier's scheme. We assume that the server is also given the Paillier encrypted private key $f$ of the LTV FHE computed during the key generation process of the converter, i.e. $s = C.\mathsf{KeyGen}$ where

$$\begin{aligned} C.\mathsf{KeyGen} = \quad & \langle \mathsf{Paillier.Enc}(f_0)), \mathsf{Paillier.Enc}(0 - f_{k-1}), \\ & \mathsf{Paillier.Enc}(0 - f_{k-2}), \ldots, \mathsf{Paillier.Enc}(0 - f_1) \rangle \ . \end{aligned}$$

The server realizes a secure converter from LTV FHE to Paillier HE as follows.

$$c' = \mathsf{Convert}(c, f) = \sum_{i=0}^{k-1} \mathsf{ScalarMul}(c_i, s_i)$$

The big summation as well as the negations are computed homomorphically. Finally, the message is recovered on the client side by first decrypting the ciphertext using Paillier and then by completing the final reductions which were postponed earlier during the LTV FHE decryption process as

$$m = [\mathsf{Paillier.Dec}(c') \bmod q] \pmod 2 .$$

### 4.3.4 Performance

To evaluate the performance of the two secure converters outlined above we realized the LTV FHE scheme, Trivium and the Paillier HE scheme with a reasonable choice of parameters on an Intel Core i7 3770K running at 3.5 GHz with 8 GB RAM. The codes were developed in the C language linked to the NTL and GMP libraries to gain access to lower level arithmetic functions. We instantiated the LTV FHE scheme with a modest choice of parameters with 512 dimensions and 256-bit coefficients. Therefore, each ciphertext in the LTV scheme will require 128 KBytes. Trivium is built with standard parameters as originally defined, and for Paillier's scheme we use a 2048-bit modulus. The most expensive operation is clearly the LTV evaluation operation. In our implementation we found the cost of an LTV relinearization operation takes about 524 msec.

The implementation results are summarized in Table 4.7. The use of converters reduces communication cost dramatically. For client side compression with Trivium

| Converter | Time | Time/H. Eval | Ciphertext size | Compression Rate |
|---|---|---|---|---|
| Trivium to LTV | 223 msec | 5.87 | 1 bit | $2^{17}\times$ |
| LTV to Paillier | 990 msec | 26 | 4096 bit | $256\times$ |

Table 4.7: Cost of conversion, in ciphertext conversion time, rate of the conversion time by the time needed for homomorphic AND evaluation with LTV, and gains in ciphertext reduction rate.

and LTV, we achieve a factor $2^{17}$ improvement in the ciphertext size! Homomorphic decompression on the server takes about 223 msec only about 5.87 times more than the time it takes to perform a homomorphic evaluation operation on the server using LTV. Server side compression takes about 990 msec, which is about 26 times as long as a single homomorphic evaluation using LTV. We gain, however, 256-fold improvement in the ciphertext size.

# Chapter 5

# Conclusion

## 5.1 Summaries and Conclusions

In this dissertation, we explored varieties of approaches to practical homomorphic encryption schemes. We set forth the criteria for a scheme to be practical and proposed a number of ways to improve existing homomorphic encryption schemes.

Versatility is one of the major problems faced by partial HE schemes. It is usually caused by the fact that partial HE schemes support only one type of operation. To solve this problem, we proposed a special family of partial HE schemes and discussed ways to convert between partial HE schemes that support different types of operations. In this way, we can convert to corresponding schemes when certain types of operations are required. This approach can extend the supported circuits of partial HE schemes significantly.

Another problem faced by certain partial HE schemes is the small message size, which can also be viewed as lacking of versatility. The ElGamal-type encryption schemes and the BGN scheme are examples of this type of scheme. We applied CRT to these schemes to replace one discrete logarithm problem in a large space by several

similar problems in a more tractable search space while retaining full security. This approach helps extending the message space of these schemes significantly.

Existing FHE schemes face serious performance problems. To address the speed problem, we implemented the Gentry-Halevi FHE scheme and the LTV FHE scheme and achieved significant improvement in performance compare to previous works. More specifically, our implementation of the Gentry-Halevi FHE scheme tuns about 174, 7.6 and 13.5 times faster than the original Gentry-Halevi code for encryption, decryption and recryption respectively. Our implementation of the LTV FHE scheme achieved millisecond level Encrypt and Decrypt and the Relinearization takes about 38 msecs for $N = 512$ and $|q| = 256$.

