Fully homomorphic encryption

CS4796: Joint Senior Honours Project

Gergely Flamich

Supervisors:
Dr Sophie Huczynska, Prof Steve Linton

School of Computer Science
University of St Andrews
Scotland

Abstract

Declaration We declare that the material submitted for assessment is my own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated. The main text of this project report is NN,NNN* words long, including project specification and plan. In submitting this project report to the University of St Andrews, I give permission for it to be made available for use in accordance with the regulations of the University Library. I also give permission for the report to be made available on the Web, for this work to be used in research within the University of St Andrews, and for any software to be released on an open source basis. I retain the copyright in this work, and ownership of any resulting intellectual property.

Contents

1 Introduction

2 Mathematical Preliminaries

Throughout this work some basic concepts will be used, the most important of which will be given below.

Definition 2.1. Strings over an alphabet: Let A be a finite set of symbols, called an **alphabet** the strings over A denoted A^* is the set of all finite length sequences of symbols from A, including ϵ, the empty string. Formally:

$$A^* = \bigcup_{n \in \mathbb{N} \cup \{0\}} A^n.$$

Here $\epsilon = () \in A^0$, the empty tuple. When talking about strings from A^*, instead of $s = (a_1, a_2, \dots, a_n) \in A^*$, we will just write $a_1 a_2 \dots a_n$.

A natural property of strings is their length, i.e. the number of symbols they are composed of. Thus, let $| \cdot | : A \mapsto \mathbb{N} \cup \{0\}$ be defined as

$$\forall s \in A^* \quad |s| = n \Leftrightarrow s \in A^n$$

A natural operation on strings is concatentaion. Formally, let A be an alphabet. Then, let $+ : A^* \times A^* \mapsto A^*$, such that

$$\forall s_1 = a_1 \dots a_n, s_2 = b_1 \dots b_m \in A^*. \quad s_1 + s_2 = a_1 \dots a_n b_1 \dots b_m \in A^*.$$

Definition 2.2. Group: Let G be a non-empty set and $\circ : G \times G \mapsto G$ a binary operation on G. Then we will say that $(G, \circ)$ is a group if the following axioms hold:

- **G1:** $\forall a, b, c \in G$ we have $(a \circ b) \circ c = a \circ (b \circ c)$. (associativity)
- **G2:** $\exists 1 \in G. \forall a \in G$ we have $1 \circ a = a \circ 1 = a$. (existence of an identity)
  It is easy to show that if such an element 1 exists, then it is unique. We will call this unique element the **identity** of G.
- **G3:** $\forall a \in G \, \exists a^{-1} \in G$ such that $a \circ a^{-1} = a^{-1} \circ a = 1$. (existence of inverses)
  It is easy to show that if this axiom holds then for every a the corresponding a^{-1} is unique. We will call this unique element the **inverse** of a.

If G is finite, we call $(G, \circ)$ a **finite group**.

Definition 2.3. Big-O notation: Let $f, g : \mathbb{N} \mapsto \mathbb{R}$ be functions. Then we will write

$$f(n) = \mathcal{O}(g(n)) \quad \text{(and say } f \text{ is of order big oh of } g\text{)}$$

if and only if

$$\exists M \in \mathbb{R}. \exists N \in \mathbb{N}. \quad \forall n \geq N \Rightarrow |f(n)| \leq M|g(n)|.$$

Intuitively, this means that f grows **at most** as fast as g.

3 A Very Brief History of Ciphers

The history of ciphers goes back at least 2000 years. We will now examine the cipher now named after Julius Ceasar, who used the following method to obfuscate his correspondance for his adversaries:

Ceasar Cipher : Take the message (written using symbols from the Latin alphabet) that we wish to obfuscate. Replace each symbol with the one that comes 3 places after it in the alphabet. For letters at the end where we could not shift, we "wrap around" to the beginning of the alphabet and carry on counting from there. For example

$$\textit{Alea iacta est} \quad \to \quad \textit{Dohd mdfzd hxz.}$$

To decrypt a message, we simply shift backwards by 3 positions, e.g.

$$\textit{Zhqm, zmgm, zmgm} \quad \to \quad \textit{Veni, vidi, vici.}$$

We can further generalise this concept to arrive at the definiton of *shift ciphers*, where instead of the fixing the shift to 3, we pick a key $k \in \mathbb{N}$, and we then shift k places forward (and backwards). With shift ciphers in mind, we now formally defined some key concepts that will be used throughout this work.

