B. Comp. Dissertation

MersennePKC

A public-key cryptosystem based on Mersenne primes

by

Thenaesh Elango

Department of Computer Science
School of Computing
National University of Singapore

2018/2019

B. Comp. Dissertation

MersennePKC

A public-key cryptosystem based on Mersenne primes

by

Thenaesh Elango

Department of Computer Science
School of Computing
National University of Singapore

2018/2019

Project No: H216040
Supervisor: Asst. Prof. Divesh Aggarwal
Co-Supervisor: Prof. Frank Stephan

Deliverables:
Report: 1 Volume

Abstract

With quantum computers soon becoming a reality, classical cryptosystems will soon no longer be viable. It is thus necessary to create new post-quantum cryptosystems that are secure against adversaries with quantum computers. We present MersennePKC, a cryptosystem adapted from (Aggarwal, Joux, Prakash, & Santha, 2018) and thought to be secure against quantum computers. We experimentally determine essential subprocedures for MersennePKC and give justification for its correctness, efficiency and security.

Acknowledgements

I would like to convey my sincere thanks and appreciation to my FYP supervisor Asst. Prof. Divesh Aggarwal for all the assistance he has rendered me over the course of this project. I would also like to thank him for introducing me to the world of computational complexity when he taught me CS5230 in AY17/18.

I would also like to thank my co-supervisor Prof. Frank Stephan for the insightful weekly meetings and comments on the project.

Contents

Chapter 1

Introduction

1.1 Classical Cryptosystems

As of 2019, the public-key cryptosystems most commonly used in practice are based on two problems thought to be hard on classical computers: the integer factorisation problem and the discrete logarithm problem. These problems, while thought to be hard on classical computers, are unfortunately easy to solve on quantum computers, which presents a major security challenge with the advent of quantum computers.

We begin by defining these problems and their associated cryptosystems in this section, in preparation for the generalisation and quantum attack in the following section.

1.1.1 Cryptosystems based on Integer Factorisation

Definition 1.1.1. Integer Factorisation Problem Let $n \in \mathbb{N}$ be a large, publicly-known constant. Given $n = pq$ where $p, q \in \mathbb{Z}_n$ are prime, determine p and q.

The hardness of integer factorisation is used in the RSA public key cryptosystem, which is one commonly used cryptosystem today. The RSA cryptosystem is given by the tuple $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$, where:

- $\mathcal{P} = \mathbb{Z}$ is the set of plaintexts
- $\mathcal{C} = \mathbb{Z}$ is the set of ciphertexts
- $\mathcal{K} = \{(n, E, D, p, q) \in \mathbb{Z}^5 : p, q \text{ prime}, n = pq, ED \equiv 1 \pmod{(p-1)(q-1)}\}$ is the set of keys, where for each key (n, E, D, p, q):
  - (n, E) is the public component
  - (D, p, q) is the private component
- $\mathcal{E} = \{e_k : \mathcal{P} \to \mathcal{C} : k = (n, E, D, p, q)) \in \mathcal{K}, e_k(x) \equiv x^E \pmod{n}\}$ is the set of encryption functions
- $\mathcal{D} = \{d_k : \mathcal{C} \to \mathcal{P} : k = (n, E, D, p, q) \in \mathcal{K}, d_k(y) \equiv y^D \pmod{n}\}$ is the set of decryption functions

1.1.2 Cryptosystems based on Discrete Logarithm

Definition 1.1.2. Discrete Logarithm Problem Let $(G, *)$ be a cyclic group with publicly-known generator $\mathcal{P}$. Given $\mathcal{Q} = n\mathcal{P}$ for some $n \in \mathbb{N}$, determine n.

The hardness of the discrete logarithm problem, with the group $(G, *)$ set to the multiplicative group $(\mathbb{F}_p, \times)$ for some large prime p, is used in the ElGamal cryptosystem, which is another commonly used cryptosystem. The ElGamal cryptosystem is given by:

- $\mathcal{P} = \mathbb{F}_p$ is the set of plaintexts
- $\mathcal{C} = \mathbb{F}_p^2$ is the set of ciphertexts

- $\mathcal{K} = \{(p, \theta, A, a) \in \mathbb{F}_n^4 : A = \theta^a\}$ is the set of keys, where for each key (p, θ, A, a):
  - (p, θ, A) is the public component
  - a is the private component
- $\mathcal{E} = \{e_k : \mathcal{P} \times \mathbb{F}_p \to \mathcal{C} : k = (p, \theta, A, a) \in \mathcal{K}, e_k(x, b) = (\theta^b, A^b)\}$ is the set of encryption functions, where $b \in \mathbb{F}_p$ is a number randomly chosen by the encryptor
- $\mathcal{D} = \{d_k : \mathcal{C} \to \mathcal{P} : k = (p, \theta, A, a) \in \mathcal{K}, d_k(B, y) = yB^{-a}\}$ is the set of decryption functions

1.2 Threat Posed by Quantum Computers

The hard problems presented in the previous section turn out to be instances of a more general problem which has an efficient solution on quantum computers. The result is that all classical cryptosystems based on those hard problems can be broken easily with quantum computers.

