Homomorphic Encryption and Database Query Privacy

by

© Sudharaka Palamakumbura

A thesis submitted to the

School of Graduate Studies

in partial fulfillment of the

requirements for the degree of

Master of Science

Department of Mathematics and Statistics

Memorial University of Newfoundland

April 2016

St John's　　　　Newfoundland & Labrador

Abstract

Homomorphic encryption is a particular type of encryption method that enables computing over encrypted data. This has a wide range of real world ramifications such as being able to blindly compute a search result sent to a remote server without revealing its content.

In the first part of this thesis, we discuss how database search queries can be made secure using a homomorphic encryption scheme based on the ideas of Gahi et al. Gahi's method is based on the integer-based fully homomorphic encryption scheme proposed by Dijk et al. We propose a new database search scheme called the *Homomorphic Query Processing Scheme*, which can be used with the ring-based fully homomorphic encryption scheme proposed by Braserski.

In the second part of this thesis, we discuss the cybersecurity of the smart electric grid. Specifically, we use the Homomorphic Query Processing scheme to construct a keyword search technique in the smart grid. Our work is based on the Public Key

Encryption with Keyword Search (PEKS) method introduced by Boneh et al. and a Multi-Key Homomorphic Encryption scheme proposed by López-Alt et al.

A summary of the results of this thesis (specifically the Homomorphic Query Processing Scheme) is published at the 14th Canadian Workshop on Information Theory (CWIT) [39].

Table of Contents

List of Tables

List of Figures

Acknowledgments

This project would not have been a success if not for some incredible people whose help I will never forget. In this spirit, I would like to thank my supervisor Prof. Hamid Usefi for his valuable support and constant encouragement; couldn't have done it without you. Brad Sheppard for the valuable conversations we had about mathematics, computer science and every other random thought that came to our mind. Faith Lee for being there when I needed the most, O'saf Gayas for providing encouragement when I lacked hope, and Hugo Morales and Nishani Perera whose constant support and kind words encouraged me during the sleepless nights that I spent on this thesis.

Thank You!

Chapter 1

Introduction

With the advent of the digital era in the 1980s, there was a rapid growth in communication and networking all over the globe. This was followed by the trend of miniaturization of digital equipment, and current standards allow us to use a smartphone with the same computational power as the whole of NASA back in the 1970s [29]. The fast-moving trend of digitalization has enabled access of most services from distant locations. For example, a recent study shows that 74% of smartphone users use a location based service (such as Google Maps) to find directions and other location based information [48]. Moreover, the adaptation of these kind of services in healthcare are becoming increasingly common with cloud-based health recording and genomic data management tools such as Microsoft Health. However, the widespread adaptation of location-based services poses a threat to users because their personal data, such as location, health records, and sometimes even genomic data, is shared on the web without any guarantee of privacy.

The privacy of data sent via the web can be guaranteed if they are encrypted before sending. However, encryption also makes server side computations impossible unless the data center is provided with the decryption key. This problem can be addressed by fully homomorphic Encryption schemes, which enable operations on encrypted data such that, when decrypted, will output a corresponding plaintext.

1.1 Motivation

Existing database systems use various query processing technologies to process database queries. For example most databases use Structured Query Language (SQL) to handle user defined queries, whereas query processing schemes such as Java Persistence Query Language (JPQL) is used by database administrators for programatically handle database queries. The processing of database records by users as well as administrators is subjected to a wide range of privacy issues. On the part of the user, the most common privacy concern is that queries they send could be viewed at the server side. For example a query that is sent to a search engine could be viewed, stored or manipulated by a malicious party (i.e., a malicious database administrator) after it is received by the server. This is a major privacy concern, and currently the only practical solution that is being used is defining strict privacy laws that govern appropriate usage of user data. Cryptographic solutions that are being used can protect data during trasmit, but once it reaches its destination, to process the queries the data packets need to be decrypted. This is the main motivation of homomorphic encryption. As we shall discuss homomorphic encryption can be used to prevent data breaches during storing as well as during query operations.

This work attempts to improve upon a method proposed by Gahi et al. [19] to homomorphically encrypt database queries. Their work specifically uses the DGHV fully homomorphic encryption scheme [15]. The DGHV scheme operates on plaintext bits separately, and thus Gahi's method requires a large amount of computations to perform even a simple operation such as integer multiplication. We propose an alternative to Gahi's method, which we call *Homomorphic Query Processing*. Our method is not restricted to the DGHV scheme and can be used with more modern fully homomorphic encryption schemes. For example, using our Homomorphic Query Processing technique with the more recent ring based fully homomorphic encryption scheme proposed by Braserski et al. [12], which work on blocks of data (such as integers) rather than single bits (as in Gahi's scheme), the number of computations can be greatly reduced.

In addition to this goal, we also look at recent approaches in integrating homomorphic encryption schemes for aggregating measurements in the smart grid. The "smart electric grid" can be defined as the next generation electric grid network that uses information technology to deliver electricity efficiently, reliably, and securely. Specifically we summarize the methods introduced by Garcia [21], Kursawe [31], Erkin [16] and Ács [3]. We then discuss methods of processing queries over encrypted data in the smart grid and propose a new keyword search scheme based on our Homomorphic Query Processing method. The proposed method also uses Public Key Encryption with Keyword Search (PEKS), introduced by Boneh et al. [9], and Multi-Key Homomorphic Encryption introduced by López-Alt et al [33]. This scheme can be used along with the previously mentioned aggregating schemes to make smart meter data encrypted as well as queryable. Finally, we conclude with a brief discussion of the challenges faced by each proposed protocol and future research directions.

1.2 Contribution and Organization

In the first part of this thesis, we focus on improving a specific database query protocol suggested by Gahi et al [19]. This scheme uses the DGHV fully homomorphic encryption scheme to compute encrypted database queries. The chapters are organized as follows. In Chapter 2, we summarize the background material, discuss the notions of full and partial homomorphisms, and formally define what it means to be fully homomorphic. In Chapter 3, we look closely at the integer based fully homomorphic encryption scheme (commonly known as the DGHV scheme) introduced by Dijk et al. [15], which is a conceptually simpler version of Gentry's original blueprint. We also introduce Gentry's idea of "bootstrapping", which transforms a somewhat homomorphic encryption scheme (such as the DGHV scheme) to a fully homomorphic one. The DGHV scheme has a wide variety of practical applications including location based privacy [20], database security [19], and privacy preserving in healthcare applications [45], among many others. Chapter 4 gives a detailed description of the models proposed by Gahi et al. [19, 20] to preserve location and database privacy using the DGHV scheme. In Chapter 5, we introduce our new protocol, Homomorphic Query Processing method, to process database queries. Our protocol does not rely on the DGHV scheme and can be used with more modern fully homomorphic encryption schemes like the ring based fully homomorphic encryption scheme proposed by Braserski et al [12]. In Chapter 6, we summarize recent integrations of homomorphic encryption schemes to preserve privacy in aggregation calculations in the smart grid. In Chapter 7, we discuss querying data in the smart electric grid and propose a new scheme that can be used to query smart meter data with respect to a given set of keywords. Finally, in Chapter 8, we conclude with a discussion of challenges faced by each proposed aggregation protocol, as well as our proposed schemes and future research directions.

A summary of results of this thesis (specifically the Homomorphic Query Processing scheme) is published at the 14th Canadian Workshop on Information Theory (CWIT) [39].

Chapter 2

Background

In this section, we present the background material needed to study the DGHV encryption scheme and Gahi's protocol. We start with an introduction to fully and partially homomorphic encryption schemes. Then, we define what it means to be fully homomorphic based on the definition proposed by Rivest et al [42]. We also give a high level overview of the construction of a fully homomorphic encryption scheme, as proposed by Gentry [22].

2.1 Homomorphic Encryption

Homomorphic encryption is a novel method that allows computations to be carried out on the ciphertext such that after decryption, the result would be the same as carrying out identical computations on the plaintext. This has novel implications such as being able to carry out operations on database queries in the form of ciphertext and returning the result to the user so that no information about the query is revealed at the server's end [10].

The idea of homomorphic encryptions is not new, and even the oldest of ciphers, ROT13 developed in ancient Rome, had homomorphic properties with respect to string concatenations [26]. Certain modern ciphers such as RSA and El Gamal also support homomorphic multiplication of cipher texts [26]. The idea of a "fully" homomorphic encryption scheme (or *privacy homomorphism*) which supports two homomorphic operations was first introduced by Rivest, Adleman, and Dertouzous in 1978 [42]. After more than three decades, the first fully homomorphic encryption scheme was founded by Gentry in 2009 with his breakthrough construction of a lattice based cryptosystem that supports both homomorphic additions and multiplications [22]. Although the lattice based system is not used in practice, it paved the way for many other simpler and more efficient fully homomorphic models constructed afterwards.

2.2 Fully and Partially Homomorphic Encryption Schemes

Homomorphic cryptosystems can be broadly categorized into fully homomorphic and partially homomorphic schemes (Figure 2.1). Partially homomorphic encryption schemes are homomorphic with respect to one operation such as multiplication in the case of the RSA cryptosystem. A schematic diagram of an additive homomorphic encryption scheme is given in Figure 2.2. Here, we have an encryption scheme (denoted by Enc) that outputs two ciphertexts c_1 and c_2 when operated on the two plaintexts x_1 and x_2 (with key k), respectively. Since the scheme is additively homomorphic, we have the following relation between the plaintexts and ciphertexts.

$$\text{Enc}_k(x_1) + \text{Enc}_k(x_2) = \text{Enc}_k(x_1 + x_2)$$

Fully homomorphic encryption schemes, on the other hand, support two operations (addition and multiplication). Up to date, there is quite a number of partially homomorphic encryption schemes that are well understood and practically used. However, the field of fully homomorphic encryption is quite recent, and the currently available fully homomorphic encryption schemes are mostly proof of concept and can only

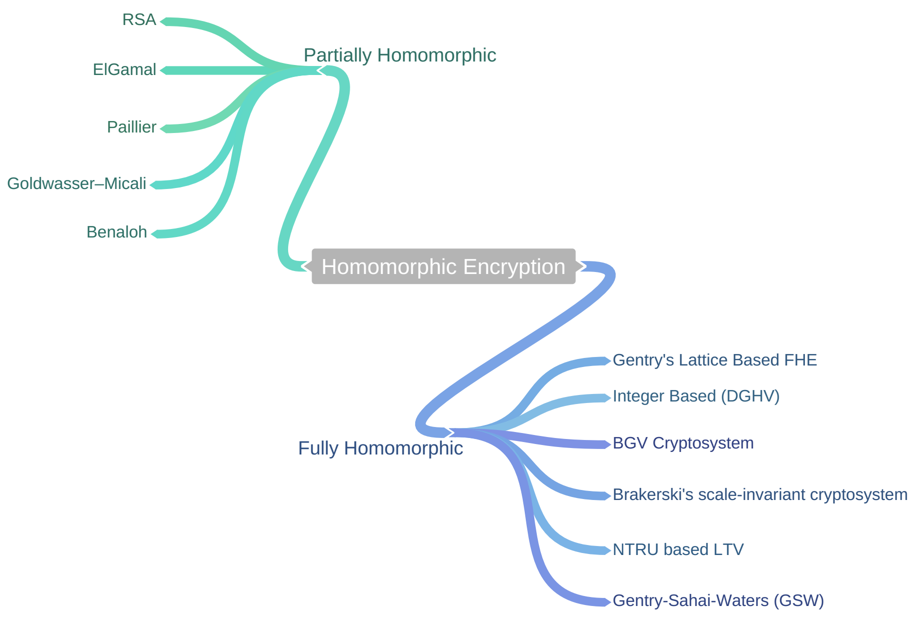

Figure 2.1: Fully and Partially Homomorphic Schemes

be carried out with respect to a small number of operations or with very high end laboratory settings [36]. One of the main reasons for the impracticality of fully homomorphic encryption schemes is that the ciphertext size and the computation time increases rapidly as the security level increases.

As much as we would like to have a cryptosystem that is homomorphic with respect to one operation, the application of such a scheme is limited by the fact that not every computation of our liking can be performed with only one operation. For example, to find the mean or average value of a sample, we would need a homomorphic encryption scheme that supports one multiplication at the very least. Moreover, the

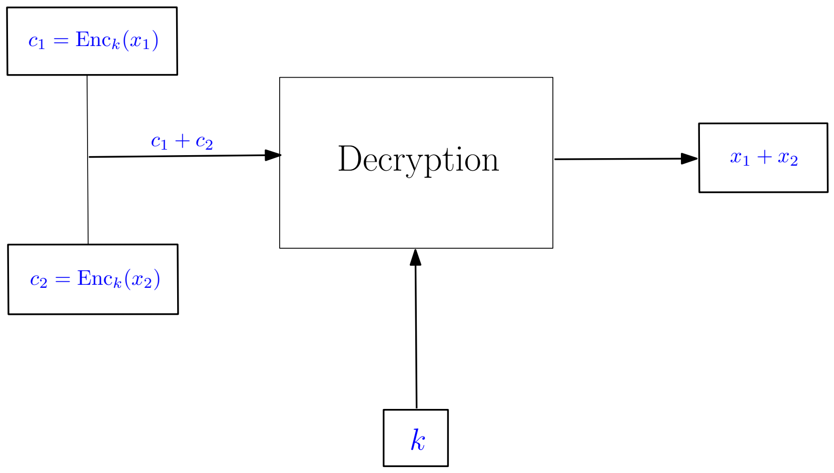

Figure 2.2: Additive Homomorphic Scheme

fact that any computation on a computational device (computers, calculators, etc.) can be designed through a Boolean circuit (a circuit that consists of addition and multiplication gates), makes fully homomorphic encryption schemes highly appealing. With a fully homomorphic encryption scheme, we can outsource computations to cloud with reasonable guarantee (upto the hardness of breaking the scheme) that the computations will be performed blindly. For example, consider the case where two parties, Alice and Bob, want to communicate with each other (Figure 2.3). Alice would like to send Bob three numbers x_1, x_2, and x_3, and get the result of a calculation performed by Bob. However, Alice does not want Bob to know the three numbers she sends. Therefore, she would like to encrypt these numbers using an appropriate encryption scheme before sending it to Bob. However, Alice also want Bob to be able to perform the calculation even though the numbers are encrypted. This problem is solved using a homomorphic encryption scheme. Alice encrypts her numbers using a homomorphic encryption scheme that is then sent to Bob. Bob can perform his calculation on the encrypted numbers as though they are not encrypted. However, Bob will only see the ciphertexts and not the actual numbers x_1, x_2, and x_3. Finally, Bob will send the result of the calculation back to Alice to decrypt and obtain the result. This process is illustrated in Figure 2.3. Currently, it is common practice to decrypt the data at the server-side before doing any calculation. However, this places the user's privacy in jeopardy because the user is forced to assume that the server is trusted. Hence, using a fully homomorphic encryption scheme is the key to successfully ensuring many security and privacy assumptions that are prevalent in current communication protocols.

We will first look at the RSA cryptosystem as a typical example of a partially homomorphic encryption scheme.