Definition 3.1. Cipher: Let $\mathcal{M}, \mathcal{C}, \mathcal{K}_{\texttt{Enc}}, \mathcal{K}_{\texttt{Dec}}$ be alphabets. We will refer to $\mathcal{M}$ as the **message space** and to $\mathcal{C}$ as the **cipher space**. A cipher C over $\mathcal{M}$, $\mathcal{K}_{\texttt{Enc}}$ and $\mathcal{K}_{\texttt{Dec}}$ is a tuple $(\texttt{Gen}, \texttt{Enc}, \texttt{Dec})$, where

- $\texttt{Gen}$ is the **key generation algorithm**, which outputs a key $(k_{\texttt{Enc}}, k_{\texttt{Dec}}) \in (\mathcal{K}^*_{\texttt{Enc}} \times \mathcal{K}^*_{\texttt{Dec}})$, chosen according to some distribution. We refer to the set $\mathcal{K} \subseteq \mathcal{K}^*_{\texttt{Enc}} \times \mathcal{K}^*_{\texttt{Dec}}$ of all possible outputs of $\texttt{Gen}$ as the **key space** of C. If for all outputs there is a polynomial time function f such that $f(k_{\texttt{Enc}}) = k_{\texttt{Dec}}$, then we call C a **symmetric cipher** and instead of the tuple we just write k. Otherwise we call C an **asymmetric cipher**. Since $\texttt{Gen}$ is usually a probabilistic algorithm, we will denote generating its output by $k \leftarrow \texttt{Gen}$ instead of $k = \texttt{Gen}$ to emphasize the randomised nature of the function.

- $\texttt{Enc} : \mathcal{K}^*_{\texttt{Enc}} \times \mathcal{M} \mapsto \mathcal{C}$ is the **encryption algorithm**, which takes a encryption key k and a message m and outputs its encoding c. We will often refer to the message as the **plaintext** and to the encrypted message as **ciphertext**.

  Note: $\texttt{Enc}$ is not necessarily deterministic. When it is probabilistic, we will write $c \leftarrow \texttt{Enc}_k(m)$ instead of $c = \texttt{Enc}_k(m)$.

- $\texttt{Dec} : \mathcal{K}^*_{\texttt{Dec}} \times \mathcal{C} \mapsto \mathcal{M}$ is the **decryption algorithm**, which takes some decryption key k and some ciphertext c and outputs its correspnding plaintext m.

  Note: $\texttt{Dec}$ is **always** deterministic (otherwise the scheme could never be assumed to be correct).

Finally, we require the correctness of C, concretely

$$\forall m \in \mathcal{M}, \forall (k, k') \in \mathcal{K}. \quad \texttt{Dec}(k', \texttt{Enc}(k, m)) = m,$$

i.e. that the cipher does not change its contents.

Note: For classical ciphers, we usually consider $\mathcal{M} = \mathcal{C}$ to be the English (or Latin) alphabet. For modern crypto systems we will always work with bitstrings, i.e. $\mathcal{M}$, $\mathcal{C}$ and $\mathcal{K}$ will be some subset of $\{0,1\}^*$.

Equivalences of alphabets Consider any alphabet $\mathcal{M} = \{m_1, \dots m_n\}$. It will be often convenient to consider the encoding of such an alphabet as something that may be studied and manipulated more easily. Formally, by an encoding we mean a bijection f between our alphabet and some other set, where calculating values of f and f^{-1} can both be done in polynomial time. Such an encoding may be using ASCII for characters of the English alphabet or associating some symbols with the elements of $\mathbb{Z}_n$. If there is such a bijection between two alphabets, since one can be efficiently transformed into the other, we will consider them equivalent.

Example Let us now revisit shift ciphers. In this case, for a set of n symbols, it will be easier to consider the encoding as elements of the additive group $\mathbb{Z}_n$. First, it is easy to see that we are dealing with a symmetric cipher, and since a shift by k and any $k + xn, x \in \mathbb{N}$ is going to be the same, $\mathcal{K} = \mathbb{Z}_n$. Then,

- `Gen`: `Gen` is uniformly picking an element from $\mathbb{Z}_n$.
- `Enc`: We note that if we relabel our symbols to their indices (i.e. $m_i \to i$), then we can express our encryption function as
$$\texttt{Enc}(k, i) = i + k \mod n.$$
- `Dec`: Similarly, performing the above relabeling, we get
$$\texttt{Dec}(k, i) = i - k \mod n.$$

Now, checking the correctness of C_{shift}: Let $k, m \in \mathbb{Z}_n$

$$\texttt{Dec}(k, \texttt{Enc}(k, m)) = m + k \mod n - k \mod n = m + k - k \mod n = m \mod n = m.$$

Since k and m were arbitrary, it holds $\forall m, k \in \mathbb{Z}_n$.