1.2.1 Hidden Subgroup Problem

The integer factorisation problem and the discrete logarithm problem are instances of what is known as the hidden subgroup problem (HSP). They are, in fact, instances of HSP on *finitely-generated Abelian groups* (Fin-Ab-HSP).

Before we can define HSP and Fin-Ab-HSP, we need to define the notion of *subgroup hiding.*

Definition 1.2.1. Subgroup Hiding Let $(G, *)$ be a group. Let K be a subgroup of G and X be an arbitrary set. A function $f : G \to X$ is said to hide H if:

- $\forall a \in G\, \exists x \in X\, [f(a * K) = \{x\}]$ is a singleton set in X i.e. f maps every element in the same coset of K to the same value in X
- $\forall a, b \in G\, [a * K \neq b * K \implies f(a * K) \cup f(b * K) = \emptyset]$ i.e. f maps distinct cosets of K to disjoint subsets of X

In the definition of subgroup hiding, we consider left cosets of the subgroup, but the definition also works for right cosets. We are also ultimately interested in subgroup hiding in Abelian groups (in Fin-Ab-HSP), where there is no distinction between left and right cosets.

We can now proceed to define HSP. The definition of Fin-Ab-HSP follows from that of HSP by requiring the group $(G, *)$ to be finitely-generated and Abelian.

Definition 1.2.2. HSP Let $(G, *)$ be a group. Let K be a subgroup of G and X be an arbitrary set. Given a function $f : G \to X$ that hides K, output a generating set for K.

We now proceed to show that the integer factorisation problem and discrete logarithm problem are both instances of Fin-Ab-HSP.

1.2.2 Integer Factorisation as a Fin-Ab-HSP Instance

We give a polynomial-time reduction from the integer factorisation problem to Fin-Ab-HSP. This requires reduction of integer factorisation to an intermediate problem: order finding.

Definition 1.2.3. Order Finding Problem Let $n \in \mathbb{N}$ be known. Given $a \in \mathbb{N}$ where $a < N$ and $\gcd(a, N) = 1$, find the smallest $r \in \mathbb{N}$ such that $a^r \equiv 1 \mod N$. We call this (a, n) an instance of the order finding problem with solution r.

Lemma 1.2.1. There is a polynomial-time Cook reduction from the integer factorisation problem to the order finding problem.

The proof of Lemma 1.2.1 is given in Subsection 5.3.2 of (Nielsen & Chuang, 2002). With this lemma in place, we can simply construct an instance of Fin-Ab-HSP for order finding. We use a construction taken from Figure 5.5 of (Nielsen & Chuang, 2002), which is a table that describes how to represent various problems as instances of HSP.

Lemma 1.2.2. Suppose $(a, n) \in \mathbb{N}^2$ is an instance of the order finding problem with solution r. Define the following sets:

$$\begin{aligned} G &= \mathbb{Z} \\ K &= r\mathbb{Z} \\ X &= \{a^j : j \in \mathbb{N}\} \end{aligned}$$

Let the function $f : G \to X$ be defined as $f(x) = a^x$. Then the group $(G, +)$, the hidden subgroup K, the set X and the function $f : G \to X$ form an instance of Fin-Ab-HSP whose solution is $\{r\}$.

Proof. $(\mathbb{Z}, +)$ is an Abelian group since $+$ is commutative, and is finitely-generated with generator set $\{1\}$. Let $x, y \in \mathbb{Z}$ such that $x + K$ and $y + K$ and distinct (hence disjoint) cosets of K. Then the following hold:

$$\begin{aligned} f(x + K) &= \{f(x + kr) : k \in \mathbb{N}\} \\ &= \{a^{x+kr} : k \in \mathbb{N}\} \\ &= \{a^x (a^r)^k : k \in \mathbb{N}\} \\ &= \{a^x (1)^k : k \in \mathbb{N}\} \\ &= \{a^x\} \end{aligned}$$

$$\begin{aligned} f(x + K) \cap f(y + K) &= \{a^x\} \cap \{a^y\} \\ &= \emptyset \end{aligned}$$

The first equation shows that f maps each coset to a single element, while the second equation shows that f maps distinct cosets to distinct values, thus proving that this is indeed an instance of Fin-Ab-HSP. Furthermore, K has generator set $\{r\}$, so $\{r\}$ will be the solution of this Fin-Ab-HSP instance. $\square$

1.2.3 Discrete Logarithm as a Fin-Ab-HSP Instance

We give a method to construct a Fin-Ab-HSP instance for the discrete logarithm problem using a single additional call to a Fin-Ab-HSP oracle. This construction is taken from Figure 5.5 of (Nielsen & Chuang, 2002).