To reduce the large ciphertext size, scheme conversion was formalized to "compress" the ciphertext of FHE schemes. Given a scheme converter, we may transfer data using block or stream ciphers with compact ciphertexts and then *convert* the ciphertext to the homomorphic one for further evaluation and then *convert* the ciphertext back into a scheme with short ciphertexts for transmission. To formalize this concept, we defined the secure converter, which allows an untrusted party to efficiently convert a ciphertext of one scheme to a ciphertext of another scheme, without affecting the data or gaining any significant information. In addition, we introduced a construction and show that a secure converter exists whenever the target scheme may homomorphically evaluate the decryption circuit of the initial scheme. Moreover, we showed how to use converters to achieve bandwidth reduction in outsourced computation applications of FHE.

## 5.2 Recommendation for Future Work

Further efforts can be made to further improve the HE schemes based on our works. The implementation of the LTV FHE scheme can be further optimized. The current implementation is a software implementation running on a single core. If GPUs or some dedicated hardware is utilized, the speed could be further improved. The parameter selection and the optimization we proposed will also apply to these possible implementations. In addition, as the running speed of the FHE schemes is increased to a level that is close to be enough for practical applications, the actual speed for real world applications using the FHE schemes becomes an interesting topic. Efficient implementation of some real world algorithms other than the AES using the LTV or other FHE schemes will be a valuable research topic.

Scheme conversion will lead to new interesting topics. The reduction of the bandwidth requirement for FHE scheme will enable some new constructions that were impractical with the bandwidth limitation. For instance, if the costly recryption process for the FHE schemes is evaluated by the client or by some trusted third parties, it can be simplified to decrypting the ciphertexts and encrypting the messages again. However, this requires transferring the ciphertexts between parties for each recryption. It is not practical due to the large ciphertext size of the FHE schemes. However, with the reduction of the bandwidth requirement, such "interactive" FHE construction becomes possible and is an interesting topic for research.

The other application of the scheme conversion will also lead to new possibilities. Recall that we convert between an additive scheme and a multiplicative scheme for limited times to build a DNF-formula evaluation scheme. It would be interesting whether we could find a pair of additive and multiplicative schemes that supports unlimited times of conversions. The existence of such schemes will imply a new

possible approach to construct fully homomorphic encryption schemes.

# Bibliography


[1] E Ray, E Schultz Virtulization Security Proc. of the 5th Annual Workshop on Cyber Security, 2009

[2] Oberheide, J. and Cooke, E. and Jahanian, F. Empirical exploitation of live virtual machine migration Proc. of BlackHat DC convention, 2008

[3] Halderman, J.A. and Schoen, S.D. and Heninger, N. and Clarkson, W. and Paul, W. and Calandrino, J.A. and Feldman, A.J. and Appelbaum, J. and Felten, E.W. LestWe Remember: Cold Boot Attacks on Encryption Keys Proc. 2008 USENIX Security Symposium

[4] Acıiçmez, O. and Koç, Ç. and Seifert, J.P. Predicting secret keys via branch prediction Topics in Cryptology–CT-RSA 2007, Springer, 2007

[5] Yao, Andrew Chi-Chih, How to generate and exchange secrets, Foundations of Computer Science, 1986., 27th Annual Symposium on. IEEE, 1986.

[6] Tomas Sander, Adam Young, Moti Yung, Non-Interactive CryptoComputing For $NC^1$. FOCS 1999: pp. 554–567.

[7] R.L. Rivest, L. Adleman, and M.L. Dertouzos. On data banks and privacy homomorphisms. In Foundations of Secure Computation, 1978.

[8] E. Mykletun, J. Girao, and D. Westhoff. Public Key Based Cryptoschemes for Data Concealment in Wireless Sensor Networks. In IEEE Int. Conference on Communications ICC, Istanbul, Turkey, June 2006.

[9] Osman Ugus, Dirk Westhoff, Ralf Laue, Abdulhadi Shoufan, Sorin A. Huss, Optimized Implementation of Elliptic Curve Based Additive Homomorphic Encryption for Wireless Sensor Networks. WESS '07, Salzburg, Austria, 2007.

[10] Aggelos Kiayias, Moti Yung, Tree-Homomorphic Encryption and Scalable Hierarchical Secret-Ballot Elections. Financial Cryptography 2010: pp. 257–271.

[11] Julien Bringer, Hervé Chabanne, Malika Izabachéne, David Pointcheval, Qiang Tang and Sébastien Zimmer, An Application of the Goldwasser-Micali Cryptosystem to Biometric Authentication, Information Security and Privacy, LNCS 4586, pp. 96–106, 2007.

[12] M. Kantarcioglu, *Privacy-preserving distributed data mining and processing on horizontally partitioned data*, PhD. dissertation, Department of Computer Science, Purdue University, 2005.