2.3 Partial Homomorphism - RSA Cryptosystem

The RSA Cryptosystem was introduced by Ron Rivest, Adi Shamir, and Leonard Adleman in 1977. Although not intentionally introduced as a homomorphic encryp-

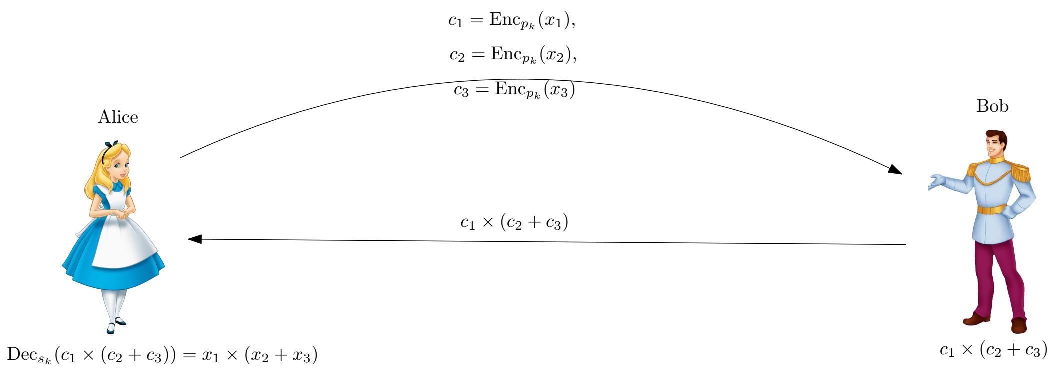
$c_1 = \mathrm{Enc}_{p_k}(x_1)$,
$c_2 = \mathrm{Enc}_{p_k}(x_2)$,
$c_3 = \mathrm{Enc}_{p_k}(x_3)$
Alice
Bob
$c_1 \times (c_2 + c_3)$
$\mathrm{Dec}_{s_k}(c_1 \times (c_2 + c_3)) = x_1 \times (x_2 + x_3)$
$c_1 \times (c_2 + c_3)$

Figure 2.3: Alice, Bob, and Fully Homomorphic Encryption

tion scheme, the RSA scheme shows homomorphic qualities with respect to multiplication. The RSA scheme operates as follows:

- **KeyGen:** Select two random large prime numbers, p and q. Compute their product, $n = pq$. Choose e such that $1 < e < \phi(n)$ and $\gcd(e, \phi(n)) = 1$, where ϕ is the Euler's Totient function. Compute d such that $ed \equiv 1 (\text{mod } \phi(n))$.

$$\text{Public Key: } (e, n), \text{ Private Key: } (d, \phi(n))$$

- **Enc:** For a message m, compute $c = m^e$ (mod n).

- **Dec:** For a ciphertext c, compute $m = c^d$ (mod n).

where **KeyGen**, **Enc**, and **Dec** denote the key generation, encryption, and decryption algorithms, respectively. Now, consider two plaintexts x_1 and x_2 encrypted by the RSA scheme.

$$Enc(x_1) = x_1^e \pmod{n} \text{ and } Enc(x_2) = x_2^e \pmod{n}$$

Therefore,

$$Enc(x_1) \times Enc(x_2) = x_1^e \times x_2^e \pmod{n} = (x_1 x_2)^e \pmod{n} = Enc(x_1 \times x_2)$$

Therefore, the RSA scheme is clearly homomorphic with respect to multiplication.

2.4 Fully Homomorphism

The importance of fully homomorphic encryption schemes and the question of whether such schemes actually exist was first proposed by Ronald Rivest, Len Adleman, and Michael Dertouzos in 1978, immediately following the introduction of the RSA cryptosystem [42]. In their own words the question statement was as follows.

> "A scheme $\mathcal{E}$ with an efficient algorithm Evaluate$_{\mathcal{E}}$ such that, for any valid public key p_k, any circuit C, and any ciphertexts $\psi_i \leftarrow \text{Encrypt}_{\mathcal{E}}(p_k, \pi_i)$ outputs,
>
> $$\psi \leftarrow \text{Evaluate}_{\mathcal{E}}(p_k, C, \psi_1, \ldots, \psi_t)$$
>
> where ψ is a valid encryption of $C(\pi_1, \ldots, \pi_t)$ under p_k."

Here, π_i denotes the plaintext. For more than three decades, it was an open problem whether such schemes exist. However, as we will see in the next chapter, this problem was answered by Craig Gentry in 2009.

2.4.1 Overview of Fully Homomorphic Encryption

At a high level, Gentry's idea can be described by the following general model. This is the blueprint that is used in all homomorphic encryption schemes that followed.

1. Develop a *Somewhat Homomorphic Encryption Scheme* that is restricted to evaluating a finite number of additions or multiplications. In other words, the somewhat homomorphic encryption scheme that we construct is only limited to evaluating low-degree polynomials.

2. Modify the somewhat homomorphic encryption scheme to make it *Bootstrappable*, that is, modifying it so that it could evaluate its own decryption circuit plus at least one additional NAND gate.

The somewhat homomorphic encryption scheme usually introduces a *noise* whenever a homomorphic operation is carried out, and when the noise exceeds a certain threshold, the scheme loses its homomorphic ability. The idea behind constructing a bootstrappable scheme is that whenever the noise level is about to reach the threshold, we can *bootstrap* it so that the resulting ciphertext will give the same encrypted value but with a lower noise. In this way, if the ciphertext is bootstrapped from time to time, an arbitrary number of operations can be carried out.

In the next chapter, we will define a fully homomorphic encryption scheme commonly known as the DGHV scheme, which is used to process database queries securely.

Chapter 3

Integer Based Fully Homomorphic Encryption (DGHV Scheme)

3.1 DGHV Scheme

The DGHV scheme was introduced by Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan in 2010, and this scheme operates on integers as opposed to lattices in Gentry's original construction. The scheme follows Gentry's original blueprint by first constructing a somewhat homomorphic encryption scheme. The key generation, encryption and decryption algorithms of the DGHV scheme are given below.

Let $\lambda \in \mathbb{N}$ be the security parameter and set $N = \lambda$, $P = \lambda^2$ and $Q = \lambda^5$. The scheme is based on the following algorithms;

- **KeyGen(**λ**)**: The key generation algorithm that randomly chooses a P-bit integer p as the secret key.

- **Enc(**m, p**)**: The bit $m \in \{0, 1\}$ is encrypted by

$$c = m' + pq,$$

  where $m' \equiv m \pmod 2$ and q, m' are random Q-bit and N-bit numbers, respectively. Note that we can also write the ciphertext as $c = m + 2r + pq$ since $m' = m + 2r$ for some $r \in \mathbb{Z}$.

- **Dec(**c, p**)**: Output (c mod p) mod 2 where (c mod p) is the integer c' in $(-p/2, p/2)$ such that p divides $c - c'$.

The value m' is called the *noise* of the ciphertext. Note that this scheme, as it is given above, is symmetric (i.e., it only has a private key). We can define the public key as a random subset sum of encryptions of zeros, that is, the public key is a randomly choosen sum from a predefined set of encryptions of zeros: $S = \{2r_1 + pq_1, 2r_2 +$

$pq_2, \dots, 2r_n + pq_n\}$. A typical encryption of the plaintext m would be,

$$\begin{aligned} c &= m + \sum_{i \in T}(2r_i + pq_i) \\ &= m + 2\sum_{i \in T} r_i + p\sum_{i \in T} q_i, \end{aligned}$$

where $T \subseteq S$. From here on we shall use m' to denote $m + \sum_{i \in T} r_i$ and q to denote $\sum_{i \in T} q_i$.

This scheme is homomorphic with respect to addition and multiplication and decrypts correctly as long as the noise level does not exceed $p/2$ in absolute value. That is, $|m'| < p/2$. Let c_1 and c_2 be two ciphertexts obtained using the DGHV scheme, that is,

$$c_1 = m'_1 + pq_1 \text{ and } c_1 = m'_2 + pq_2,$$

where $m'_1 = m_1(\text{mod } 2)$ and $m'_2 = m_2(\text{mod } 2)$. Addition of c_1 and c_2 results in,

$$c_1 + c_2 = (m'_1 + m'_2) + p(q_1 + q_2).$$

Thus,

$$\text{Enc}(m_1, p) + \text{Enc}(m_2, p) = \text{Enc}(m_1 \oplus m_2, p).$$

Decryption works as long as,

$$|m'_1 + m'_2| < \frac{p}{2}.$$

Similarly, multiplying c_1 and c_2 results in,

$$\begin{aligned} c_1c_2 &= (m'_1 + pq_1)(m'_2 + pq_2) \\ &= m'_1m'_2 + p(m'_1q_2 + m'_2q_1 + pq_1q_2) \end{aligned}$$

Hence,

$$\text{Enc}(m_1, p) \times \text{Enc}(m_2, p) = \text{Enc}(m_1 \times m_2, p)$$

Here, decryption works as long as,

$$|m'_1m'_2| < \frac{p}{2}.$$

Hence, this is a somewhat homomorphic encryption scheme in the sense that once the noise level exceeds $p/2$, the scheme loses its homomorphic ability. In other words, this scheme can evaluate any function that is composed of a collection of additions and multiplications as long as the $p/2$ threshold is not reached. Since any computation can be expressed as a collection of additions and multiplications (i.e., a Boolean Circuit), this would imply that extending this scheme to perform an arbitrary amount of additions and multiplications would allow us to compute any Boolean function without decrypting the ciphertext. This is the breakthrough idea of Gentry called Bootstrapping, discussed in the next section.

We shall illustrate the DGHV scheme mentioned previously through a small example. Let $N = \lambda = 2$. Then, $P = \lambda^2 = 4$ and $Q = \lambda^5 = 32$. Suppose the plaintext bit that we would like to encrypt is 1. Choose $p = 10$ as the secret key, $q = 15$ and $m' = 3$ so that,

$$c = 3 + (10 \times 15) = 153.$$

Thus the ciphertext for this particular choice of parameters is 153. The decryption is given by,

$$(c \bmod p) \bmod 2 = (153 \bmod 10) \bmod 2 = 3 \bmod 2 = 1.$$

And therefore we can see that decryption works as expected.

3.2 Bootstrapping

Suppose we have two ciphertexts c_1 and c_2 encrypted using the DGHV scheme. That is,

$$c_1 \leftarrow \text{Enc}(m_1, pk_1) \text{ and } c_2 \leftarrow \text{Enc}(m_2, pk_1)$$

where $m_1, m_2 \in \{0, 1\}$ are the corresponding plaintexts and pk_1 is the public key that has the corresponding secret key sk_1. Encrypt the secret key sk_1 by a second public key pk_2. We will denote the vector of encrypted bits of sk_1 by $\overline{sk_1}$. That is each bit sk_{1i} of sk_1 is encrypted by the DGHV scheme:

$$\overline{sk_{1i}} \leftarrow \text{Enc}(sk_{1i}, pk_2)$$

Denote the decryption function augmented by an addition as Dec_{Add} and the decryption function augmented by a multiplication as Dec_{Mult}. The two functions Dec_{Add} and Dec_{Mult} can be defined as follows:

$$\mathrm{Dec}_{Add}(c_1, c_2) = m_1 + m_2$$

and

$$\mathrm{Dec}_{Mul}(c_1, c_2) = m_1 m_2$$

Encrypt each ciphertext c_1 and c_2 by the second public key pk_2. Since c_1 and c_2 are integers we have to perform the encryption over each bit separately as before.

$$\overline{c_{1i}} \leftarrow \mathrm{Enc}(c_{1i}, pk_2) \text{ and } \overline{c_{2i}} \leftarrow \mathrm{Enc}(c_{2i}, pk_2)$$

Finally output,

$$c \leftarrow \mathrm{Evaluate}(pk_2, \mathrm{Dec}_{Add}, \overline{sk_1}, \overline{c_1}, \overline{c_2}),$$

where Evaluate denotes evaluating the Dec_{Add} function homomorphically (in other words, evaluating the Boolean circuit corresponding to the Dec_{Add} function). Note that $\overline{sk_1}, \overline{c_1}$, and $\overline{c_2}$ are all encrypted by pk_2. Since the Evaluate algorithm can evaluate Dec_{Add}, it will take the cipher texts $\overline{c_1}, \overline{c_2}$, and $\overline{sk_1}$ and evaluate the function Dec_{Add}

homomorphically. Therefore,

$$\begin{aligned} c &= c_1 + c_2 \\ &= \text{Enc}(m_1, pk_1) + \text{Enc}(m_2, pk_1) \\ &= \text{Enc}(m_1 \oplus m_2, pk_1) \end{aligned}$$

Similarly the function Dec_{Mul} can be evaluated homomorphically to obtain,

$$c' \leftarrow \text{Evaluate}(pk_2, \text{Dec}_{Mult}, \overline{sk_1}, \overline{c_1}, \overline{c_2}),$$

where $c' = \text{Enc}(m_1 m_2, pk_1)$. Hence, whenever the noise level is about to reach its threshold value (i.e. $p/2$) we can perform a homomorphic decryption on the ciphertext, which will give us a re-encryption of the same ciphertext with a lower noise level. Thus, any arbitrary function (precisely a Boolean function) can be evaluated by bootstrapping the ciphertext periodically.

In the next chapter, we look at how the DGHV scheme can be used to process queries in a database. This is useful when queries have to be processed secretly such that only the user knows what he is searching for. For example, a typical application of this kind of scheme would be a medical database, where the database should not know what the user is searching for in order to preserve privacy of the users.

Chapter 4

Query Processing Using the DGHV Scheme

The DGHV scheme can be used to create a protocol that establishes blind searching in databases. This method was proposed by Gahi et al. [19].

Suppose we need to retrieve a particular record(s) from the database. Typically, we send a query to the database encrypted using the DGHV scheme. Let v_i be the i^{th} bit of the query v and c_i be the i^{th} bit of a record R in database D. Both the query and the database record is encrypted using the DGHV scheme. Suppose the plaintext bit corresponding to v_i is m_i and the plaintext bit corresponding to c_i is m'_i.

Then,

$$v_i = m_i + 2r_i + pq_i$$

and

$$c_i = m_i' + 2r_i' + pq_i',$$

where r_i, r_i', q_i and q_i' are random numbers and p is the secret key. The server shall compute the following sum for each record:

$$I_r = \prod_i (1 + c_i + v_i), \text{ for every } R \in D, \tag{4.1}$$

where r denotes the index of record R. If $m_i = m_i'$, then $m_i + m_i' = 0$ or $m_i + m_i' = 2$ since $m_i, m_i' \in \{0, 1\}$:

$$\begin{aligned} 1 + c_i + v_i &= 1 + (m_i' + 2r_i' + pq_i') + (m_i + 2r_i + pq_i) \\ &= 1 + (m_i + m_i') + 2(r_i + r_i') + p(q_i + q_i') \end{aligned}$$

Case 1: $m_i + m_i' = 0$,

$$\begin{aligned} 1 + c_i + v_i &= 1 + 2(r_i + r_i') + p(q_i + q_i') \\ &= \text{Enc}(1) \end{aligned}$$

Case 2: $m_i + m_i' = 2$,

$$\begin{aligned} 1 + c_i + v_i &= 1 + 2(1 + r_i + r_i') + p(q_i + q_i') \\ &= \text{Enc}(1) \end{aligned}$$

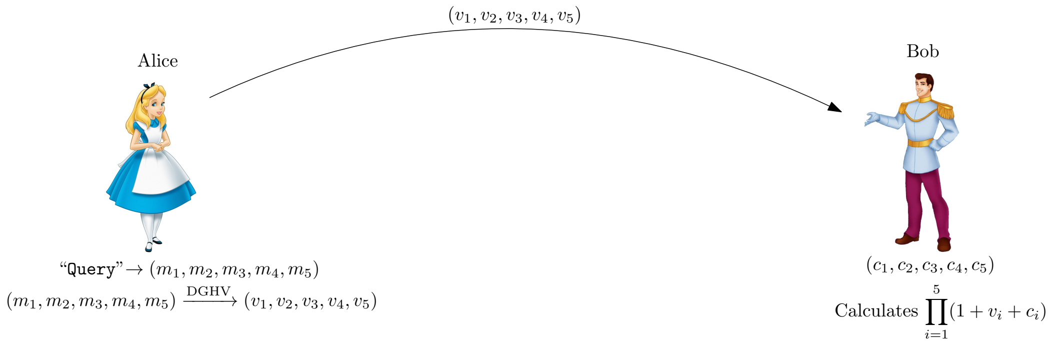
$(v_1, v_2, v_3, v_4, v_5)$
Alice
Bob
"Query"$\rightarrow (m_1, m_2, m_3, m_4, m_5)$
$(m_1, m_2, m_3, m_4, m_5) \xrightarrow{\text{DGHV}} (v_1, v_2, v_3, v_4, v_5)$
$(c_1, c_2, c_3, c_4, c_5)$
Calculates $\prod_{i=1}^{5}(1 + v_i + c_i)$

Figure 4.1: Calculation of I_r values

This results in $I_r = \text{Enc}(1)$. On the other hand, if $m_i \neq m'_i$, then $m_i + m'_i = 1$ since $m_i, m'_i \in \{0, 1\}$. Therefore,

$$\begin{aligned} 1 + c_i + v_i &= 1 + (m'_i + 2r'_i + pq'_i) + (m_i + 2r_i + pq_i) \\ &= 1 + (m_i + m'_i) + 2(r_i + r'_i) + p(q_i + q'_i) \\ &= 1 + 1 + 2(r_i + r'_i) + p(q_i + q'_i) \\ &= 2(1 + r_i + r'_i) + p(q_i + q'_i) \\ &= \text{Enc}(0) \end{aligned}$$

This results in $I_r = \text{Enc}(0)$. Hence, for each record in the database we will have an I_r value that is equal to Enc(1) or Enc(0) depending on whether the search query matches the corresponding record or not (Figure 4.1). Secondly, we calculate the partial sums of the I_r values (Table 4.1).