Lemma 1.2.3. Suppose $N \in \mathbb{N}$ and $a, b \in \mathbb{Z}_N$ such that $\exists s \in \mathbb{Z}_p\,[b = a^s]$. Determine $r = \inf\{r \in \mathbb{N} : a^r \equiv 1 \pmod{N}\}$ by solving the order finding problem (reduce to Fin-Ab-HSP and call the oracle). Let $l \in \mathbb{Z}_r$ and define the following:

$$\begin{aligned} G &= \mathbb{Z}_r \times \mathbb{Z}_r \\ K &= (l, -ls)\mathbb{Z}_r \\ X &= \{a^j : j \in \mathbb{N}\} \end{aligned}$$

Let the function $f : G \to X$ be defined as $f(x, y) = a^{sx+y} \mod N$. Then the group $(G, +)$, the hidden subgroup K, the set X and the function $f : G \to X$ form an instance of Fin-Ab-HSP whose solution is $\{l, -ls\}$.

Proof. $(\mathbb{Z}_r \times \mathbb{Z}_r, +)$ is an Abelian group since $+$ is commutative. Since $(\mathbb{Z}_r \times \mathbb{Z}_r, +)$ is a finite Abelian group, by the classification theorem for finite Abelian groups it is isomorphic to a finite product of finitely generated groups and hence itself finitely generated. Let $\alpha = (\alpha_0, \alpha_1), \beta = (\beta_0, \beta_1) \in \mathbb{Z}_r \times \mathbb{Z}_r$ such that $\alpha + K$ and $\beta + K$ and distinct (hence disjoint) cosets of K. Then the following hold:

$$\begin{aligned} f(\alpha+K) &= \{f(\alpha_0+lx,\alpha_1-lsx) \pmod N : x\in\mathbb{Z}_r\} \\ &= \{a^{s(\alpha_0+lx)+(\alpha_1-lsx)} \pmod N : x\in\mathbb{Z}_r\} \\ &= \{a^{s\alpha_0+lsx+\alpha_1-lsx} \pmod N : x\in\mathbb{Z}_r\} \\ &= \{a^{s\alpha_0+\alpha_1} \pmod N\} \end{aligned}$$

$$\begin{aligned} f(\alpha+K)\cap f(\beta+K) &= \{a^{s\alpha_0+\alpha_1} \pmod N\}\cap\{a^{s\beta+\beta} \pmod N\} \\ &= \{f(\alpha_0,\alpha_1)\}\cap\{f(\beta_0,\beta_1)\} \\ &= \emptyset \end{aligned}$$

The second equation holds because $f(x+l,y-ls) = a^{s(x+l)+(y-ls)} = a^{sx+ls+y-ls} = a^{sx+y} = f(x,y)$, so f maps two domain points to the same codomain point if and only if the two domain points belong to the same coset of K.

The first equation shows that f maps each coset to a single element, while the second equation shows that d maps distinct cosets to distinct values, thus proving that this is indeed an instance of Fin-Ab-HSP. K has generator set $\{l,-ls\}$, so $\{l,-ls\}$ will be the solution of this Fin-Ab-HSP instance. $\square$

After solving the Fin-Ab-HSP instance for $\{l,-ls\}$, one can easily compute s, solving the discrete logarithm problem. Note that the extra Fin-Ab-HSP oracle call does not increase the hardness of the problem beyond the hardness of Fin-Ab-HSP.

1.2.4 Efficient Quantum Algorithms for Fin-Ab-HSP

We have seen that the integer factorisation problem and the discrete logarithm problem can both be reduced to Fin-Ab-HSP. Unfortunately, a class of efficient algorithms cleverly making use of the quantum Fourier transform was presented in (Shor, 1994) to solve Fin-Ab-HSP on quantum computers in polynomial time, thus breaking cryptosystems whose hardness is based on those problems.

These algorithms, together with significant progress made in constructing physical quantum computers, make it necessary for new cryptosystems to be created that are secure against adversaries armed with quantum computers.

1.3 A Post-Quantum Cryptosystem

In this report, we give a modified version of the Mersenne-prime based key-exchange protocol in (Aggarwal et al., 2018), thought to be secure against quantum computers, and experimentally determine a suitable algorithm for a major subprocedure that is required in the cryptosystem.

In this section, we give a brief description of how the cryptosystem is expected to work, deferring a full description and analysis to Chapter 4.

Let $p = 2^n - 1$ be a Mersenne prime and let $\mathcal{H}_h^n$ be the set of all n-bit strings with Hamming weight h. We take n and h to be fixed parameters of this cryptosystem. We identify the set of bitstrings $\mathbb{F}_2^n$ with the finite field $\mathbb{F}_p$, identifying both 0^n and 1^n in $\mathbb{F}_2^n$ with $0 \in \mathbb{F}_p$ and taking all arithmetic operations on n-bit strings with respect to $\mathbb{F}_p$.

We assume that Bob has a private key $(F, G) \in \mathcal{H}_h^n \times \mathcal{H}_h^n$ with a corresponding public key $H = \frac{F}{G}$. We assume that there is a publicly known randomness extractor (e.g. a hash function) $Ext : \mathcal{H}_h^n \times \mathcal{H}_h^n \to \mathbb{F}_2^\lambda$, for some $\lambda \in \mathbb{N}$.