Kerckhoffs' principle Throughout the history of cryptography it has become evident that one must not rely on the assumption that the adversary cannot obtain every detail about one's cryptosystem. This lead to the following important guiding principle in the design of new schemes:

> The security of a scheme must lay in the key and not the scheme.

4 Perfect Security

In order for us to be able to analyse our schemes, we must have two well defined notions: what does it mean that a scheme is *secure* and how *powerful* is the adversary who wishes to break our scheme.

The adversary We will start from a rather paranoid, but very useful perspective: we will assume that the adversary has unbounded computational power. This means that we will have to fend off attacks that may involve computing arbitrary *decidable* functions, no matter how long they take to calculate. Also, by Kerckhoff's principle we assume that the adversary has perfect knowledge of both our encryption and decryption functions.

Security We will also impose a very stringent constraint on what we consider secure.

Definition 4.1. Perfect Security We say that a scheme Π is perfectly secure if

$$\forall m \in \mathcal{M}, c, c' \in \mathcal{C}. \quad \mathbb{P}(M = m \mid C = c) = \mathbb{P}(M = m \mid C = c').$$

Indistinguishability We will require that given any ciphertext and a uniformly random string from the cipher alphabet, the adversary cannot *distinguish* the two strings whatsoever. To formalise this, we will first need to define a **security experiment**:

Let $\Pi = (\texttt{Gen}, \texttt{Enc}, \texttt{Dec})$ be a scheme, and $\mathcal{A}$ be any adversary. Then, we will define $\texttt{Expr}^{\texttt{eav}}_{\Pi,\mathcal{A}}$ as follows:

1. $\mathcal{A}$ generates two messages m_0 and m_1 from the message space $\mathcal{M}$.

2. A random bit b is chosen from $\{0, 1\}$.

3. Put $k \leftarrow \texttt{Gen}()$ and $c = \texttt{Enc}_k(m_b)$.

4. $\mathcal{A}$ is given c, and outputs $b' = \{0, 1\}$

5. The experiment outputs 1 if $b = b'$ and write $\texttt{Expr}^{\texttt{eav}}_{\mathcal{A},\Pi} = 1$, and 0 otherwise. If the output is 1, we say that $\mathcal{A}$ **succeeds**.

Now, we are ready for our first definition of security:

Definition 4.2. Perfect Indistinguishability We say that a scheme Π is perfectly indistinguishable iff for all adversaries $\mathcal{A}$

$$\mathbb{P}(\texttt{Expr}^{\texttt{eav}}_{\mathcal{A},\Pi} = 1) = \frac{1}{2}.$$

Theorem 4.1. *A scheme Π is perfectly secure if and only if it is perfectly indistinguishable.*

Proof. haha □

Theorem 4.2. *A necessary condition for a scheme $\Pi = (\texttt{Gen}, \texttt{Enc}, \texttt{Dec})$ to be perfectly secure, is that $|\mathcal{K}| \geq |\mathcal{M}|$.*

Proof. Assume $|\mathcal{K}| < |\mathcal{M}|$. We will show that this implies that perfect security is broken.
The idea is to use the fact that if we are able to brute-force search through all the keys, we will learn some information about the keys and the messages.

For every $c \in \mathcal{C}$, let $\mathcal{M}_c = \{m \mid m = \texttt{Dec}_k(c) k \in \mathcal{K}\}$. Note, that since $\texttt{Dec}$ is a function, $|\mathcal{M}_c| \leq |K|$, as $\texttt{Dec}$ may map two ciphertexts to the same plaintext for two different keys, but it can never map a ciphertext to two different plaintexts for the same key. Hence, $\mathcal{M}_c \subset \mathcal{M}$ and in particular $\mathcal{M} \setminus \mathcal{M}_c \neq \emptyset$.
Now, fix $c \in \mathcal{C}$, and pick $m \in \mathcal{M} \setminus \mathcal{M}_c$. Since by definition c cannot map to m, we have $\mathbb{P}(M = m|C = c) = 0$. But then

$$\mathbb{P}(M = m) \neq 0 = \mathbb{P}(M = m|C = c)$$

which violates perfect security, so our initial assumption must be false. □

5 Computational Security

6 Public-Key Crypto Schemes

6.1 An Example: RSA

In this section we finally illustrate how the previously discussed ideas can be combined into a concrete scheme. However, before we can do that we must make a brief detour to prove some results we will be using later.