Table 4.1: Example database and corresponding I_r and S_r values

Database Records	I_r	S_r
$(1,1,0,0)$	Enc(1)	Enc (1)
$(1,0,1,0)$	Enc(0)	Enc (1)
$(1,1,0,0)$	Enc(1)	Enc (2)
$(1,1,0,1)$	Enc(0)	Enc (2)
$(1,0,0,0)$	Enc(0)	Enc (2)

$$S_r = \sum_{i \leq r} I_i. \tag{4.2}$$

As an example, let us consider a database that has five records, each encoded with 4 bits. If the query sent by the user is (Enc(1), Enc(1), Enc(0), Enc(0)), we obtain the corresponding I_r and S_r values, as shown in Table 4.1. Using these partial sums, we can then calculate the sequence $(I'_{r,j})$ for every record R with index r and every positive integer $j \leq r$:

$$I'_{r,j} = I_r \prod_i (1 + \bar{j}_i + S_{r,i}) \text{ for every } R \in D, \tag{4.3}$$

where $S_{r,i}$ is the i^{th} bit of S_r and $\bar{j}_i$ represents the i^{th} bit of the encryption of j, where $j \leq r$. Hence, these sequences have the property that whenever $I_r = \text{Enc}(1)$ and $S_r = \text{Enc}(j)$, we have $I'_{r,j} = \text{Enc}(1)$. Otherwise, $I'_{r,j} = \text{Enc}(0)$. In our previous example, we would have,

$$(I'_1) = (\mathrm{Enc}(1))$$

$$(I'_2) = (\mathrm{Enc}(0), \mathrm{Enc}(0))$$

$$(I'_3) = (\mathrm{Enc}(0), \mathrm{Enc}(1), \mathrm{Enc}(0))$$

$$(I'_4) = (\mathrm{Enc}(0), \mathrm{Enc}(0), \mathrm{Enc}(0), \mathrm{Enc}(0))$$

$$(I'_5) = (\mathrm{Enc}(0), \mathrm{Enc}(0), \mathrm{Enc}(0), \mathrm{Enc}(0), \mathrm{Enc}(0))$$

Finally, we calculate,

$$(R') = \sum_k \mathrm{Enc}(R_k)(I'_k), \tag{4.4}$$

where R_k is the k^{th} record in D. So, (R') is a sequence containing only the encrypted records that matches our search query. Note that the definition of (R') relies on adding vectors of different lengths. This is done in the natural way, whereby all the vectors are made the same length by padding with zeros prior to addition. In the above example, we obtain,

$$(R') = (\mathrm{Enc}(R_1), \mathrm{Enc}(R_3), \mathrm{Enc}(0), \mathrm{Enc}(0))$$

At this point the sequence (R') will contain all the records that match our query, but with trailing encryptions of zeros we do not need. Hence, a second sum is calculated

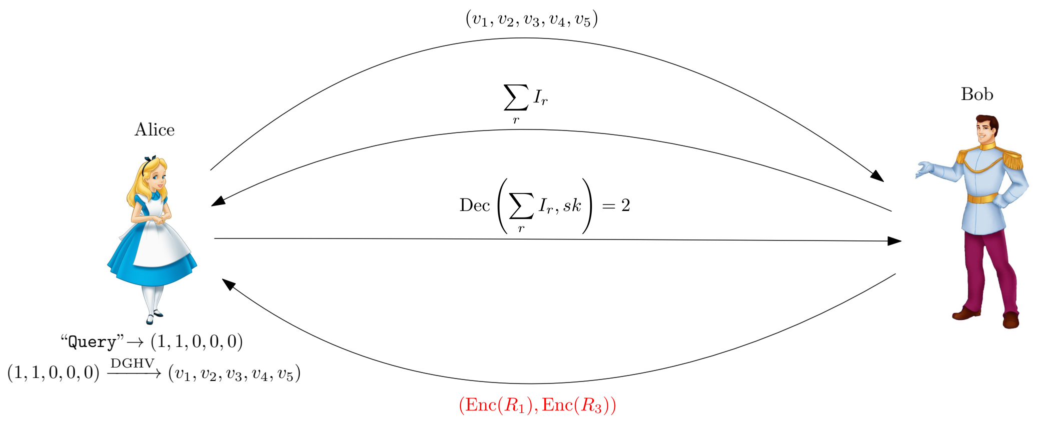
$(v_1, v_2, v_3, v_4, v_5)$
$\sum_r I_r$
Bob
Alice
$\mathrm{Dec}\left(\sum_r I_r, sk\right) = 2$
"Query"$\rightarrow (1,1,0,0,0)$
$(1,1,0,0,0) \xrightarrow{\mathrm{DGHV}} (v_1, v_2, v_3, v_4, v_5)$
$(\mathrm{Enc}(R_1), \mathrm{Enc}(R_3))$

Figure 4.2: Alice, Bob, and Gahi's Protocol

at the server side to determine the number of terms that are useful in the sequence:

$$n = \sum_r I_r$$

This result can be returned to the user and decrypted to obtain the number of records that match the search query. Hence, the sequence (R') can be truncated at the appropriate point and returned to the user for decryption. The whole process is illustrated in Figure 4.2. An update query can be performed by,

$$R_{new} = (1 + I_r)R + I_r U, \text{ for every } R \in D,$$

where U is the new value that we wish to insert whenever the query matches R (or $I_r = \mathrm{Enc}(1)$). A deletion of a record can be performed by,

$$R_{new} = (1 + I_r)R \text{ for every } R \in D.$$

To perform all these operations without exceeding the maximum noise permitted $(p/2)$, it is necessary to choose the parameters $N, P,$ and Q appropriately.

Remark

Although according to Gahi's original paper [19] both the query and the database records are encrypted when calculating I_r in equation 4.1, we note that it is not strictly necessary. We show below that only encrypting the query is sufficient. Let v_i be the i^{th} bit of the query v, and let the plaintext bit corresponding to v_i is m_i. Now suppose only the query is encrypted:

$$v_i = m_i + 2r_i + pq_i,$$

where r_i, q_i are random numbers and p is the encryption key. Let c_i be the i^{th} bit of the database record. Now, the database is not encrypted and therefore if $c_i = m_i$, then $c_i + m_i = 0$ or $c_i + m_i = 2$.

Case 1: $c_i + m_i = 0$,

$$\begin{aligned} 1 + c_i + v_i &= 1 + c_i + (m_i + 2r_i + pq_i) \\ &= 1 + 2r_i + pq_i \\ &= \text{Enc}(1) \end{aligned}$$

Case 2: $c_i + m_i = 2$,

$$\begin{aligned} 1 + c_i + v_i &= 1 + c_i + (m_i + 2r_i + pq_i) \\ &= 1 + 2(1 + r_i) + pq_i \\ &= \text{Enc}(1) \end{aligned}$$

On the other hand, if $c_i \neq m_i$ then $c_i + m_i = 1$, since $c_i, m_i \in \{0, 1\}$,

$$\begin{aligned} 1 + c_i + v_i &= 1 + c_i + (m_i + 2r_i + pq_i) \\ &= 1 + (c_i + m_i) + 2r_i + pq_i \\ &= 2(1 + r_i) + pq_i \\ &= \text{Enc}(0) \end{aligned}$$

Therefore, we note that for the purpose of equation 4.1, the database records need not be encrypted.

Gahi's method works on plaintext bits and thus requires significant computational ability on the part of the server. This is due to the fact that it is restricted to the DGHV scheme which processes plaintext bits separately. In the next chapter we propose an alternative protocol called the Homomorphic Query Processing Scheme. This protocol enables us to process database queries using more modern fully homomorphic encryption schemes such as the ring based scheme proposed by Braserski et al. [12], which act on blocks of plaintext rather than single bits.

Chapter 5

Homomorphic Query Processing

The main drawback in Gahi's method is that it requires an enormous number of homomorphic operations because it employs the DGHV encryption scheme, which uses bitwise encryption. We propose an alternative protocol called *Homomorphic Query Processing* that is compatible with the more recent ring-based fully homomorphic encryption scheme introduced by Braserski et al. [12]. The major advantage is that Braserski's method works on plaintext and ciphertext blocks and thus the number of homomorphic operations required can be greatly reduced.

We first give a brief introduction to the ring based fully homomorphic Encryption Scheme proposed by Braserski, and then proceed to define our Homomorphic Query Processing method.

5.1 Ring Based Fully Homomorphic Encryption

This encryption scheme was introduced by Braserski, et al [12] and operates on the polynomial ring $R = \mathbb{Z}[X]/\langle f(x)\rangle$; the ring of polynomials with integer coefficients modulo $f(x)$, where $f(x) = \prod_{\substack{1\leq k\leq n \\ \gcd(k,n)=1}} \left(x - e^{2i\pi\frac{k}{n}}\right)$ is the n^{th} cyclomatic polynomial. The plaintext is defined on the ring $R_t = \mathbb{Z}_t[x]/\langle f(x)\rangle$, where t is an integer. The key generation and encryption functions make use of two distributions χ_{key} and χ_{err} on R for generating small elements. The uniform distribution χ_{key} is used in the key generation, and the discrete Gaussian distribution χ_{err} is used to sample small noise polynomials. Specific details can be found in [11] and [12]. The scheme is based on the following algorithms.

- **KeyGen**$(n, q, t, \chi_{key}, \chi_{err})$: Operating on the input degree n and moduli q and t, this algorithm generates the public and private keys $(pk, sk) = (h, f)$, where $f = [tf' + 1]_q$ and $h = [tgf^{-1}]_q$. Here, the key generation algorithm samples small polynomials from the key distribution $f', g \rightarrow \chi_{key}$ such that f is invertible modulo q and $[.]_q$ denotes coefficients of polynomials in R reduced by modulo q.

- **Encrypt**(h, m): Given a message $m \in R$, the Encrypt algorithm samples small error polynomials $s, e \rightarrow \chi_{err}$ and outputs, $c = [\lfloor q/t \rfloor [m]_t + e + hs]_q \in R$, where

$\lfloor . \rfloor$ denotes the floor function.

- **Decrypt**(f, c): Given a ciphertext c, this algorithm outputs, $m = \left[\left\lfloor \frac{t}{q}[fc]_q \right\rceil\right]_t \in R$.

- **Add**(c_1, c_2): Given two ciphertexts c_1 and c_2, this algorithm outputs $c_{add}(c_1, c_2) = [c_1 + c_2]_q$.

- **Mult**(c_1, c_2): Multiplication of ciphertexts is performed in two steps. First, compute $\widetilde{c}_{mult} = \left[\left\lfloor \frac{t}{q} c_1 c_2 \right\rceil\right]_q$. However, this result cannot be decrypted to the original plaintext using the decryption key f. Therefore, a process known as key switching is done to transform the ciphertext so that it can be decrypted with the original secret key. For more details, we refer to [11].

This encryption scheme is homomorphic with respect to addition and multiplication of plaintexts modulo t. The main advantage in using Braserski's encryption scheme is that it can be used to encrypt blocks of plaintext instead of dealing with single bits, as in the DGHV scheme [15]. For example, consider the block of plaintext bits, 10100. The integer representation of this block is the value 20. We can represent this integer using the polynomial $X^2 + X^4 = \sum_{i=0}^{4} 2^i z_i$, where z_i is the i^{th} bit of 10100. In general, if z is an integer and its binary representation is, $z = (\pm 1) \sum_{i=0}^{l} 2^i z_i$, where $z_i \in \{0, 1\}$ and $l = \lceil \log_2 |z| \rceil$, then we can encode the number z as $\sum_{i=0}^{l} z_i X^i \in R$.

5.2 Converting the Plaintext Space into a Field

As we shall see, in our *Homomorphic Query Processing* method, we invert certain plaintext elements and thus the plaintext space should be a field. Therefore, we now discuss how to convert the plaintext ring in Braserski's method to a field. Note that the plaintext space in Braserski's method is defined on the polynomial ring, $R_t = \mathbb{Z}_t[x]/\langle f(x)\rangle$. We shall select $t = p$, where p is a prime number. Then R_p is a field if and only if f is irreducible over $\mathbb{Z}_p$. Recall that f is the n^{th} cyclomatic polynomial defined as follows:

$$f(x) = \prod_{\substack{1\leq k\leq n\\ \gcd(k,n)=1}} \left(x - e^{2i\pi\frac{k}{n}}\right)$$

Let $f(x) = (x-\alpha_1)(x-\alpha_2)\dots(x-\alpha_n)$ be a polynomial defined on $\mathbb{Q}[x]$. The discriminant of f, denoted by $\Delta(f)$, is defined [25] as,

$$\Delta(f) = \prod_{i<j}(\alpha_i - \alpha_j)^2$$

It has been proved in [25] that the n^{th} cyclotomic polynomial reduces modulo all primes if and only if the discriminant of the n^{th} cyclotomic polynomial is a square in $\mathbb{Z}$. Hence, by choosing a cyclotomic polynomial whose discriminant is not a square we can find a prime p such that f is irreducible over $\mathbb{Z}_p$. Furthermore, it is shown in [25] that whenever the discriminant of a cyclotomic polynomial f is not a square in $\mathbb{Z}$, there exist infinitely many primes such that f is irreducible over $\mathbb{Z}_p$. Thus, we can choose a cyclotomic polynomial with non-square discriminant and check for irreducibility using a standard polynomial irreducibility test such as Rabin's test, until we obtain a prime for which the cyclotomic polynomial is irreducible. For example, even if we consider a large cyclotomic polynomial with non-square discriminant like the 107th cyclotomic (which has degree 106), and consider the primes less than 100, it can be seen that it is irreducible over many primes: $\mathbb{Z}_2, \mathbb{Z}_5, \mathbb{Z}_7, \mathbb{Z}_{17}, \mathbb{Z}_{31}, \mathbb{Z}_{43}, \mathbb{Z}_{59}, \mathbb{Z}_{67}, \mathbb{Z}_{71}, \mathbb{Z}_{73}$ and $\mathbb{Z}_{97}$.

We now propose our Homomorphic Query Processing scheme, which is compatible with the Braserski's ring based fully homomorphic encryption scheme mentioned previously.