Suppose Alice wishes to share a message $m \in \mathbb{F}_2^\lambda$ with Bob over an insecure channel. The idea is that Alice generates two random strings $A, B \in \mathcal{H}_h^n$ and sends $(C, c) = (AH + B, Ext(A, B) \oplus m)$ to Bob.

Bob's task, having received (C, c), is to recover the message m. Using his private key G, Bob first computes $D = CG = AF + BG$. Bob now needs to compute (A, B) given D. We experimentally deduce an algorithm in Chapter 3 to do this. After Bob obtains (A, B), he obtains the message by computing $Ext(A, B) \oplus c = m$.

1.4 Our Work

THis FYP is focused on the cryptosystem described in Section 1.3. In particular, we present the following:

- an experimentally-determined message extraction algorithm to obtain (A, B) from $D = AF + BG$
- an implementation of the cryptosystem with this message extraction algorithm
- analysis of the correctness, efficiency and security of the cryptosystem

1.5 Structure of this Report

This report is split into 5 chapters.

In Chapter 1 (this chapter), we explain why post-quantum cryptography is needed, give a proposal for a a post-quantum cryptosystem based on an existing one defined in (Aggarwal et al., 2018) and explain our tasks and contributions.

In Chapter 2, we give preliminaries and definitions used throughout the rest of this report (and even in Chapter 1).

In Chapter 3, we experimentally determine an algorithm to extract $(A, B) \in \mathcal{H}_h^n \times \mathcal{H}_h^n$ given $D = AF + BG$ (where $F, G \in \mathcal{H}_h^n$). This is an essential part of the cryptosystem. We give justifications as to the correctness of the algorithm from experimental data.

In Chapter 4, we define the cryptosystem proper. We give assumptions and justifications for the security and efficiency of the cryptosystem.

In Chapter 5, we give some implementation details of the cryptosystem and our concluding remarks.

Chapter 2

Preliminaries & Definitions

2.1 Mersenne Primes

Definition 2.1.1. Mersenne Prime We call $p \in \mathbb{N}$ a Mersenne prime if p is a prime number and $\exists n \in \mathbb{N}$ such that $p = 2^n - 1$ and n is prime.

We note that not all integers of the form $2^n - 1, n \in \mathbb{N}$ are prime numbers. A simple counterexample would be that $2^4 - 1 = 15 = 3 \times 5$ is not a prime number. We therefore rely on a known table of Mersenne primes p and their exponents n, such as Table A000043 in (Sloane et al., 2008).

2.2 Bitstrings

Throughout this report, we let $\mathcal{B}_n = \mathbb{F}_2^n - \{1^n\}$ be defined for every n for which $2^n - 1$ is a Mersenne prime. We define the following bijection:

$$\mu : \mathcal{B}_n \to \mathbb{F}_{2^n-1}$$
$$\mu = (x_1, \ldots, x_n) \mapsto \sum_{i=1}^{n} 2^{i-1} x_i$$

We sometimes write an element $x \in \mathcal{B}_n$ by its bit representation $x_n \dots x_1$ in big-endian order, where each $x_i \in \{0, 1\}$. In such situations, we will also use b^k to denote the k-bit string of all bs, where $b \in \{0, 1\}$ and $k \in \mathbb{N}_{\leq n}$.

2.2.1 Identification of $\mathcal{B}_n$ with $\mathbb{F}_{2^n-1}$

Definition 2.2.1. Addition and Multiplication on $\mathcal{B}_n$ We define the addition and multiplication operators on $\mathcal{B}_n$ the following way:

$$\forall a, b \in \mathcal{B}_n \, [a + b = \mu^{-1}(\mu(a) + \mu(b))]$$
$$\forall a, b \in \mathcal{B}_n \, [a * b = \mu^{-1}(\mu(a) * \mu(b))]$$

We note that $2^n - 1$ is prime (by requirement that it be a Mersenne prime), so $\mathbb{F}_{2^n-1}$ is a field with the usual addition and multiplication modulo $2^n - 1$. Letting the additive and multiplicative identities in $\mathcal{B}_n$ be 0^n and $0^{n-1}1$ respectively, we then see that μ induces a ring structure on $\mathcal{B}_n$ (with the addition and multiplication operators from Definition 2.2.1) and is itself a ring isomorphism between $\mathcal{B}_n$ and $\mathbb{F}_{2^n-1}$.

Therefore, $\mathcal{B}_n$ and $\mathbb{F}_{2^n-1}$ are isomorphic as fields and we can use their elements and (addition and multiplication) operators interchangeably from this point onwards without making a distinction between them. We will henceforth rarely use μ explicitly (mostly to avoid confusing it with the mean of a random variable), though it should be clear from context when used.

We can perform addition in $\mathcal{B}_n$ in $O(n)$ by the association above, since addition of two n-bit numbers modulo $2^n - 1$ can be performed in $O(n)$. We can perform multiplication in $\mathcal{B}_n$ in $O(n \log n \log \log n)$ using the algorithm given in (Schönhage & Strassen, 1971).