6.1.1 Mathematical set-up

Theorem 6.1. *Order Let G be a finite group and $g \in G$. Then, there exist positive integers n such that $g^n = 1$. We shall call the smallest such integer the order of g, denoted $|g|$ called the* ***order*** *of g.*

Proof. Since G is finite, there must exist $n, m \in \mathbb{N}, n \neq m$ such that

$$g^n = g^m.$$

Without loss of generality, we may assume $n > m$. Then, multiplying both sides with the inverse of g^m, we get

$$g^{n-m} = 1.$$

□

Theorem 6.2. *Let G be a finite group and $g \in G$. Then, the order of g divides the order of G.*

Proof. Assume that $|g| \nmid |G|$. Then, $|G| = q|g| + r$ for q a non-negative integer and r a positive integer. □

Corollary 6.2.1. *Let G be a finite group. Then,*

$$\forall g \in G \quad g^{|G|} = 1.$$

Proof. Let $g \in G$. By Theorem 6.2 $|G| = q|g|$ for some $q \in \mathbb{N}$. Therefore,

$$g^{|G|} = g^{q|g|} = (g^{|g|})^q = 1^q = 1.$$

□

Corollary 6.2.2. *Let G be a finite group. Then,*

$$\forall g \in G, p \in \mathbb{N} \quad g^p = g^{p \mod |G|}.$$

Proof. Let $g \in G, p \in \mathbb{N}$. Then, $p = q|G| + r$ for q non-negative integer and r a positive integer. Then, by the previous corollary $g^{|G|} = 1$, so

$$g^p = g^{q|G|+r} = (g^{|G|})^q g^r = 1^q g^r = g^r = g^{p \mod |G|}.$$

□

Before we may move on to the final corollary, we quickly make the following observation:

Theorem 6.3. *Let $n \in \mathbb{N}$. The elements x of $\mathbb{Z}_n \setminus \{0\}$ such that $\gcd(x,n) = 1$ form a group under multiplication with identity 1 and inverse $a \in \mathbb{Z}_n$, where $ax + bn = 1$ from Bèzout's identity. We denote this group by $\mathbb{Z}_n^*$*

Proof. It is very easy to check that the above indeed fulfils all group axioms. □

The order of $\mathbb{Z}_n$ is very important number-theoretically, so much so that it is usually dentoted $\phi(n) = |\mathbb{Z}_n|$ and is called the **Euler phi-function**. Now, we are ready to state the theorem of our interest:

Theorem 6.4. *(Euler's Theorem)*
Let $a, n \in \mathbb{N}$ such that $a < n$ and $\gcd(a,n) = 1$. Then,

$$a^{\phi(n)} = 1 \mod n.$$

Proof. By the above conditions $a \in \mathbb{Z}_n^*$. Then, the statement is a special case of Corollary 6.2.2, where $G = \mathbb{Z}_n^*$. □

6.1.2 The construction

7 Homomorphic Crypto Schemes

At this point, we start to finally focus on the topic of the dissertation: homomorphic schemes. Intuitively, these encapsulate the idea of encrypted comptutation over encrypted data, in sense that the adversary may at most observe the operation take place, but both the inputs and the output of it remain secret. In this section we give the formal definition, construct an example scheme known as the Paillier scheme. Finally, we close with a discussion on the limitation of homomorphic schemes.

7.1 Definition

7.2 Paillier's Scheme

Paillier encryption is an additively homomorphic crypto scheme, that operates over $\mathbb{Z}_{N^2}$ where $N = pq$ for some p, q primes. In particular, it takes advantage of the fact that $\mathbb{Z}_{N^2} \simeq \mathbb{Z}_N \times \mathbb{Z}_N^*$.

7.3 Limitations

8 Lattices

In the previous section we have seen the motivating idea behind Gentry's scheme: encrypted computation. In this section, we aim to provide a brief introduction to the tool that helped the scheme come to life: lattices. We will define what a lattice is and see some key definitions and theorems. We will close the section with a construction of a simple lattice-based crypto scheme to illustrate how the theory may be used to construct secure schemes with the appropriate choice of a trapdoor.

8.1 GGH

9 Fully Homomorphic Encryption

9.1 Dr Craig Gentry's Scheme