5.3 Homomorphic Query Processing

We begin by defining the value F_i for the i^{th} record (denoted by R_i) in the database. We write $\text{Enc}(m)$ for the $\text{Enc}(m, pk)$, where pk is the public key of the user. Then the user sends $\text{Enc}(m)$ to the server to search for the records that match m. Now, the server computes the following:

$$F_i = \left(\prod_{R_k \neq R_i} \text{Enc}(m - R_k) \right) \left(\prod_{R_k \neq R_i} \text{Enc}\left(R_i - R_k\right)^{-1} \right), \tag{5.1}$$

where each of the products above is over all the records R_k such that $R_k \neq R_i$. Since we are dealing with a fully homomorphic encryption scheme, we can compute $\text{Enc}(m - R_k)$ values by computing $\text{Enc}(m) - \text{Enc}(R_k)$. Also, since all the R_i values are known to the server, the term $\prod_{R_k \neq R_i} \text{Enc}\left(R_i - R_k\right)^{-1}$ can be reduced to a simpler form using the homomorphic property of the encryption scheme in order to perform a single encryption. Hence,

$$F_i = \left(\prod_{R_k \neq R_i} \text{Enc}(m - R_k)\right) \text{Enc}\left(\prod_{R_k \neq R_i} (R_i - R_k)\right)^{-1}.$$

Note that whenever $m = R_i$ (query being equal to the record we are comparing), we have $F_i = \text{Enc}(1)$ and $F_i = \text{Enc}(0)$, otherwise. Note that here we are assuming that the query is contained somewhere in the database. If the query is not contained anywhere in the database, an encryption of something other than 1 or 0 will be the output. This special scenario is discussed later.

Now, we define G_i's, the partial sums of the F_i values, as follows:

$$G_i = \sum_{j \leq i} F_j. \tag{5.2}$$

Using these partial sums, we can then calculate the sequence $(F'_{i,k})$ corresponding to each record as follows,

$$F'_{i,k} = (F_i) \left(\prod_{j \neq k} (G_i - \text{Enc}(j))\right) \left(\text{Enc}\left(\prod_{j \neq k} (k - j)\right)^{-1}\right), \tag{5.3}$$

where $1 \leq k \leq i$. It can be seen that $F'_{i,k} = \text{Enc}(1)$ if $F_i = \text{Enc}(1)$ and $G_i = \text{Enc}(k)$ are both satisfied. Hence, the sequences $(F'_{i,k})$ have the property that whenever $F_i = \text{Enc}(1)$ (i.e., the i^{th} record matches the query), we have an Enc(1) at the k^{th} position of the sequence where $G_i = \text{Enc}(k)$. All other entries of the sequence are encryptions of zero. Therefore,

$$(R') = \sum_k \text{Enc}(R_k)(F'_k), \tag{5.4}$$

where R_k is the k-th record in D will give us a sequence containing only the encrypted records that match our search query. Note that the definition of (R') relies on adding vectors of different lengths. This is done in the natural way, whereby all the vectors are made the same length by padding with zeros prior to addition.

To further illustrate our scheme, let us consider an example where the database contains five records, each with 4 bits of data. Also, let our encryption scheme encrypt 2 bits at a time. Then, if the search query is (Enc(2), Enc(3)), the corresponding F_i and G_i values are given in Table 2.

The resulting sequences (F'_i) would be similar as in Gahi's scheme,

$$(F'_1) = (\text{Enc}(0))$$

$$(F'_2) = (\text{Enc}(1), \text{Enc}(0))$$

$$(F'_3) = (\text{Enc}(0), \text{Enc}(0), \text{Enc}(0))$$

Table 5.1: Example database and corresponding F_i and G_i values

Database Records	F_i	G_i
$(0, 0, 1, 0)$	Enc(0)	Enc (0)
$(1, 0, 1, 1)$	Enc(1)	Enc (1)
$(1, 0, 0, 1)$	Enc(0)	Enc (1)
$(1, 0, 1, 1)$	Enc(1)	Enc (2)
$(1, 1, 0, 0)$	Enc(0)	Enc (2)

$$(F_4') = (\mathrm{Enc}(0), \mathrm{Enc}(1), \mathrm{Enc}(0), \mathrm{Enc}(0))$$

$$(F_5') = (\mathrm{Enc}(0), \mathrm{Enc}(0), \mathrm{Enc}(0), \mathrm{Enc}(0), \mathrm{Enc}(0))$$

Therefore, the sequence (R') would be,

$$(R') = (\mathrm{Enc}(R_2), \mathrm{Enc}(R_3), \mathrm{Enc}(0), \mathrm{Enc}(0), \mathrm{Enc}(0))$$

At this point, the sequence (R') will contain all the records that match our query but with trailing encryptions of zeros which we do not need. Hence, a second sum is calculated at the server side to determine the number of terms that are useful in the sequence:

$$n = \sum_r F_r.$$

Then n will be returned to the user and decrypted to obtain the number of records that match the search query. Hence, the sequence (R') can be truncated at the appropriate point and returned to the user for decryption.

It should be noted that the server will know the number of records that match the user's query. We believe that this information is not sufficient for the server to gain any additional information about the search query. Alternatively, we could return the whole sequence without truncation, keeping the number of matching records private from the server. However, the communication overhead will be increased significantly in this case, since the length of the sequence will be equal to the number of records in the database.

As promised previously, we now look at the special case where the record that is searched for is not contained anywhere in the database. In this case the value F_i will be something other than an encryption of 1 or 0. These garbage encrypted values will carry themselves into the rest of the protocol, resulting in equation 5.4 with a nonsensical sequence. Hence, if the user receives a nonsensical sequence as the final result, it implies that the record that was searched is not contained in the database. As an alternative approach, we can compute $\prod_i (\text{Enc}(m) - \text{Enc}(R_i))$ prior to computing F_i (equation 5.1) and send it to the user to decrypt. If the result is zero then m is contained in the database, and if it is non-zero, m is not contained in the database and therefore the user can send a message to the server to abort the search.

5.4 Comparison of Our Scheme vs. Gahi's Scheme

Our scheme has the main advantage of having the potential to be used with more recent fully homomorphic encryption schemes rather than being restricted to the DGHV scheme. This gives the flexibility to use our method with block based encryption schemes such as Braserski's [12], which reduces the number of encryption steps. For example, referring back to equation 4.1, we can see that the I_r values are calculated by comparing the query with each record bit-wise. If there are m records in the database and each of them are encrypted using n bits, the number of operations that are required to calculate all the I_r values will be $\mathcal{O}(nm)$. In our Homomorphic Query Processing method, equation 5.1 acts as the analog of equation 4.1. However, the encryptions are done block-wise in our scheme, and hence the number of operations it would take to calculate the F_i value in equation 5.1 will be $\mathcal{O}(m)$. For equation 4.2 in Gahi's method, the number of operations that should be performed to calculate all the partial sums will be $\mathcal{O}(nm^2)$, since there are $\mathcal{O}(m^2)$ multiplications and each multiplication should be done bit-wise; whereas the calculation of partial sums in our scheme (equation 5.2), the number of operations is reduced to $\mathcal{O}(m^2)$.

Similarly, equations 4.3 and 4.4 in Gahi's method use $\mathcal{O}(nm)$ and $\mathcal{O}(nm^2)$ number of operations, respectively, but their counterparts in our scheme, (equations 5.3 and 5.4) have $\mathcal{O}(m)$ and $\mathcal{O}(m^2)$ operations, respectively. Thus, it can be seen that in each step of our scheme, the number of operations performed is reduced by a factor of n compared to Gahi's method.

Chapter 6

Homomorphic Encryption and Smart Grid

In this chapter, we give an introduction to the smart electric grid as well as the cryptographic protocols that are being used in it. We look at several proposed cryptographic schemes that can be used to compute the total consumption function which is the commonly needed calculation on the part of the utility provider.

6.1 Smart Grid and Cybersecurity

Smart grids are a new class of communication grids that has recently gained widespread attention, mainly in the field of electricity distribution. Conventional power grids consist of two main components: the infrastructure that make up the electric network and the manpower needed to carry out the associated operations. The infrastructure consists of physical components that makes up the nuts and bolts of the grid such as power plants, cable lines, transformers, substations, etc. Along with these components workers are needed for metering and to collect data to adjust tariff rates accordingly. However, with the surge of complex consumption patterns, mainly due to the wide range of electric devices that are commonplace in modern households, appropriate billing according to the level of consumption of users is necessary. The fixed tariff scheme that is commonplace was no longer suitable to meet the demands of highly variable consumption patterns.

Due to these insufficiencies, a new innovative electric grid, emerged during the 21st century, commonly known as the "smart grid." It can be defined as an electric grid with embedded digital processing and communications. Rather than the traditional one-way electricity distribution from the power plant to consumers, the smart grid acts more like a communication network that automatically collects user data and sends it to the distributor, where the distributor adjusts the tariffs and loads according to the collected data [37]. In addition, the ultimate goal of implementing the smart grid is that it will possess the following qualities [13]:

- **Intelligent:** The capability to sense system failure and perform automated prevention measures. It is also believed that the smart grid will be able to respond to these situations faster than a human operator.
- **Efficient:** Due to optimization of load balancing, it will have the capacity to meet increased consumer demand without additional infrastructure.
- **Accommodating:** The ability to easily integrate and include different energy sources to the grid. This is very important since there is evidence that new energy sources, such as solar and wind energy, are rapidly coming online.
- **Motivating:** The ability to connect the consumer and utility in real time so that users can fine-tune their energy usage based on individual preferences. For example, having the ability to track daily energy usage informs the consumer about where most of the energy is spent, and the consumer may opt to change their lifestyle accordingly.
- **Opportunistic:** The ease of integration into the market with capabilities such as plug-and-play devices (e.g., smart meters).

- **Quality Focused:** The ability to provide power with the desired quality standards. For example, with the increasing trend in using electronic devices, consumers demand uninterrupted and very stable power signals without any spikes and disturbances.

- **Resilient:** The smart grid is more distributed and hence there is a need for increased resilience to attacks and disaster recovery measures.

- **Green:** More environmentally friendly and providing an optimized power management scheme which contributes to a greener economy.

Although the smart grid is an innovative technological breakthrough, there are many challenges associated with the collection and dissemination of user data. Most of the data sent are related to usage patterns of electricity. For example, a household can have a certain pattern of electricity usage, and this can be used to determine personal behavioral patterns [6]. This idea is explained in more detail in the following example.

Figure 6.1, taken from [6], shows how the power usage of a typical household fluctuates in accordance with personal activity. The spikes in the graph represent occasions where the greatest amount of power is used. Looking at the intensity and the distribution of energy fluctuations, the devices in operation at that specific time can be inferred. This is possible because each device that uses a significant amount

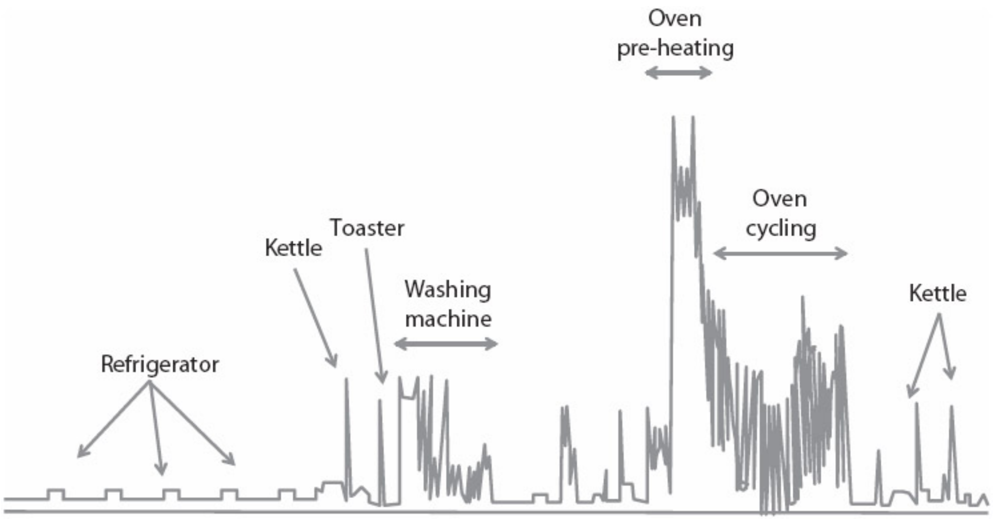

Figure 6.1: Residential Power Usage to Personal Activity Mapping

of power has its own signature power fluctuation. This type of load monitoring is known as non-intrusive load monitoring (NALM) and it can be used to infer the number of consumers living within a household and their behaviors [2]. The privacy risk associated with this kind of load monitoring process is immense, as it provides an attacker a chance to predict things such as the time at which people are present in the house and an estimate of the number of people present.

Hence, the need for smart grid cyber security arises. The smart grid consists of millions of electronic devices that have varying levels of hardware and software constraints. Thus blindly applying protocols of IT security may not prove to be useful. For example, a typical encryption scheme used in IT such as the Advanced Encryption Standard (AES) might be too cumbersome for smart grid devices with limited computational ability. The AES was originally designed for 8-32 bit processors, making it less effective when used with constrained devices [7]. Thus, it is necessary to have dedicated protocols that are tailored specifically for the smart grid. The security challenges of the smart grid are based on the following properties [6]:

- Scalability: The size of smart grid grows and involves cross linking of multiple diverse networks. Compatibility issues integrating such diverse protocols is an ongoing challenge.
- Mix of legacy and modern devices: The smart grid consist of many devices that are used not only in the generation and distribution of electricity but also in communication networks. Therefore, all devices used in the smart grid might not correspond to the same level of functionality as a modern device. Hence, it is important that protocols are designed in a way that unifies all the new and old devices.
- Hybrid model of management: The security of the grid should be distributed and centralized, that is, a device should have the ability to locally authenticate if the communication network fails for some reason. Hence, a local authentication system is necessary, along with a centralized database that synchronizes

credentials.

- Evolving standards and regulations: The standards and regulations that are currently in place for smart grid cybersecurity are rapidly changing in accordance with new innovations and technologies. Hence, building utilities with implementation standards that are valid for prolonged periods of time is a challenge.

In addition to these challenges, the current security goals for the smart grid have been outlined by the National Institute of Standards and Technology (NIST) smart grid interoperability panel [2]. Smart grid security goals can be condensed into three main categories: Confidentiality, Integrity, and Availability.

- **Confidentiality:** Confidentiality refers to using access controls to restrict data access by unauthorized entities. For example, smart meter data should only be accessed by the grid operators, not other consumers.

- **Integrity:** Integrity is the prevention of unauthorized modification of data. Loss of integrity can lead to incorrect decisions regarding power management.

- **Availability:** Availability is the reliable and timely access to information by authorized entities. Denial of Service (DOS) attacks attempt to block or corrupt communication in the smart grid. Loss of availability can lead to incorrect tariff calculations and power failures.

Thus, it is evident that there is a need for smart grid security protocols that allow secure transmission and storage of data. In the next section, we examine a few recently proposed encryption protocols that offer improvements to smart grid cybersecurity.

6.2 Cryptography-Based Data Exchange Schemes for Smart Grid

The transmission of information in the smart grid makes it susceptible to information leakage and to unauthorized access to sensitive information. To know more about where information leakage can occur, we consider the parties involved in a smart grid [16].

- **Consumers/Customers:** The consumers of a smart grid are its end users. Each customer has a *Smart Meter*, which is used to measure their energy usage and report it to the energy supplier at a specified frequency. For example, the smart meters implemented by Hydro One send measurements hourly [38]. Smart meters should be cost efficient and thus have limited computational power.

Consumers should have access to their energy usage patterns via smart meter readings.

- **Grid Operators:** Grid operators are the companies that manage the distribution of electricity. Operators can be part of an energy producer or separate entities. The distributor will use the smart meter data to efficiently distribute electricity to its consumers. Electricity distribution uses smart meter data for load balancing.