2.2.2 Bit Operations in $\mathcal{B}_n$

Definition 2.2.2. We define the following cyclic bit shift operators $s_{\mathcal{L}}$ and $s_{\mathcal{R}}$ the following way:

$$\begin{aligned} s_{\mathcal{L}}(x_n \dots x_1) &= x_{n-1} \dots x_1 x_n \\ s_{\mathcal{R}}(x_n \dots x_1) &= x_1 x_n \dots x_2 \end{aligned}$$

Lemma 2.2.1. Let $x \in \mathcal{B}_n$. Then $\mu(s_{\mathcal{L}}(x)) = \mu(a) \times 2$ and $\mu(s_{\mathcal{R}}(x)) = \mu(a) \times 2^{-1}$.

Proof. Write $x \in \mathcal{B}_n$ as $x_n \dots x_1$. There are two cases: $x_n = 0$ and $x_n = 1$. Suppose $x_n = 0$. Then:

$$\begin{aligned} \mu(x_n \dots x_1) \times 2 &= x_{n-1}2^{n-1} + \dots + x_1 2^1 \\ &= \mu(x_{n-1} \dots x_1 0) \\ &= \mu(x_{n-1} \dots x_2 x_n) \\ &= \mu(s_{\mathcal{L}}(x_n \dots x_1)) \end{aligned}$$

Suppose $x_n = 1$. Then:

$$\begin{aligned} \mu(x_n \dots x_1) \times 2 &= x_n 2^n + x_{n-1}2^{n-1} + \dots + x_1 2^1 \\ &= 1 + x_{n-1}2^{n-1} + \dots + x_1 2^1 \\ &= x_n 2^0 + x_{n-1}2^{n-1} + \dots + x_1 2^1 \\ &= \mu(x_{n-1} \dots x_1 0) \\ &= \mu(x_{n-1} \dots x_1 x_n) \\ &= \mu(s_{\mathcal{L}}(x_n \dots x_1)) \end{aligned}$$

This proves that $\mu(s_{\mathcal{L}}(x)) = \mu(a) \times 2$. Since $s_{\mathcal{R}}$ is the inverse operation of $s_{\mathcal{L}}$, it holds that $\mu(s_{\mathcal{R}}(x)) = \mu(a) \times 2^{-1}$. $\square$

We can now associate the operation $a \mapsto a \times 2^k$, for $k \in \mathbb{N}, a \in \mathbb{F}_{2^n-1}$ with the operation $s_{\mathcal{L}}^k$ and associate the operation $a \mapsto a \times 2^{-k}$ with $s_{\mathcal{R}}^k$. This association makes it possible to perform $a \mapsto a \times 2^{\pm k}$ on $\mathbb{F}_{2^n-1}$ in amortised $O(1)$ by cyclic bit shifting.

Definition 2.2.3. For any $x, y \in \mathcal{B}_n$, define $a \oplus b$ to be the sum of a and b regarded as elements of the vector space $\mathbb{F}_2^n$ over the field $\mathbb{F}_2$ (i.e. the bitwise exclusive-or operation).

We note that the $\oplus$ operation on $\mathcal{B}_n$ can clearly be performed in $O(n)$.

2.2.3 Hamming Weight

Definition 2.2.4. Let $x = x_n \dots x_1 \in \mathcal{B}_n$. Then the *Hamming weight* of x is the cardinality of the set $\{i \in \mathbb{Z}_n : x_i = 1\}$.

Throughout this report, we define $\mathcal{H}_h^n \subseteq \mathcal{B}_n$ to be the set of all bitstrings in $\mathcal{B}_n$ with Hamming weight h.

Lemma 2.2.2. Suppose $A, B \in \mathcal{B}_n$. Then the following hold:

$$Ham(A + B) \leq Ham(A) + Ham(B)$$
$$Ham(A * B) \leq Ham(A) * Ham(B)$$

The above lemma is given as Lemma 2 in (Aggarwal et al., 2018) and proven there.

2.3 Statistics

Definition 2.3.1. Normal Distribution We say that a random variable follows a normal distribution with mean μ and variance σ^2, written $X \sim N(\mu, \sigma^2)$ if the probability mass function f_X is given by:

$$f_X(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp{(-\frac{(x-\mu)^2}{2\sigma^2})}$$

Lemma 2.3.1. Gaussian Tail Bound Let $X \sim N(\mu, \sigma^2)$. Then:

$$\Pr[|X-\mu| > t] \leq 2\exp\left(-\frac{t^2}{2\sigma^2}\right) \tag{2.1}$$

$$\Pr[X-\mu > t] \leq \exp\left(-\frac{t^2}{2\sigma^2}\right) \tag{2.2}$$

$$\Pr[X-\mu < -t] \leq \exp\left(-\frac{t^2}{2\sigma^2}\right) \tag{2.3}$$

The Gaussian tail bound (2.1) is a slightly looser version of Lemma 2.1 in (Samorodnitsky, 1991). (2.2) and (2.3) are one-sided versions, derived from the fact that the normal distribution is symmetric about the mean.