[13] S. Goldwasser, S. Micali, Probabilistic Encryption, J. Comp. Sys. Sci., 28, pp. 270–299, 1984.

[14] Josh Benaloh, Dense Probabilistic Encryption, SAC 94, pages 120–128, 1994.

[15] D. Naccache, J. Stern. A New Public Key Cryptosystem Based on Higher Residues. Proceedings of the 5th ACM CCS, pages 59–66, 1998.

[16] P. Paillier, Public-key cryptosystems based on composite degree residuosity classes, in Advances in Cryptology EUROCRYPT'99, LNCS 1592, pp. 223–238, Springer, New York, NY, USA, 1999.

[17] I. Damgård and M. Jurik. A Length-Flexible Threshold Cryptosystem with Applications. ACISP '03, pp. 350–356.

[18] T. Okamoto and S. Uchiyama. A New Public-Key Cryptosystem as Secure as Factoring. Eurocrypt' 08, LNCS 1403, pp. 308-318, 1998.

[19] D. Boneh, E. Goh, K. Nissim, Evaluating 2-DNF Formulas on Ciphertexts, TCC '05, LNCS 3378, pp. 325-341, 2005.

[20] C. Peikert and B. Waters. Lossy Trapdoor Functions and Their Applications. STOC '08, pp. 187–196.

[21] A. Kawachi, K. Tanaka, K. Xagawa. Multi-bit cryptosystems based on lattice problems. PKC '07, pp. 315–329.

[22] C.A. Melchor, G. Castagnos, and P. Gaborit. Lattice-based homomorphic encryption of vector spaces. ISIT '08, pp. 1858–1862.

[23] Carlos Aguilar Melchor and Philippe Gaborit, Javier Herranz, Additively Homomorphic Encryption with $d$-Operand Multiplications. CRYPTO 2010, pp. 138–154, 2010.

[24] F. Armknecht and A.-R. Sadeghi. A new approach for algebraically homomorphic encryption. Eprint 2008/422.

[25] C. Gentry, Fully homomorphic encryption using ideal lattices, Symposium on the Theory of Computing (STOC), 2009, pp. 169-178.

[26] Van Dijk, Marten, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. Advances in Cryptology EUROCRYPT 2010 (2010): 24-4

[27] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. Innovations in Theoretical Computer Science, ITCS (2012): 309-325.

[28] Adriana Lopez-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In Proceedings of the 44th symposium on Theory of Computing, pp. 1219-1234. ACM, 2012.

[29] Craig Gentry and Shai Halevi, "Implementing Gentry's fully-homomorphic encryption scheme," *Advances in Cryptology–EUROCRYPT 2011*, pp. 129–148, 2011.

[30] Craig Gentry, Shai Halevi, and Nigel Smart. Homomorphic evaluation of the AES circuit. Advances in Cryptology – CRYPTO 2012 (2012): 850-8

[31] Yin Hu, William J. Martin and Berk Sunar. Enhanced Flexibility for Homomorphic Encryption Schemes via CRT. Industrial Track of ACNS 2012, 2012.

[32] Wei Wang, Yin Hu, Lianmu Chen, Xinming Huang and Berk Sunar. Accelerating Fully Homomorphic Encryption on GPUs. Proceeding of 2012 IEEE HPEC, 2012.

[33] Naehrig, Michael, Kristin Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical?. In proceedings of the 3rd ACM workshop on Cloud computing security workshop. ACM, 2011.

[34] Brakerski, Zvika, and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on. IEEE, 2011.

[35] Gentry, Craig, and Shai Halevi. Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on. IEEE, 2011.

[36] P. Paillier, Trapdooring discrete logarithms on elliptic curves over rings, ASIACRYPT 2000, LNCS 1976, pp. 573–584. 2000.

[37] Craig Gentry, Shai Halevi, and Nigel Smart. Fully homomorphic encryption with polylog overhead. Manuscript, 2011.

[38] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. Manuscript at http://eprint.iacr.org/2011/133, 2011.

[39] Jeffrey Hoffstein, Jill Pipher, and Joseph Silverman. NTRU: A ring-based public key cryptosystem. Algorithmic number theory (1998): 267-288.

[40] Damien Stehle and Ron Steinfeld. Making NTRU as secure as worst-case problems over ideal lattices. Advances in CryptologyCEUROCRYPT 2011 (2011): 27-4

[41] Wei Wang, Yin Hu, Lianmu Chen, Xinming Huang and Berk Sunar, Accelerating Fully Homomorphic Encryption on GPUs. Proceeding of 2012 IEEE HPEC, 2012.