- **Communication Network:** This is the network in which all parties involved exchange data. The communication channel should be secured to maintain the privacy of data.

- **Electricity Producer:** This is the company that produces electricity and distributes it via the grid operator's infrastructure. The producer should manage power production according to the demand and also set tariffs according to customer usage patterns.

- **Aggregator:** The aggregator collects all the smart meter data from consumers and aggregates it to obtain values that the electricity producer can use. For example, the aggregator can compute the average power consumption of consumers in a certain area at a given time frame. This will help the producer and

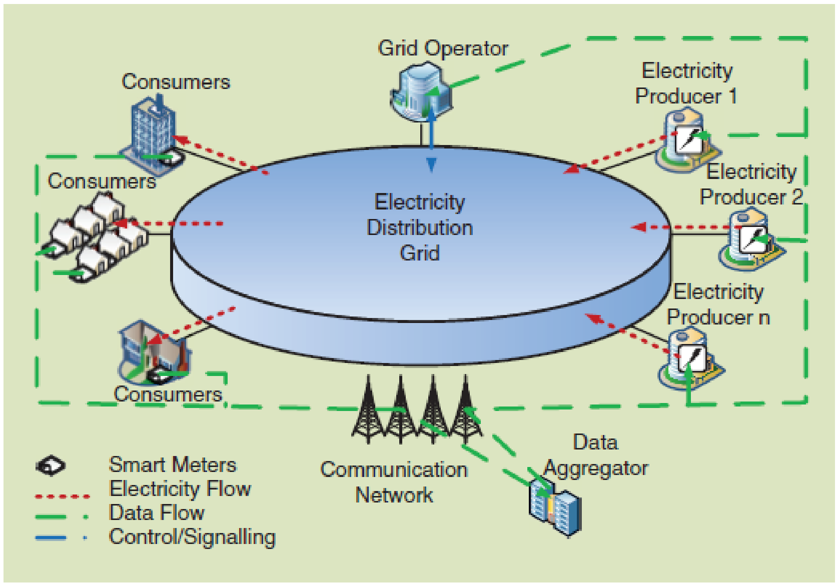

Figure 6.2: Smart Grid and its Stockholders

the distributor to balance the loads efficiently. A schematic representation of the contributing parties is shown in Figure 6.2 (taken from [16]).

One of the obvious ways to entrust security in the smart grid is to have a centralized grid management system where the smart meters only measure and send the data in an encrypted format to a centralized server. The centralized server acts as the aggregator and communicates with each smart meter to calculate total consumption, handle load balancing, etc. Since the grid operator controls the cryptographic keys for each smart meter, it is free to perform any function on the data. However, this places a universal trust on the grid operator/aggregator, which is undesirable [21].

If the grid operator is compromised, the whole smart grid will be compromised as well. Hence, recent proposals focus on grid management that is more widely distributed [14,40]. Rather than relying on a centralized grid operator for total security, the goal is to distribute trust to smart meters and substations and to design them to be independent of the grid operator.

One of the novel methods of preserving privacy in the smart grid is to use homomorphic encryption schemes. Since homomorphic encryption schemes allow computing on encrypted data, the grid operator does not require handling of smart meter data in plaintext format. Before looking into how homomorphic encryption schemes can be used in the smart grid context, we specify the types of statistical functions that the grid operator wants to compute. The most typical and important calculations that any grid operator wants to perform is the total consumption $C_T(t)$ and billing $B(t)$ at a given time t. Both values can be represented by the general summation of the smart meter readings $m_{i,t}$:

$$GS(t) = \sum_{M_s} f(m_{i,t}),$$

where i indicates the smart meter index and t indicates the time slot. Here, M_s represents index i or time t, depending on whether the total consumption or the billing function is being calculated. In the case of the total consumption, we are interested in calculating the summation over all smart meters and therefore the summation becomes $\sum_i f(m_{i,t})$. For billing of a single household over a given time t, the summation is $\sum_t f(m_{i,t})$. In the case of total consumption, f is the identity function (i.e., $f(m_{i,t}) = m_{i,t}$); for billing, f is an appropriate billing function [16]. Typically, the billing function is calculated by multiplying the energy rate designated by the power provider with the total consumption during the given period. We now look into privacy preserving schemes that aim to compute these aggregation functions.

6.2.1 Encryption Schemes for Total Consumption

In this section, we examine several encryption protocols that aim to protect privacy in calculating the total consumption, namely:

$$C_T(t) = \sum_{M_s} f(m_{i,t}) = \sum_i m_{i,t}$$

We assume that smart meters have the following capabilities.

- The ability to communicate with the utility provider as well as each other. The connections can be wired or wireless. For example, currently implemented smart meter protocols use Bluetooth and ZigBee [5, 47].

- Every smart meter on the network has the ability to provide a valid digital certificate for authentication. This automatically assumes that there is a certificate authority.

- Smart meters can perform cryptographic operations. Due to hardware constraints, it is generally agreed that they can only perform fairly simple cryptographic protocols. The exact type of cryptographic protocols that a smart meter is allowed to carry out varies according to different sources, but they can typically perform hash functions, pseudorandom number generators, symmetric (e.g., Advanced Encryption Standard), and asymmetric encryption (e.g., Rivest-Sharmir-Adleman (RSA), Paillier, and El Gamal) [16].

Secret Sharing, first proposed by Shamir [43], is one of the first methods for preserving privacy in smart meter data. Its basic operating principle is to divide a secret S into n pieces such that, with all n pieces, the secret can be reconstructed, but any number of pieces less than n is inadequate to find the secret. The method that we are going to look into was proposed by Garcia and Jacobs and it uses Shamir's Secret Sharing as well as the Paillier additive homomorphic encryption scheme [21].

First, we briefly describe the Paillier cryptosystem. The Paillier encryption scheme consist of the following algorithms:

- **KeyGen**(p, q, g): The key generation phase takes two large primes p and q as inputs and computes $n = pq$ and $\lambda(n) = \text{lcm}(p-1, q-1)$, where $\text{lcm}(p-1, q-1)$ denotes the least common multiple of $p-1$ and $q-1$. Then, select a random integer $g \in \mathbb{Z}^*_{n^2}$, where $\mathbb{Z}^*_{n^2}$ denotes the invertible elements of $\mathbb{Z}_{n^2}$ and the order of g is a multiple of n. Note that since $g \in \mathbb{Z}_{n^2}^*$, g and n^2 are relatively prime. Therefore g is relatively prime to n. Thus by Carmichael's Theorem, $g^{\lambda(n)} \equiv 1 \bmod n$. For calculating the decryption function the value $g^{\lambda(n)} \bmod n^2$ is necessary. Note that, $g^{\lambda(n)} \bmod n^2 \equiv g^{\lambda(n)} \bmod n$ and therefore by the previous result, $g^{\lambda(n)} \bmod n^2 \equiv 1 \bmod n$. Thus, subtracting one from $g^{\lambda(n)} \bmod n^2$ will give a number divisible by n. Define,

$$\mathcal{L}(u) = \frac{u-1}{n},$$

and compute $\mathcal{L}(g^{\lambda(n)} \bmod n^2)$. Notice that since $g^{\lambda(n)}$ is calculated in $\bmod n^2$, it is greater than zero and strictly less than n^2. Thus dividing $g^{\lambda(n)} \bmod n^2$ by n results in a value greater than or equal to zero, but strictly less than n. That is, $\mathcal{L}(g^{\lambda(n)} \bmod n^2) \in \mathbb{Z}_n$. Since $n = pq$, as long as $\mathcal{L}(g^{\lambda(n)} \bmod n^2)$ is not congruent to a multiple of p or q mod n, then $\mathcal{L}(g^{\lambda(n)} \bmod n^2)$ has an inverse. This existence of inverse is crucial for decryption, and therefore if the inverse does not exist, choose a new value for g before proceeding further. Finally calculate $\mu = (\mathcal{L}(g^{\lambda(n)} \bmod n^2))^{-1} \bmod n$. The public key of this scheme is (n, g) and the private key is (λ, μ).

- **Enc**(m, r): The encryption phase takes a message from the message space $\mathbb{Z}_n$

and a random number $r \in \mathbb{Z}_n^*$ and computes the ciphertext,

$$E(m, r) = g^m r^n \text{mod } n^2$$

- **Dec**(c, p, q): The decryption of the ciphertext c is given by,

$$m = \mathcal{L}(c^\lambda \text{ mod } n^2).\mu \text{ mod } n$$

The Paillier scheme is additively homomorphic since $E(m_1, r)E(m_2, r) = g^{m_1+m_2}r^{2n} = E(m_1 + m_2, r)$. Further details on this scheme can be found in [41].

The basic goal of Garcia's protocol is to divide a smart meter measurement into shares that are equal to the number of participants in the protocol. We will explain the protocol using a three party scenario. Suppose Alice, Bob, and Charles have smart meters and their smart meter readings are m_A, m_B, and m_C, respectively. Each party will acquire three shares from their measurements. For example, Alice will have the shares $m_A(1), m_A(2)$, and $m_A(3)$.

$$\text{Alice: } m_A = m_A(1) + m_A(2) + m_A(3)$$

$$\text{Bob: } m_B = m_B(1) + m_B(2) + m_B(3)$$

$$\text{Charles: } m_C = m_C(1) + m_C(2) + m_C(3)$$

Then, Alice will keep one share and encrypt the other two shares with the Paillier encryption scheme using the public keys of Bob and Charles, respectively (Figure

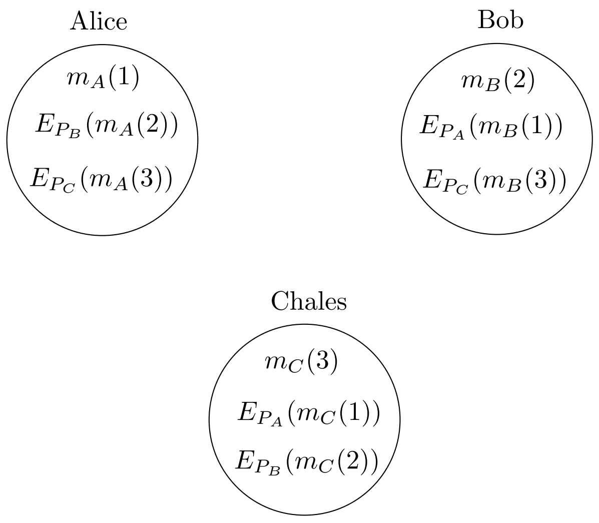

Figure 6.3: Partitioning the Smart Meter Readings

6.3). Each party then sends their encrypted shares to the utility provider/aggregator, keeping the unencrypted value to themselves. The aggregator will multiply the shares, which are encrypted using the same key and due to the homomorphic nature of the Paillier encryption scheme, we have,

$$\text{Alice: } E_{P_A}(m_B(1)).E_{P_A}(m_C(1)) = E_{P_A}(m_B(1) + m_C(1))$$

$$\text{Bob: } E_{P_B}(m_A(2)).E_{P_B}(m_C(2)) = E_{P_B}(m_A(2) + m_C(2))$$

$$\text{Charles: } E_{P_C}(m_A(3)).E_{P_C}(m_B(3)) = E_{P_C}(m_A(3) + m_B(3))$$

The aggregator sends these values to the respective party that owns the corresponding private key, who then decrypts the value and adds their share to it (Figure 6.4).

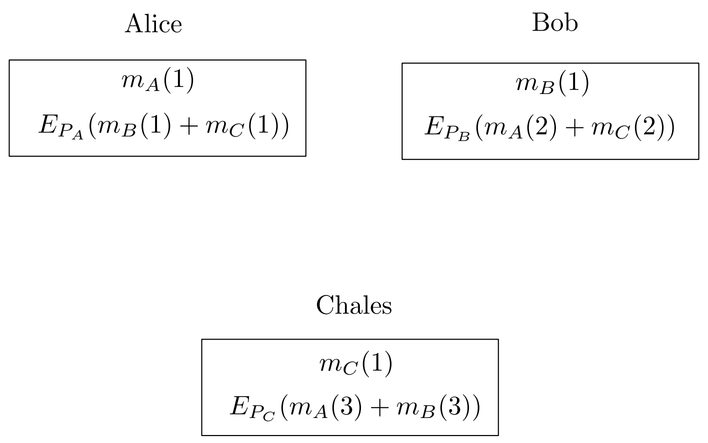
Alice
$m_A(1)$
$E_{P_A}(m_B(1) + m_C(1))$
Bob
$m_B(1)$
$E_{P_B}(m_A(2) + m_C(2))$
Chales
$m_C(1)$
$E_{P_C}(m_A(3) + m_B(3))$

Figure 6.4: Calculation of Partial Aggregations

Finally, each party sends their value to the aggregator to calculate the total consumption value by adding all the values it receives. This protocol can be extended to n parties easily. Let m_i denote the smart meter value of the i^{th} party and let p_i denote his public key. Let $m_{j,i}$ denote the j^{th} share of m_i. That is, $m_i = \sum_j m_{j,i}$. The aggregator will calculate,

$$\mathcal{E}_i = \prod_{j \neq i} E_{p_i}(m_j(i)) = E_{p_i}\left(\sum_{j \neq i} m_j(i)\right),$$

which he sends to the i^{th} party for decryption. Upon decryption he can add $m_i(i)$ as usual to obtain the sum of all the values encrypted under p_i.

This method uses a simple protocol to establish smart data privacy; however, in the case of n parties, each party will have to perform $n-1$ encryptions and therefore the total number of encryptions will be $n(n-1) = \mathcal{O}(n^2)$. Similarly, it can be shown that the number of multiplications for an n-party protocol is also $\mathcal{O}(n^2)$. Due to this quadratic running time, this protocol is not quite scalable.

Hence, a second approach is presented by Kursawe et al. [31]. Kursawe's approach is based on comparison protocols where the aggregator knows a rough estimate of the total consumption (e.g., from past measurements). Each of the parties computes $g_i^{m_1+r_1}, g_i^{m_2+r_2}, g_3^{m_3+r_3}$ and so forth, where g_i is a unique identifier based on a serial number or the date and time of the measurement. The values r_1, r_2, and r_3 are computed in random such that their sum is equal to zero. The aggregator can take the product of all the smart meter outputs to obtain,

$$\prod_j g_i^{m_j+r_j} = g_i^{\sum_j (m_j+r_j)} = g_i^{\sum_j m_j}.$$

Since the aggregator has an approximate value, C_{Tot} of the total consumption he can now compute $g_i^{C_{Tot}}, g_i^{C_{Tot}-1}, g_i^{C_{Tot}+1}, \ldots$ and compare for equality. The authors also propose several protocols that describe ways to generate r_j and g_i. We consider a protocol that is based on the Diffie-Hellman Key Exchange scheme.

In this scheme, we assume that each meter j has a secret key $X = R_j$ and a corresponding public key Pub_j. For each round i, a generator g_i of the Diffie-Hellman group is chosen.

- First, each smart meter j computes a round-specific public key, $\text{Pub}_{i,j} = g_i^{R_j}$

based on its secret key and the generator.

- The round keys are certified and distributed among all the smart meters included in the aggregation.