2.4 Security

We give some important security definitions, most of which will be used in Chapter 4.

Definition 2.4.1. Advantage Let $\mathcal{X}_0$ and $\mathcal{X}_1$ be probability distributions. Let $\mathcal{A}$ be a Turing machine which takes an input that is from one of the two probability distributions and attempts to output $b \in \{0,1\}$ if the input is from $\mathcal{X}_b$. Let $X_0 \sim \mathcal{X}_0$ and $X_1 \sim \mathcal{X}_1$. We define the *advantage* of $\mathcal{A}$ in distinguishing X_0 and X_1 by:

$$Adv^{\mathcal{A}}(X_0, X_1) = |\Pr[\mathcal{A}(X_0) = 1] - \Pr[\mathcal{A}(X_1) = 1]|$$

2.4.1 Public-Key Cryptosystems

Definition 2.4.2. Public Key Cryptosystem A public key cryptosystem is defined as a tuple $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$, where:

- $\mathcal{P}$ is the set of possible plaintexts.
- $\mathcal{C}$ is the set of possible ciphertexts.

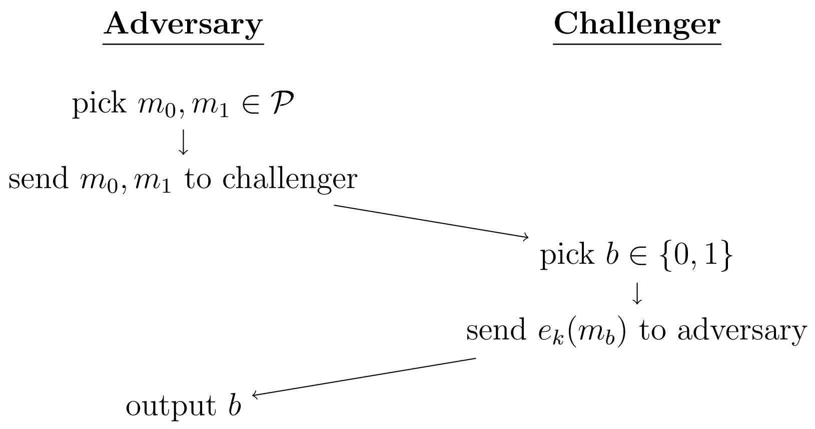
- $\mathcal{K}$ is the set of possible keys, each of which has a private component and a public component.

- $\mathcal{E}$ is the set of encryption functions. Each encryption function e_k depends on <u>only the public component</u> of a key $k \in \mathcal{K}$ and transforms a plaintext in $\mathcal{P}$ to a ciphertext in $\mathcal{C}$. The encryption functions may take in more than just a plaintext from $\mathcal{P}$ as input; they may also take in other values that are meant to be uniformly and randomly chosen by the encryptor from their respective sets. This is meant for ensuring semantic security by introducing randomness into the encryption process.

- $\mathcal{D}$ is the set of decryption functions. Each decryption function d_k depends on <u>both the public and private components</u> of a key $k \in \mathcal{K}$ and transforms a ciphertext in $\mathcal{C}$ to a plaintext in $\mathcal{P}$.

To avoid cumbersome definitions, we may choose to present $\mathcal{E}$ and $\mathcal{D}$ as encryption and decryption algorithms rather than sets of encryption/decryption functions. We also let $pub(\mathcal{K})$ and and $pri(\mathcal{K})$ be the set of public keys and private keys respectively in $\mathcal{K}$. Each cryptosystem has an associated security parameter λ, which represents the difficulty of breaking the cryptosystem.

We now proceed to define two notions of security for public key cryptosystems. Consider the following game between a probabilistic polynomial-time adversary and a challenger:


Adversary
Challenger
pick $m_0, m_1 \in \mathcal{P}$
send m_0, m_1 to challenger
pick $b \in \{0,1\}$
send $e_k(m_b)$ to adversary
output b

We define two kinds of security: ciphertext indistinguishability under chosen plaintext attack (IND-CPA) and ciphertext indistinguishability under chosen ciphertext attack (IND-CCA). Both have to do with how well an adversary guesses the correct b in the game illustrated above, but differ what information the adversary is allowed to access in the process of playing the game.