[42] T. ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory, IT- 31(4):469–472, 1985.

[43] Crypto++ Library 5.6.1, http://www.cryptopp.com/

[44] Frederiksen, T.K., A Practical Implementation of Regev's LWE-based Cryptosystem, 2010.

[45] MPIR 2.4.0, http://www.mpir.org/

[46] Arjen K. Lenstra, Eric R. Verheul, Selecting Cryptographic Key Sizes. Vol 14, No 4, Journal of cryptology, Springer, 2001. pp. 255–293.

[47] Victor S. Miller, The Weil Pairing, and Its Efficient Calculation, Journal of Cryptology 17(4):235-261, Springer, 2004

[48] Menezes, A.J. and Van Oorschot, P.C. and Vanstone, S.A., Handbook of applied cryptography, CRC Press, 2005.

[49] E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In: *Advances in Cryptology - CRYPTO ?9*, LNCS vol. 1666, pp. 537?54, Springer, 1999.

[50] C. Gentry, *A Fully Homomorphic Encryption Scheme*. Ph.D. thesis, Department of Computer Science, Stanford University, 2009.

[51] A. Schönhage and V. Strassen, Schnelle multiplikation grosser zahlen. Computing, vol. 7, no. 3, pp. 281–292, 1971.

[52] N. Emmart and C. Weems, High precision integer multiplication with a gpu using strassen's algorithm with multiple fft sizes. In Parallel Processing Letters, vol. 21, no. 3, p. 359, 2011.

[53] J. Solinas, Generalized mersenne numbers. Technical Reports, 1999.

[54] D. Bailey, FFTs in external of hierarchical memory. In Proceedings of the 1989 ACM/IEEE conference on Supercomputing, ACM, 1989, pp. 234–242.

[55] C. Mclvor, M. McLoone, and J. McCanny, Fast montgomery modular multiplication and RSA cryptographic processor architectures. In Signals, Systems and Computers, 2003. Conference Record of the Thirty-Seventh Asilomar Conference on, vol. 1. IEEE, 2003, pp. 379–384.

[56] A. Daly and W. Marnane, Efficient architectures for implementing montgomery modular multiplication and RSA modular exponentiation on reconfigurable logic. In Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays. ACM, 2002, pp. 40–49.

[57] P. Giorgi, T. Izard, A. Tisserand *et al.*, Comparison of modular arithmetic algorithms on gpus. ParCo'09: International Conference on Parallel Computing. 2009.

[58] P. Montgomery, Modular multiplication without trial division. In Mathematics of computation, vol. 44, no. 170, pp. 519–521, 1985.

[59] P. Barrett, Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor. In Advances in Cryptology CRYPTO'86. pp. 311–323. Springer, 1987.

[60] NTL: A Library for doing Number Theory, http://www.shoup.net/ntl

[61] GMP: The GNU Multiple Precision Arithmetic Library, http://gmplib.org

[62] Nicolas Gama and Phong Nguyen. Predicting lattice reduction. Advances in Cryptology-EUROCRYPT 2008 (2008): 31-5

[63] Richard Lindner and Chris Peikert. Better key sizes (and attacks) for LWE-based encryption. Topics in Cryptology CT-RSA 2011 (2011): 319-339

[64] Jeffrey Hoffstein, Joseph H. Silverman, and William Whyte. Estimated breaking times for NTRU lattices. In version 2, NTRU Cryptosystems (2003) http://www.ntru.com/cryptolab/tech_notes.html, 1999.

[65] Schnorr, Claus-Peter, and Martin Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. Mathematical programming 66, no. 1 (1994): 181-199.

[66] Chen, Yuanmi, and Phong Q. Nguyen. BKZ 2.0: better lattice security estimates. In Advances in CryptologyCASIACRYPT 2011, pp. 1-20. Springer Berlin Heidelberg, 201

[67] Canright, David. A very compact S-box for AES. Cryptographic Hardware and Embedded Systems-CHES 2005 (2005): 441-45

[68] Bellare, Mihir, Anand Desai, David Pointcheval, and Phillip Rogaway. Relations among notions of security for public-key encryption schemes. In Advances in Cryptology CRYPTO'98, pp. 26-45. Springer Berlin/Heidelberg, 1998.

[69] Jean-Philippe Aumasson, Itai Dinur, Willi Meier, and Adi Shamir. Cube Testers and Key Recovery Attacks On Reduced-Round MD6 and Trivium Fast Software Encryption - FSE 2009, 1–22.

[70] De Cannière, Christophe. Trivium: A stream cipher construction inspired by block cipher design principles. Information Security (2006): 171-186.