- Each smart meter computes the value,

$$g_i^{r_j} = \prod_{k \neq j} \mathrm{Pub}_{i,k}^{(-1)^{k<j} R_j},$$

where $k < j$ takes the value 1 if k is less than j, and is 0 otherwise. It can be seen that the summation of all r_j values is equal to zero:

$$\sum_j r_j = \sum_j \sum_{k \neq j} (-1)^{k<j} R_k R_j = 0$$

Thus, each smart meter can calculate $g_i^{m_j} g_i^{r_j}$ to obtain $g_i^{m_j + r_j}$, as required by the protocol. A summary of this protocol is depicted in Figure 6.5. As before, the number of messages exchanged between n parties is $\mathcal{O}(n^2)$ because each smart meter must distribute its Diffie Hellman key to every other smart meter. The number of multiplications is $\mathcal{O}(n)$; thus, this protocol has fewer multiplications than the previous scheme. Furthermore, in Kursawe's original paper [31], it is suggested that a key size of 256 bits should be used for the Diffie Hellman algorithm, which is significantly smaller compared to the key size of the Pallier encryption scheme proposed earlier by Garcia and Jacobs [21].

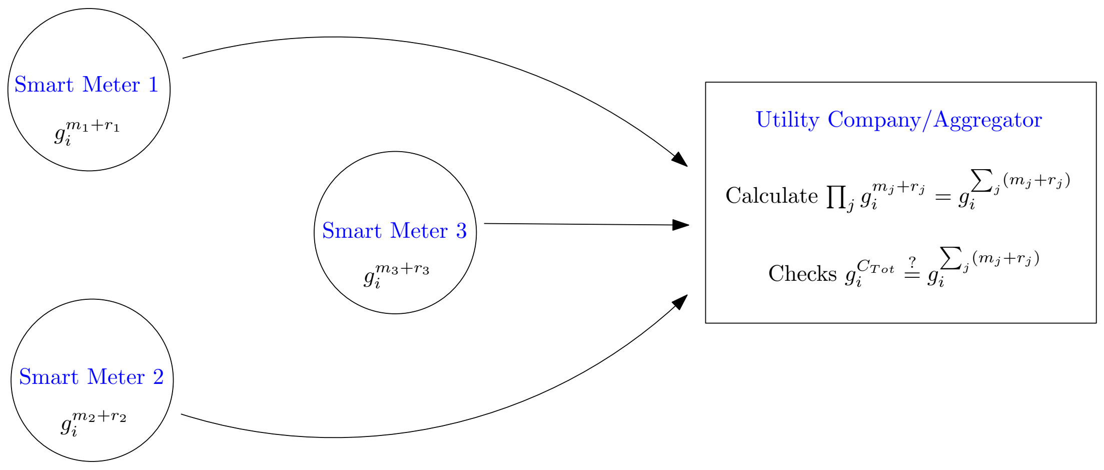
Smart Meter 1
$g_i^{m_1+r_1}$
Smart Meter 3
$g_i^{m_3+r_3}$
Smart Meter 2
$g_i^{m_2+r_2}$
Utility Company/Aggregator
Calculate $\prod_j g_i^{m_j+r_j} = g_i^{\sum_j (m_j+r_j)}$
Checks $g_i^{C_{Tot}} \overset{?}{=} g_i^{\sum_j (m_j+r_j)}$

Figure 6.5: Kursawe's Protocol for Data Aggregation

The third approach is by Erkin and Tsudik and uses a slightly altered version of the Paillier encryption scheme [17]. The idea behind Erkin and Tsudik's scheme is to break n into shares $n_1, n_2, n_3 \dots$ and distribute one share per smart meter. Each smart meter will then encrypt their consumption value as,

$$E(m_i, r) = g^{m_i} r^{n_i} \text{mod } n^2,$$

where i denotes the i^{th} smart meter. These values are then sent to the aggregator. Note that the aggregator cannot decrypt these values, even if he has the secret key, since in order for decryption to work, the random numbers r^{n_i} should be of the form r^n. However, after receiving all the shares, the aggregator can add them together to obtain the total consumption value, which he can now decrypt (see Figure 6.6).

$$\begin{aligned}\prod_i E(m_i, r) &= \prod_i g^{m_i} r^{n_i} \text{mod } n^2 \\ &= g^{\sum_i m_i} r^{\sum_i n_i} \text{mod } n^2 \\ &= g^{\sum_i m_i} r^{n} \text{mod } n^2 \\ &= E\left(\sum_i m_i, r\right)\end{aligned}$$

This method assumes that every party has the same random number, r. One way of achieving this is to take r as a hash of the time stamp. The main advantage of Erkin and Tsudik's protocol is that smart meters only need to perform one encryption. However, each smart meter must have the ability to perform Paillier encryption, a hash function, and random number generation (for r). Hence, the number of encryptions is reduced at the cost of introducing several different cryptographic algorithms. The major downside of this protocol is that the smart meters have to perform Paillier encryption as well as computing hashes and random numbers. To mitigate this issue, Ács and Castelluccia [3] proposed a very simple protocol that does not use computationally intensive schemes such as Paillier encryption. The encryption function of the cryptosystem that Ács and Castelluccia introduced is given by,

$$E(m, k, n) = m + k \text{ mod n}$$

where m is the message to be encrypted, k is the key and n is a large integer. Note

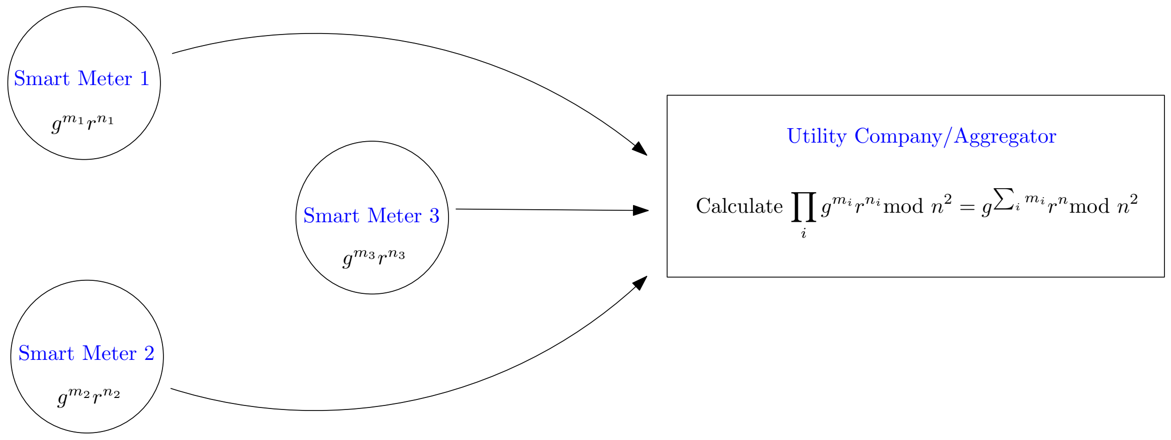
Smart Meter 1
$g^{m_1}r^{n_1}$
Smart Meter 3
$g^{m_3}r^{n_3}$
Smart Meter 2
$g^{m_2}r^{n_2}$
Utility Company/Aggregator
Calculate $\prod_i g^{m_i}r^{n_i} \bmod n^2 = g^{\sum_i m_i}r^n \bmod n^2$

Figure 6.6: Erkin and Tsudik's Protocol for Data Aggregation

that this scheme is homomorphic with respect to addition.

$$\begin{aligned} E(m_1,k_1,n)+E(m_2,k_2,n) &= (m_1+k_1) \text{ mod n} + (m_2+k_2) \text{ mod n} \\ &= m_1+m_2+k_1+k_2 \text{ mod n} \\ &= E(m_1+m_2,k_1+k_2,n) \end{aligned}$$

The protocol starts by choosing a subset of smart meters. Let us walk through an example with three consumers: Alice, Bob, and Charles. Each user shares a random number with every other user in a cyclic fashion, as shown in Figure 6.7.

In our example, Alice sends r_1 to Bob, Bob sends r_2 to Charles, and Charles sends r_3 to Alice. Let us also assume that each participant shares a secret key with the aggregator: K_A, K_B and K_C respectively. Alice adds to her measurement what she

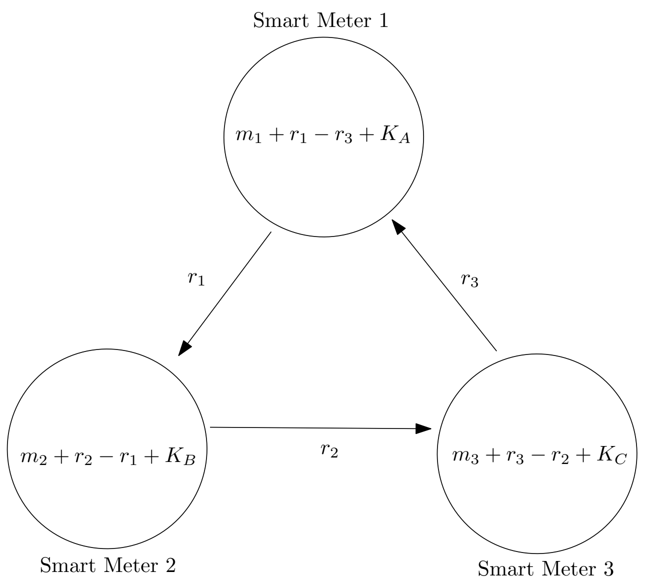

Figure 6.7: Ács and Castelluccia's Protocol for Data Aggregation

sends (in this case r_1), while subtracting what she receives (in this case r_3). Bob and Charles do the same. Finally, all participants encrypt their values using the aforementioned encryption scheme. The results are given below.

$$\text{Alice: } E(m_1, K_A, n) = m_1 + r_1 - r_3 + K_A \text{ mod n}$$

$$\text{Bob: } E(m_2, K_B, n) = m_2 - r_1 + r_2 + K_B \text{ mod n}$$

$$\text{Charles: } E(m_1, K_A, n) = m_1 + r_3 - r_2 + K_C \text{ mod n}$$

Each smart meter sends these encrypted values to the aggregator, which adds everything together so that the random numbers are canceled out. It can then decrypt the result using the sum of the shared secret keys. This scheme can be extended to N parties easily, in which case the sharing of random numbers will be given by a directed cycle graph of length N. Thus, for a scheme involving N parties, the number of communications will be $\mathcal{O}(N)$.

In the next chapter, we discuss data querying techniques in the smart grid. We also construct a keyword data querying technique based on the Homomorphic Query Processing Scheme that was introduced in Chapter 5. We also use the principles of Public Key Encryption with Keyword Search (PEKS) and Multi-Key Homomorphic Encryption to construct this method. This scheme can be used in combination with any of the data encryption techniques discussed in this chapter making smart meter data encrypted as well as queryable.

Chapter 7

Query Processing in the Smart Grid

All of the aforementioned techniques that use homomorphic encryption in the smart grid focus on securing data aggregation. Although there is a substantial amount of literature on data protection and encryption in the smart grid, there are very few studies on querying encrypted data. The need for more research in this area has been emphasized by the cybersecurity working group in the NIST Smart Grid Interoperability Panel in their *Guidelines for Smart Grid Cybersecurity* [2]. Alongside protecting data privacy through cryptographic countermeasures, when using smart meter data for decision making and price predicting, it is critical that the encrypted data is queried in encrypted form. Therefore, designing encrypted data querying techniques for the smart grid is essential while preserving the formal security goals (i.e., confidentiality, integrity, and availability). In this manner, utility companies can extract data based on their individual needs.

In this chapter, we look at existing techniques for querying encrypted data and use our Homomorphic Query Processing method in conjunction with a keyword search technique proposed by Boneh [10] to propose a new method for keyword searching in the smart grid. Smart meter data is usually collected by service providers for statistical purposes such as computing the total consumption function, as explained in Chapter 6. Our keyword search technique will help categorize smart meter data prior to their use.

7.1 Encrypted Data Querying Techniques

The problem of querying over encrypted data has been extensively studied by cryptographic and database communities [28]. There are several techniques available, depending on the type of query and the type of data that need to be supported. We give a summary of some of these proposed techniques.

7.2 Order-Preserving Encryption

Ideally, we want a cryptographic scheme that allows comparison-predicate evaluation over encrypted data. That is, an efficient scheme that can evaluate comparison operators such as “less than” ($<$), “greater than” ($>$), and “equality ” ($=$) over encrypted data. The first order-preserving encryption scheme was introduced by Agrawal et al [4]. This scheme allows the execution of range queries over encrypted data as well as maintain indexes for efficient access. However, Agrawal’s scheme only works for numeric data. A more efficient, order preserving encryption scheme was found by Boldyreva et al. [8]; they also provided a security analysis of Agrawal’s scheme. In addition, Boldyreva’s analysis suggests that the disadvantage of using order-preserving schemes is that they need to be deterministic. That is, all encryptions of a given plaintext are identical. This makes order-preserving schemes susceptible to statistical attacks.

7.3 Searchable Encryption Techniques

As its name suggests, “searchable encryption” techniques have the ability to search or categorize encrypted data. The first searchable encryption scheme was introduced by Boneh et al [9]. This scheme is commonly known as “Public Key Encryption with

Keyword Search" (PEKS), and it can support the equality query over encrypted data restricted to a particular collection of keywords. Then Golle et al. [24] proposed a scheme that supports conjunctive keyword search over encrypted data. Furthermore, Shi et al. [44] proposed an encryption scheme that supports multidimensional range queries over encrypted data (MRQED). This technique uses an interval tree data structure to construct a representation of intervals along each dimension. The authors show that the time complexity of the MRQED scheme is $\mathcal{O}(D \log T)$, where D is the number of dimensions, and T is the number of discrete values for each dimension.

7.4 Keyword Search in the Smart Grid

In Chapter 6, we discussed several proposed approaches to aggregate data in the smart grid. Although these methods provide novel techniques to calculate the total consumption, utility providers oftentimes require the ability to categorize data before doing any calculation. Most energy providers have customer categories, and each category has different tariff rates. For example, BC Hydro, the main energy provider in British Columbia, has two customer categories, Residential customers and Business customers, where business customers are sub-categorized into Small General Service customers, Medium General Service customers, and Large General Service customers.

In addition, both residential and business customers are categorized into Irrigation Rate and Transmission Rate customers depending on their needs and consumption habits. Each of these categories have different tariff rates; thus, as a utility provider, it is essential that, for example, the smart meter data sent from a residential customer is handled differently than the smart meter data sent from a business customer.

We propose a new method to categorize smart meter data based on a collection of keywords. This method works alongside the aggregation protocols discussed in Chapter 6, allowing the utility provider the ability to categorize smart meter data before computing total consumption for each category. We demonstrate this concept in the following example. Considering the customer categories of BC Hydro, let us assume the set of keywords is {residence, smallbusiness, mediumbusiness, largebusiness, irrigation, transmission}. We encrypt each of these keywords with a multi-key homomorphic encryption scheme and send them to the utility provider, along with the smart meter data encrypted by a suitable scheme. The utility provider will be able to search the encrypted keywords, and would be able to know whether a particular keyword of his choice is included in the keyword set or not. We employ several ideas in constructing this method: our Homomorphic Query Protocol proposed in Chapter 5, Boneh's Public Key Encryption with Keyword Search (PEKS) scheme [9], and the multi-key fully homomorphic encryption scheme proposed by López-Alt et al. [33].

First, we take a closer look at Boneh's PEKS scheme [9].

7.4.1 Public Key Encryption with Keyword Search (PEKS)

The PEKS scheme was initially introduced as a mechanism to search encrypted data for specific keywords. The following example, as well as the definitions accompany it, are taken from [9].

Suppose a user (Alice) wants to read her email on different devices: her laptop, desktop, pager, etc. Alice's email server will route the email to the appropriate device based on certain keywords that the email contain. For example, if Bob sends an email with the keyword "urgent," the email is routed to Alice's pager, and when Bob sends an email with the keyword "lunch," it will be routed to Alice's desktop. Each email is expected to contain a small number of keywords in the header or subject line. The Mobile People Architecture [34] provides a prototype of this type of email routing.