Suppose the encryption functions are of the form $e_k : \mathcal{P} \times A \to \mathcal{C}$, where A is the set over which some values are picked uniformly and randomly during encryption. Let $\mathcal{X}_{m_0,m_1}$ (resp. $\mathcal{Y}_{m_0,m_1}$) be the set of possible information available to the adversary who picked m_0, m_1 in the above game, in the case where the challenger chooses $b = 0$ (resp. $b = 1$).

$$\mathcal{X}_{m_0,m_1} = \{(m_0, m_1, e_k(m_0, a)) : a \in A\}$$
$$\mathcal{Y}_{m_0,m_1} = \{(m_0, m_1, e_k(m_1, a)) : a \in A\}$$

Define two oracles:

- $\mathcal{O}_k^{\mathcal{E}}$ is an encryption oracle that takes as input a plaintext $m \in \mathcal{P}$ and outputs a ciphertext $e_k(m, \dots)$, making uniform and random choices (the … part) if required by the encryption algorithm

- $\mathcal{O}_k^{\mathcal{D}}$ is a decryption oracle that takes as input a ciphertext $c \in \mathcal{C}$ and outputs a plaintext $d_k(c)$

With these definitions in place, we can proceed to define IND-CPA and IND-CCA.

Definition 2.4.3. IND-CPA Let $\mathcal{A}$ be a probabilistic polynomial time adversary with access to $\mathcal{O}_k^{\mathcal{E}}$ who picked $m_0, m_1 \in \mathcal{P}$ in the game illustrated above. Let $X \sim \mathcal{U}(\mathcal{X}_{m_0,m_1})$ and $Y \sim \mathcal{U}(\mathcal{Y}_{m_0,m_1})$ be random variables. Then:

$$Adv^{\mathcal{A}}(X, Y) \in O(2^{-\lambda})$$

Definition 2.4.4. IND-CCA Let $\mathcal{A}$ be a probabilistic polynomial time adversary with access to $\mathcal{O}_k^{\mathcal{D}}$ who picked $m_0, m_1 \in \mathcal{P}$ in the game illustrated above. Let $X \sim \mathcal{U}(\mathcal{X}_{m_0,m_1})$ and $Y \sim \mathcal{U}(\mathcal{Y}_{m_0,m_1})$ be random variables. Then, provided the adversary does not query the oracle with the ciphertext sent by the challenger:

$$Adv^{\mathcal{A}}(X, Y) \in O(2^{-\lambda})$$

IND-CPA is equivalent to the notion of *semantic security* and the two terms are often used interchangeably. In the definition of IND-CCA, if the adversary is allowed to adaptively choose ciphertexts to decrypt, it is called IND-CCA2; otherwise it is called IND-CCA1.

2.4.2 Key Exchange Mechanisms

Definition 2.4.5. Key Exchange Mechanism A key exchange mechanism is defined as a tuple $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \overline{\mathcal{E}}, \overline{\mathcal{D}})$, where:

- $\mathcal{P}$ is the set of possible plaintexts.

- $\mathcal{C}$ is the set of possible ciphertexts.
- $\mathcal{K}$ is the set of possible keys, each of which has a private component and a public component.
- $\overline{\mathcal{E}}$ is an *encapsulation algorithm* that takes as input a public key, makes random choices (like the encryption algorithm in a public key cryptosystem) and outputs a pair $(C, K) \in \mathcal{C} \times \mathcal{P}$
- $\overline{\mathcal{D}}$ is a *decapsulation algorithm* that takes as input both a private key and a ciphertext, and either outputs a plaintext or gives an error

Following the definition in Subsection 2.2 of (Aggarwal et al., 2018), we call a key exchange mechanism $(1-\delta)$-correct if, given a public key k_{pub} and a corresponding private key k_{pri}, $\Pr[\overline{\mathcal{D}}(k_{pri}, C) = K | \overline{\mathcal{E}}(k_{pub}) = (C, K)] \geq 1 - \delta$.

We can give definitions of IND-CPA and IND-CCA for key exchange mechanisms as well following that given in (Aggarwal et al., 2018), with $\mathcal{O}_k^{\overline{\mathcal{E}}}$ and $\mathcal{O}_k^{\overline{\mathcal{D}}}$ being the encapsulation and decapsulation oracles respectively. Consider a fixed key $k \in \mathcal{K}$, which has a public component k_{pub} and private component k_{pri}.

Definition 2.4.6. IND-CPA Let $\mathcal{A}$ be a probabilistic polynomial time adversary with access to $\mathcal{O}_k^{\overline{\mathcal{E}}}$. Let $X = (C, K_0) \sim \overline{\mathcal{E}}(k_{pub})$ and $Y = (C, K_1)$ where K_1 is uniformly and randomly sampled from $\mathcal{P}$. Then $Adv^{\mathcal{A}}(X, Y) \in O(2^{-\lambda})$.

Definition 2.4.7. IND-CCA Let $\mathcal{A}$ be a probabilistic polynomial time adversary with access to $\mathcal{O}_k^{\overline{\mathcal{D}}}$. Let $X = (C, K_0) \sim \overline{\mathcal{E}}(k_{pub})$ and $Y = (C, K_1)$ where K_1 is uniformly and randomly sampled from $\mathcal{P}$. Then $Adv^{\mathcal{A}}(X, Y) \in O(2^{-\lambda})$, provided the adversary is not allowed to query the oracle with C.