However, the above scheme is only possible if the emails are not encrypted. The mail server will have to make decisions according to the keywords that it sees, and if the emails are encrypted, this is not possible. In email as well as other communication networks (such as the smart grid), this ability to process unencrypted data is a violation of privacy. The goal of PEKS is to solve this issue by enabling the server to check whether certain keyword(s) are contained in the data without learning anything about the data itself. In the above example, Alice should be able to specify some keywords that the mail server will check in the mail it receives; however, the server should not learn anything about the content of the messages nor the keywords they contain.

We begin by defining the Public Key Encryption with Keyword Search (PEKS) scheme, similar to [9]. For the purpose of explanation, suppose Bob wants to send an email to Alice with the keywords $W_1, W_2, \ldots, W_k$. Bob encrypts the email using Alice's public key. Bob sends the following message:

$$\left[E_{pk}(M), \mathrm{PEKS}(pk, W_1), \ldots, \mathrm{PEKS}(pk, W_k)\right],$$

where pk is Alice's public key, M is the email body, and PEKS is the encryption scheme that allows searching of specific keywords. More details of the PEKS algorithm is given in Definition 7.1-7.2. The goal of this scheme is to allow Alice to send a secret trapdoor, T_W, to the server, which enables the server to check whether Bob's message contains the keyword W or not. The server should only learn whether W is contained in Bob's message and nothing else. The server can check for all such emails that contain the keyword W and send them back to Alice. This kind of scheme is called a "non-interactive public key encryption with keyword search" or in shorthand, a "searchable public key encryption" scheme. Note that in this protocol, Alice and Bob never interact with each other, hence the term "non-interactive" is used.

Definition 7.1 A non-interactive public key encryption with keyword search (sometimes abbreviated it as “searchable encryption”) scheme consists of the following polynomial time randomized algorithms:

1. KeyGen(s): Takes a security parameter, s, and generates a public/private key pair pk, sk.

2. PEKS(pk, W): For a public key pk and a word W, a searchable encryption of W is produced.

3. Trapdoor(sk, W): Given Alice’s private key and a word W a trapdoor T_W is produced.

4. Test(pk, S, T_W): Given Alice’s public key, a searchable encryption $S = \text{PEKS}(pk, W_0)$, and a trapdoor $T_W = \text{Trapdoor}(sk, W)$, the output is “yes” if $W = W_0$ and “no” otherwise.

First, Alice runs the KeyGen algorithm and generates her public and private keys, pk and sk, respectively. Then, she creates the trapdoor T_W for each keyword W that she wants the server to search for. The server uses the trapdoor T_W as input to the test algorithm to determine if the message contains W as a keyword.

The security of the PEKS scheme is defined as follows. All the PEKS(pk, W) values should not reveal any information about W unless T_W is available. Also, it is assumed that the attacker is able to obtain T_W for any W of his choice. Furthermore, the attacker should not be able to distinguish an encryption of W_0 from W_1 for which he did not obtain the trapdoor. Let A be an active attacker. Then the security of the PEKS scheme is defined by the following game between A and a challenger. The security parameter s is provided to both the attacker and the challenger.

Definition 7.2 PEKS Security Game:

1. The challenger runs the KeyGen(s) algorithm to generate pk and sk. It assigns pk to the attacker.

2. The attacker can adaptively ask the challenger for the trapdoor T_W for any keyword $W \in \{0,1\}^*$ of his choice.

3. At some point, the attacker A sends the challenger two words W_0, W_1 on which he wishes to be challenged. The only restriction is that the attacker did not previously ask for the trapdoors T_{W_0} or T_{W_1}. The challenger picks a random $b \in \{0,1\}$ and gives the attacker $C = \text{PEKS}(pk, W_b)$. We refer to C as the challenge PEKS.

4. The attacker can continue to ask for trapdoors T_W for any keyword W of his choice as long as $W \neq W_0, W_1$.

5. Eventually, the attacker A outputs $b' \in \{0, 1\}$ and wins the game if $b = b'$.

In summary, the attacker wins the security challenge if he correctly guesses whether he was given the PEKS for W_0 or W_1. The attacker's advantage in breaking the PEKS is given by,

$$\text{Adv}_A(s) = \left|\Pr[b = b'] - \frac{1}{2}\right|.$$

We propose a new method of processing queries with respect to keywords in the smart grid using this PEKS scheme along with the notion of multi-key fully homomorphic encryption proposed by López-Alt et al [33]. In the next section, we give a brief introduction to multi-key fully homomorphic encryption.

7.4.2 Multi-Key Fully Homomorphic Encryption

Fully homomorphic encryption schemes such as Gentry's original lattice based construction [23] and Braserski's ring-based construction [12] successfully outsource computations while keeping the data encrypted. However, these fully homomorphic encryption schemes focus on a single user. That is, if Alice wants to upload her data to a remote server, she can encrypt her data using a fully homomorphic encryption scheme before uploading so that the server can perform computations. However, the server can compute on these encrypted data only if all the data uploaded by Alice is encrypted via a single key. This is exactly what we want in a scenario where computations involve the data of a single user, but there are scenarios where computations should be carried out on data belonging to multiple users. A perfect example of this is the smart grid. Each user will upload data encrypted by their private key, and ideally we want the server to compute functions on these encrypted data without decrypting them. Furthermore, the server should be able to independently carry out the function of its choosing without consulting the users (non-interactively), and the result should be encrypted. Finally, in the decryption phase, the users who collaborated in the encryption will interact with the server to perform the decryption. This is what we hope to achieve by a multi-key fully homomorphic encryption scheme.

In the next section, we outline the multi-key fully homomorphic encryption scheme proposed by López-Alt et al. [33], which is derived from the modified NTRU encryption scheme [46].

Definition 7.3 Modified NTRU Encryption Scheme: This scheme was derived from the original NTRU encryption scheme by Hoffstein et al [27]. The scheme is parametrized by the ring $R = \mathbb{Z}/\langle x^n + 1\rangle$, where n is a power of two, q is an odd prime number, and $\mathcal{X}$ is a B-bounded distribution over R for $B \leq q$. Here "B-bounded" means that the magnitude of the coefficients of a polynomial sampled from $\mathcal{X}$ is less than B. Also, we define $R_q = R/qR$ and use $[\,.\,]_q$ to denote coefficient-wise reduction modulo

q into the set $\{-\lfloor \frac{q}{2} \rfloor, \ldots, \lfloor \frac{q}{2} \rfloor\}$. The scheme consists of the following algorithms.

- **Keygen**: Key generation samples "small" polynomials $f', g \leftarrow \mathcal{X}$ and sets $f = 2f' + 1$ such that $f(\text{mod } 2) = 1$. If f is not invertible in R_q, it resamples f'. Otherwise, it computes the inverse f^{-1} of f in R_q and sets,

  $$sk = f \text{ and } pk = \left[2gf^{-1}\right]_q,$$

  where sk and pk stand for secret key and public key, respectively.

- **Enc**(m, pk): To encrypt a bit $m \in \{0, 1\}$, the encryption algorithm samples "small" polynomials $s, e \leftarrow \mathcal{X}$, and outputs the ciphertext,

  $$c = [hs + 2e + m]_q,$$

  where $h = pk$.

- **Dec**(c, sk): To decrypt a ciphertext c, the decryption algorithm computes $\mu = [fc]_q$ and returns $\mu(\text{mod } 2)$.

The correctness of this scheme is verified by observing that,

$$\begin{aligned} [fc]_q &= [fhs + 2fe + fm]_q \\ &= [2gs + 2fe + fm]_q. \end{aligned}$$

Note that since the elements $g, s, f,$ and e are sampled from the B-bounded distribution, and $B \leq q$, the magnitude of the coefficients in $2gs + 2fe + fm$ are less then $q/2$ and hence there is no reduction modulo q. That is, $[2gs + 2fe + fm]_q = 2gs + 2fe + fm$. Therefore, $\mu = 2gs + 2fe + fm$, which implies that $\mu(\text{mod } 2) = m$ since $f(\text{mod } 2) = 1$.

In the following section, we give a brief summary of the multi-key homomorphic properties of this scheme.

Let $c_1 = [h_1 s_1 + 2e_1 + m_1]_q$ and $c_2 = [h_2 s_2 + 2e_2 + m_2]_q$ be ciphertexts under the keys $h_1 = \left[2g_1 f_1^{-1}\right]_q$ and $h_2 = \left[2g_2 f_2^{-1}\right]_q$, respectively. Then $[c_1 + c_2]_q$ and $[c_1 c_2]_q$ decrypts to $m_1 + m_2$ and $m_1 m_2$, respectively, under the joint key $f_1 f_2$, as follows:

$$\begin{aligned} f_1 f_2 \left(c_1 + c_2\right) &= 2(f_1 f_2 e_1 + f_1 f_2 e_2 + f_2 g_1 s_1 + f_1 g_2 s_2) + f_1 f_2 (m_1 + m_2) \\ &= 2e_{add} + f_1 f_2 (m_1 + m_2), \end{aligned}$$

where $e_{add} = f_1 f_2 e_1 + f_1 f_2 e_2 + f_2 g_1 s_1 + f_1 g_2 s_2$ is the noise of the resulting ciphertext. Similarly,

$$\begin{aligned} f_1 f_2 (c_1 c_2) &= 2(2g_1 g_2 s_1 s_2 + g_1 s_1 f_2 (2e_2 + m_2) + g_2 s_2 f_1 (2e_1 + m_1) + \\ &\quad f_1 f_2 (e_1 m_2 + e_2 m_1 + 2e_1 e_2)) + f_1 f_2 (m_1 m_2) \\ &= 2e_{mult} + f_1 f_2 (m_1 m_2), \end{aligned}$$

where $e_{mult} = 2g_1g_2s_1s_2+g_1s_1f_2(2e_2+m_2)+g_2s_2f_1(2e_1+m_1)+f_1f_2(e_1m_2+e_2m_1+2e_1e_2)$ is the noise of the resulting ciphertext. Therefore, the ciphertexts $[c_1 + c_2]_q$ and $[c_1c_2]_q$ decrypt to $m_1 + m_2$ and m_1m_2 as long as the noise elements e_{add} and e_{mult} do not become too large. This scheme can be made fully homomorphic using Gentry's bootstrapping technique and is thus able to evaluate any Boolean circuit. After evaluating the required circuit at the server, the parties involved in the encryption collaborate to run a secure multi-party computation protocol to evaluate the decryption circuit. For further details about this scheme we refer to [33].

Now we will see how the PEKS scheme and the multi-key homomorphic encryption scheme can be used with the Homomorphic Query Processing method to evaluate database queries.

7.5 Query Processing Scheme for Smart Grid

In this section, we describe how the PEKS scheme (Section 7.4.1) can be used to construct a model that allows query processing of keywords in the smart grid. For this task, we use the Homomorphic Query Processing method that we introduced in section 5.3, along with the multi-key homomorphic encryption scheme mentioned in section 7.4.2.

Ideally, the server/utility company will want to categorize smart meter data into sections according to various characteristics. For example, querying and separating data by user type (i.e., household, business etc.), or by users that belong to different payment plans, might be some examples where this might be useful. In this scheme, the smart meters send their meter readings appended by a collection of keywords. Suppose the number of smart meters communicating with the utility provider is n and denote these smart meters by $SM_1, \ldots, SM_n$. Furthermore, the readings of the smart meters are $m_1, m_2, \ldots, m_n$, respectively.

- The smart meter readings $m_1, m_2, \ldots, m_n$ will be encrypted by a suitable encryption scheme, as discussed in section 6.2.1. The choice of this encryption depends on the relative strengths and weaknesses of the methods discussed and the given smart grid requirements. We denote these encrypted values by $E(m_1), \ldots, E(m_n)$, where the encryption keys are the relevant public keys of the smart meters.

- The keywords are encrypted by a multi-key homomorphic encryption scheme (such as 7.3) and appended to $E(m_1), \ldots, E(m_n)$, respectively. Let us denote the keywords belonging to smart meter i by $K_1^i, \ldots, K_m^i$ and the encryptions of these values by $\mathrm{MKH}_{h_i}(K_1^i), \ldots, \mathrm{MKH}_{h_i}(K_m^i)$, where h_i denotes the public key of SM_i. Then, each smart meter sends the following data blocks to the server:

$$\begin{aligned}
SM_1 &: [E(m_1), \text{MKH}_{h_1}(K_1^1), \ldots, \text{MKH}_{h_1}(K_m^1)] \\
SM_2 &: [E(m_2), \text{MKH}_{h_2}(K_1^2), \ldots, \text{MKH}_{h_2}(K_m^2)] \\
&. \quad . \quad . \quad . \quad . \quad . \quad . \quad . \\
&. \quad . \quad . \quad . \quad . \quad . \quad . \quad . \\
&. \quad . \quad . \quad . \quad . \quad . \quad . \quad . \\
SM_n &: [E(m_n), \text{MKH}_{h_n}(K_1^n), \ldots, \text{MKH}_{h_n}(K_m^n)]
\end{aligned}$$

- The server will query the data sent by the smart meters and see whether they contain certain keywords. Suppose the server wants to search the data sent by SM_i for the keyword K. The server can use the same general principle of our Homomorphic Query Processing method (equation 5.1) to check which keyword matches K.

  Let g and h be the secret key and the public key of the server, respectively, and g_i and let h_i be the secret key and public key of SM_i, respectively. Define,

$$P_j = \prod_{K_l \neq K_j} \text{MKH}_h(K) - \text{MKH}_{h_i}(K_l)$$

  and

$$Q_j = \prod_{K_l \neq K_j} \text{MKH}_{h_i}(K_i) - \text{MKH}_{h_i}(K_l),$$

where $\mathrm{MKH}_x(m)$ denotes that m is encrypted under the multi-key homomorphic encryption scheme with public key x. Also, let m be the number of keywords sent by SM_i. Notice that the combined decryption key of the P_j is $(gg_i)^{m-1}$ since P involves a product of $m-1$ ciphertexts, each with decryption key gg_i, whereas the combined decryption key of the Q_j is $g_i^{2(m-1)}$ since Q involves a product of $m-1$ ciphertexts, each with decryption key g_i^2. The server can calculate P_j and Q_j given all the encrypted keywords sent by the user. However, the server is unable to decrypt the results because it does not have the combined decryption keys.

- Finally the smart meter (SM_i) and the server collaborate to compute F_i, defined as follows:

$$F_j = \frac{P_j}{Q_j} = \frac{\prod_{K_l \neq K_j} \mathrm{MKH}_h(K) - \mathrm{MKH}_{h_i}(K_l)}{\prod_{K_l \neq K_j} \mathrm{MKH}_{h_i}(K_i) - \mathrm{MKH}_{h_i}(K_l)}$$

  The numerator (P_j) and denominator (Q_j) calculated in the preceding step by the server are sent to SM_i, where they are multiplied by g_i^{m-1} and $g_i^{2(m-1)}$, respectively. Then, these values are sent back to the server securely, where the server can multiply P_j with its part of the secret key, g^{m-1}. Hence the server will have both P_j and Q_j in plaintext and can calculate F_j by taking the division of the two results. This would result in a table of F_j values similar to Table 5.1, except the fact that each value is in plaintext. For example, if K matches the 5th, keyword F_5 would be equal to 1, whereas all the other F_j values would be zero. Hence, the server would know whether the keyword is contained in the data sent by SM_i. This process is repeated for each smart meter.

This scheme has the advantage that homomorphic encryption is only used in the querying and keyword encrypting processes, thereby reducing the computational overhead. In the next chapter, we assess the relative strengths and weaknesses of these protocols, the challenges in implementing them, and future research directions.

Remark:

It should be noted that we can use our Homomorphic Query Processing method (section 5.3) with Boneh's PEKS scheme (section 7.4.1) to propose an alternative approach to keyword search in emails. Recall that Boneh's scheme applies to the case where two users, Alice and Bob, wish to communicate with each other through email. Bob sends an email to Alice, who would like to route the email to the appropriate device depending on the keywords it contain. Alice achieves this by sending a trapdoor function to the server, enabling the server to search for a particular keyword. In place of the trapdoor function, Alice can send the keyword (say K) that she wishes to search, encrypted by Bob's public key. If we use Braserski's encryption scheme in encrypting all the keywords, the server can use the keywords sent by Bob, along with the encrypted keyword sent by Alice, to compute F_i values for each keyword (in other words, compare K with each keyword) using equation 5.1. The server can then collaborate with Alice to decrypt the result, which will indicate whether K is contained in the set of keywords.

Chapter 8

Challenges and Future Research Directions of the Smart Grid

Most of the implementation challenges faced by the smart grid today boils down to three main categories: hardware limitations, cryptographic protocols, and signal processing. In this section, we examine each of these challenges and show how the previously discussed protocols fall into these categories.

8.1 Hardware Limitation Challenges

One of the most essential elements of a smart grid is its scalability. Most smart grids consist of tens of thousands of consumers and the cryptographic protocols that are used should be scalable to this size. For example, Garcia/Jacob's protocol [21] and Kursawe's [31] approach employ a quadratic complexity of communication for the aggregator (the number of messages that should be exchanged for N smart meters is $\mathcal{O}(N^2)$) and a linear complexity for each smart meter (since each smart meter should distribute $\mathcal{O}(N)$ number of messages). In comparison, Erkin/Tsudik's protocol [16] and Ács/Castellucia's protocol [3] have a linear communication complexity for the aggregator and constant complexity for the smart meters. Hence, it is more desirable to use the latter two protocols in terms of their scalability.

From the point of view of the grid operators, smart meters should be cheap and easily replaceable. Therefore, most smart meters used today have limited computational power that is insufficient to carry out complex homomorphic encryption schemes. For example, the Open Smart Grid Protocol and the smart metering protocols used in the Netherlands employ symmetric encryption schemes such as RC4 and DES [18,30]. Although weaknesses in these protocols have been reported (see [32]), they remain in use due to their high usability.

8.2 Cryptographic Protocol Challenges

Almost all the protocols presented here are designed with the assumption in mind that consumers and smart meters are not malicious. However, the real-world situation is quite different. Apart from the privacy concerns associated with consumer data, there is also a possibility that users might act maliciously to alter or sabotage the correct behavior of smart meters. Although challenges that are faced by tampering smart meters have been discussed in [1,35], there appears to be a lack of research on protocols that achieve both the privacy goals related to aggregation of measurements as well as smart meter tamper proofness.

Another challenge faced in aggregation protocols is that every participant should use the same key in order for the measurements to be homomorphically combinable. Most well known homomorphic encryption schemes work in this manner. For example, the same public and private keys should be used in Erkin and Tsudik's protocol [16] for Pallier encryption scheme to work in the desired manner. This creates a problem in terms of key distribution: If one party is malicious or compromised, there is a risk of exposing the private keys used by all the other parties and the aggregator. One measure towards resolving this problem is to use encryption schemes such as the one used by Ács and Castelluccia [3], where each pair of parties uses their own secret key. However, these schemes have a high communication overhead due to the huge amount of key distribution.

Most cryptographic protocols depreciate due to increasing processing power and newer attacking algorithms. Therefore, it is necessary that the devices that are connected to the smart grid should have some secure means of upgrading their crypto algorithms. In the case of fully fledged computers, this is quite easy to implement, for example, through a software or driver update. Similarly, smart grid devices should be designed with future upgrades to crypto algorithms in mind.

8.3 Future Research Directions

It is evident that the consequences of smart grid cybersecurity can be quite significant. Deploying a smart grid without proper security mechanisms in place would result in utility fraud, loss of user consumption data etc. One of the most critical issues of smart grid cybersecurity is the problem of key distribution. The simplest method is to use a single key that is shared by all the users of the smart grid. However, this approach has the potential to fail catastrophically because a breach at a single node will compromise the security of the entire grid. The main challenges associated with the smart grid are due to its stringent security requirements and limited computational resources. Although many Cryptographic solutions have been proposed both by academia and industry, most of these involve an unrealistic communication overhead for smart devices with limited computational power.

In this work, we have discussed several proposed methods to aggregate smart meter measurements using homomorphic encryption. The first approach by Garcia and Jacobs [21] uses the concept of secret sharing to break each smart meter reading into several parts and to distribute these parts among other participants. However, the number of homomorphic encryptions of this protocol grows quadratically with the number of smart meters, making it unsuitable for large-scale smart grids. On the other hand, Kursawe's [31] approach using the Diffie Hellman key exchange protocol reduces the number of homomorphic encryptions at the cost of introducing a large communication overhead. The third approach by Erkin and Tsudik [16] solves both of these problems, but it assumes the smart meters are capable of performing several computationally expensive cryptographic schemes (i.e., Paillier encryption, a hash function, and random number generation). The final protocol proposed by Ács and Castelluccia [3] introduces a lightweight additive homomorphic cryptosystem that mitigates the problems associated with secret sharing and computationally intensive cryptosystems like Paillier. Therefore, from a practical standpoint, the best scheme currently available for secret aggregation is Ács and Castellucia's scheme. However, further research has to be carried out especially in order to determine how the random values between the smart meters (r_1, r_2 and r_3 in Figure 6.7) can be securely distributed.

8.3.1 Improvements to Homomorphic Query Processing

In addition to these challenges that are inherently present in the smart grid, there is also room for improvement in our proposed methods: the Homomorphic Query Processing (section 5.3) and the encrypted keyword technique in the smart grid (section 7.5). The Homomorphic Query Processing method uses plaintext inverses to calculate the value of F_i in equation 5.1, essentially requiring the plaintext space in the fully homomorphic encryption scheme to be a field. It would be a major improvement if our scheme could be made independent of plaintext inverses, thus providing more flexibility to the user in choosing the plaintext space. One of the approaches that can be used to construct an alternative method to calculate F_i is to use homomorphic subtraction of the two multiplicands of equation 5.1. For example, $\text{Enc}(a) - \text{Enc}(b) = \text{Enc}(a - b)$ will be equal to an encryption of zero if a and b are equal and an encryption of some other value if not. We believe this idea can be developed further to create an alternative to our proposed method.

In addition, a next logical step is to improve this method to process conjunctive queries, which involve comparison of two or more values of the table simultaneously. A typical solution to this problem is to compare each individual predicate of the conjunct separately, but this has efficiency drawbacks.

Finally it would be interesting to look at avenues in which our homomorphic query processing scheme can be implemented, so as to obtain experimental results.

8.3.2 Query Processing in the Smart Grid

In this thesis, we have investigated the problem of querying encrypted data in the smart grid. We have proposed a method using PEKS [9] and multi-key homomorphic encryption [33] to tackle this problem. The amount of research done in the area of multi-key homomorphic encryption schemes seems to be quite limited, and to our knowledge, the only multi-key encryption scheme currently available is the one introduced by López-Alt et al. [33] (section 7.4.2). Thus, it will be a major improvement if our query processing method can be extended to any fully homomorphic encryption scheme. In addition, users will typically want to perform multi-keyword search queries with logical operations such as AND, OR and NOT; therefore, the question of providing more query functionality that supports these operations is also an open area of research.

Bibliography

[1] Energy theft in the advanced metering infrastructure. In E. Rome and R. Bloomfield, editors, *Critical Information Infrastructures Security*, volume 6027 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2010.

[2] NIST IR 7628. Guidelines for smart grid cyber security. `http://www.nist.gov/smartgrid/upload/nistir-7628_total.pdf`, September 2010.

[3] G. Ács and C. Castelluccia. I have a dream!: Differentially private smart metering. In *Proceedings of the 13th International Conference on Information Hiding*, IH'11, pages 118–132, Berlin, Heidelberg, 2011. Springer-Verlag.

[4] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order preserving encryption for numeric data. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, SIGMOD '04, pages 563–574, New York, NY, USA, 2004. ACM.

[5] ZigBee Alliance. Zigbee smart energy - greener homes, business and utilities., Sep 2015.

[6] L.T. Berger and K. Iniewski. *Smart Grid Applications, Communications, and Security.* Wiley, 1st edition, 2012.

[7] A. Bogdanov, L.R. Knudsen, G. Leander, C. , C.Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin, and C. Vikkelsoe. Small-footprint block cipher design – how far can you go?, 2007.

[8] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill. Order-preserving symmetric encryption. In *Proceedings of the 28th Annual International Conference on Advances in Cryptology: The Theory and Applications of Cryptographic Techniques*, EUROCRYPT '09, pages 224–241, Berlin, Heidelberg, 2009. Springer-Verlag.

[9] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In Christian Cachin and JanL. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522. Springer Berlin Heidelberg, 2004.

[10] D. Boneh, C. Gentry, S. Halevi, F. Wang, and D.J. Wu. Private database queries using somewhat homomorphic encryption. *Applied Cryptography and Network Security (ACNS)*, 7954:102–118, 2013.

[11] J.W. Bos, K. Lauter, J. Loftus, and M. Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. In Stam M, editor, *Cryptography and Coding – 14th IMA International Conference*, volume 8308 of *Lecture notes in computer science*, pages 45–64. Springer, 2013.

[12] Z. Braserski and V Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. *Advances in Cryptology - CRYPTO*, 6841:505–524, 2011.

[13] Prepared by Litos Strategic Communication. The smart grid, an introduction. Technical report, US Department of Energy, 2013.

[14] C. Castelluccia, E. Mykletun, and G. Tsudik. Efficient aggregation of encrypted data in wireless sensor networks. In *Mobile and Ubiquitous Systems: Networking and Services, 2005. MobiQuitous 2005. The Second Annual International Conference on*, pages 109–117, July 2005.

[15] M. Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. *Advances in Cryptology – EUROCRYPT 2010*, 6110:24–43, 2010.

[16] Z. Erkin, J.R. Troncoso-Pastoriza, R.L. Lagendijk, and F. Perez-Gonzalez. Privacy-preserving data aggregation in smart metering systems: an overview. *Signal Processing Magazine, IEEE*, 30(2):75–86, March 2013.

[17] Z. Erkin and G. Tsudik. Private computation of spatial and temporal power consumption with smart meters. In F. Bao, P. Samarati, and J. Zhou, editors, *Applied Cryptography and Network Security*, volume 7341 of *Lecture Notes in Computer Science*, pages 561–577. Springer Berlin Heidelberg, 2012.

[18] European Telecommunications Standards Institute (ETSI). Open smart grid protocol (OSGP), 2012.

[19] Y. Gahi, M. Guennoun, and K. El-Khatib. A secure database system using homomorphic encryption schemes. In *The Third International Conference on Advances in Databases, Knowledge, and Data Applications*, January 2011.

[20] Y. Gahi, M. Guennoun, Z. Guennoun, and K. El-Khatib. Privacy preserving scheme for location-based services. *Journal of Information Security*, 3(2):105–112, 2012.

[21] F. Garcia and B. Jacobs. Privacy-friendly energy-metering via homomorphic encryption. In J. Cuellar, J. Lopez, G. Barthe, and A. Pretschner, editors,

Security and Trust Management, volume 6710 of *Lecture Notes in Computer Science*, pages 226–238. Springer Berlin Heidelberg, 2011.

[22] C. Gentry. *A fully homomorphic encryption scheme.* PhD thesis, Stanford University, 2009. crypto.stanford.edu/craig.

[23] C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC '09 Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009.

[24] P. Golle, J. Staddon, and B. Waters. Secure conjunctive keyword search over encrypted data. In Markus Jakobsson, Moti Yung, and Jianying Zhou, editors, *Applied Cryptography and Network Security*, volume 3089 of *Lecture Notes in Computer Science*, pages 31–45. Springer Berlin Heidelberg, 2004.

[25] B. Harrison. On the reducibility of cyclotomic polynomials over finite fields. *The American Mathematical Monthly*, 114(9):813–818, 2007.

[26] H. Hesse and C. Matthies. Introduction to homomorphic encryption. Technical report, Universitatsverlag Potsdam, December 2013.

[27] J. Hoffstein, J. Pipher, and J. Silverman. Ntru: A ring-based public key cryptosystem. In JoeP. Buhler, editor, *Algorithmic Number Theory*, volume 1423 of

Lecture Notes in Computer Science, pages 267–288. Springer Berlin Heidelberg, 1998.

[28] B. Hore, S. Mehrotra, M. Canim, and M. Kantarcioglu. Secure multidimensional range queries over outsourced data. *The VLDB Journal*, 21(3):333–358, 2012.

[29] M. Kaku. *Physics of the Future.* Anchor Books, 1 edition, February 2012.

[30] S. Keemink and B. Roos. Security analysis of dutch smart metering systems, jul 2008.

[31] K. Kursawe, G. Danezis, and M. Kohlweiss. Privacy-friendly aggregation for the smart-grid. In *Proceedings of the 11th International Conference on Privacy Enhancing Technologies*, PETS'11, pages 175–191, Berlin, Heidelberg, 2011. Springer-Verlag.

[32] K. Kursawe and C. Peters. Structural weaknesses in the open smart grid protocol. Cryptology ePrint Archive, Report 2015/088, 2015. `http://eprint.iacr.org/`.

[33] A. López-Alt, E. Tromer, and V. Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, STOC '12, pages 1219–1234, New York, NY, USA, 2012. ACM.

[34] P. Maniatis, M. Roussopoulos, E. Swierk, K. Lai, G. Appenzeller, X. Zhao, and M. Baker. The mobile people architecture. *ACM Mobile Computing and Communications Review*, 3:36–42, 1999.

[35] P. McDaniel and S. McLaughlin. Security and privacy challenges in the smart grid. *Security Privacy, IEEE*, 7(3):75–77, May 2009.

[36] M. Naehrig, K. Lauter, and V. Vaikuntanathan. Can homomorphic encryption be practical? In *CCSW '11 Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, pages 113–124, 2011.

[37] Office of Electricity Delivery & Energy Reliability. Smart grid. `http://energy.gov/oe/services/technology-development/smart-grid`.

[38] Hydro One. Frequently asked questions - my meter. `http://www.hydroone.com/MyHome/MyAccount/MyMeter/Pages/FAQ.aspx`, August 2015.

[39] S. Palamakumbura and H. Usefi. Database query privacy using homomorphic encryptions. In *Information Theory (CWIT), 2015 IEEE 14th Canadian Workshop on*, pages 71–74, July 2015.

[40] S. Peter, K. Piotrowski, and P. Langendoerfer. On concealed data aggregation for WSNS. In *Consumer Communications and Networking Conference, 2007. CCNC 2007. 4th IEEE*, pages 192–196, Jan 2007.

[41] P.Paillier. Public-key cryptosystems based on composite degree residuosity classes. In J. Stern, editor, *Advances in Cryptology EUROCRYPT 99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer Berlin Heidelberg, 1999.

[42] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[43] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.

[44] E. Shi, J. Bethencourt, T.H. Chan, D. Song, and A. Perrig. Multi-dimensional range query over encrypted data. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, SP '07, pages 350–364, Washington, DC, USA, 2007. IEEE Computer Society.

[45] B. Soubhagya, M.G. Venifa, and A.C.J. Jeya. A homomorphic encryption technique for scalable and secure sharing of personal health record in cloud computing. *International Journal of Computer Application*, 67(11):40–44, April 2013.

[46] D. Stehlé and R. Steinfeld. Making ntruencrypt and ntrusign as secure as standard worst-case problems over ideal lattices. Cryptology ePrint Archive, Report 2013/004, 2013. `http://eprint.iacr.org/`.

[47] www.bluetooth.com. Smart energy and bluetooth technology, Sep 2015.

[48] K. Zickuhr. Location based services. `http://www.pewinternet.org/2013/09/12/location-based-services/`, September 2013.