2.5 Miscellaneous

Definition 2.5.1. Projection Let A be a set, I be an indexing set and $B = A^{|I|}$. Then for all $i \in I$ let π_i denote the projection onto the ith coordinate. That is, for all $b = \{a_j\}_{j \in I} \in B$, we have $\pi_i(b) = a_i$.

Definition 2.5.2. Randomness Extractor Let A and B be sets. A randomness extractor (or extractor) is a function $Ext : A \to B$ such that $Ext(a)$ is indistinguishable from a uniformly and randomly chosen $b \in B$ for an adversary that does not know $a \in A$.

Chapter 3

Message Extraction

3.1 Overview

In Section 1.3, we introduced MersennePKC but did not provide an algorithm for Bob to obtain the plaintext $(A, B) \in \mathcal{H}_h^n \times \mathcal{H}_h^n$ given $D = AF + BG$ obtained from multiplying his private key G with the ciphertext $C = A\frac{F}{G}$ sent by Alice. In this chapter, we deduce an algorithm to obtain (A, B) from D by way of an experiment.

We make and experimentally confirm two hypotheses:

- First Iteration Hypothesis
- Second Iteration Hypothesis

We then use the two hypotheses to deduce an algorithm for computing (A, B) from D.

3.2 Experimental Strategy

For the entire experiment, we fix cryptosystem parameters $n = 86243$ (a Mersenne prime) and $h = 128$. These parameters were chosen because we desire to have $h = 128$ bits of security, and $n = 86243$ is the smallest Mersenne exponent for which we obtain viable experimental results (for this value of h) in Section 3.3.

3.2.1 Setup

The experiment is to be repeated some $N \in \mathbb{N}$ times.

For each repetition ι of the experiment, simulate Alice generating a random plaintext $(A, B) \in \mathcal{H}_h^n \times \mathcal{H}_h^n$ by sampling:

- $a_1, \dots, a_h$ from $\mathcal{U}(\mathbb{Z}_n)$ without replacement, where $\mu(A) = \sum_{k=1}^{h} 2^{a_k}$
- $b_1, \dots, b_h$ from $\mathcal{U}(\mathbb{Z}_n)$ without replacement, where $\mu(B) = \sum_{k=1}^{h} 2^{b_k}$

We also generate Bob's private key $(F, G) \in \mathcal{H}_h^n \times \mathcal{H}_h^n$ in the same way, that is by sampling:

- $f_1, \dots, f_h$ from $\mathcal{U}(\mathbb{Z}_n)$ without replacement, where $\mu(F) = \sum_{k=1}^{h} 2^{a_k}$
- $g_1, \dots, g_h$ from $\mathcal{U}(\mathbb{Z}_n)$ without replacement, where $\mu(B) = \sum_{k=1}^{h} 2^{b_k}$

Let $\alpha_\iota = \{a_1, \dots, a_h\}$. Let $D = AF + BG$.

For any subset $U \in \alpha_\iota$ we then define the following functions $\phi_\iota, \phi_{\iota,U} : \mathbb{Z}_n \to \mathbb{Z}$ in the following way:

$$\begin{aligned} \phi_\iota(i) &= Ham(D) - Ham(D - 2^i F) \\ \phi_{\iota,U}(i) &= Ham(D - \sum_{k \in U} 2^k F) - Ham(D - \sum_{k \in U} 2^k F - 2^i F) \end{aligned}$$

3.2.2 First Iteration: Measurements and Hypothesis

For each repetition $\iota \in \{1, \ldots, N\}$, we compute

$$\mathcal{I}_\iota = \{(i, \phi_\iota(i) : i \in \alpha_\iota\}$$
$$\mathcal{J}_\iota = \{(i, \phi_\iota(i) : i \notin \alpha_\iota\}$$

We then compute the set of all "wanted" points $\mathcal{I}$ and the corresponding frequency map (number of occurrences of each Hamming weight) $\mathcal{F}_\mathcal{I}$:

$$\mathcal{I} = \bigcup_{\iota=1}^{N} \mathcal{I}_\iota$$
$$\mathcal{F}_\mathcal{I} = \{(m, \Delta) : m = |\{(i, d) \in \mathcal{I} : d = \Delta\}|\}$$

Additionally, we compute the set of points which are "unwanted", but result in the largest Hamming weight decreases (among all the unwanted points in the same repetition) when chosen. This set $\mathcal{J}_*$, together with corresponding frequency map $\mathcal{F}_{\mathcal{J}_*}$ is given by:

$$\mathcal{J}_* = \bigcup_{\iota=1}^{N} \{(i, d) : d = \max \pi_2(\mathcal{J}_\iota)\}$$
$$\mathcal{F}_{\mathcal{J}_*} = \{(m, \Delta) : m = |\{(i, d) \in \mathcal{J}_* : d = \Delta\}|\}$$

We then plot two histograms: $H_\mathcal{I}$ for $\mathcal{F}_\mathcal{I}$ and $H_{\mathcal{J}_*}$ for $\mathcal{F}_{\mathcal{J}_*}$. We fit the normal distributions $N(\mu_\mathcal{I}, \sigma^2_\mathcal{I})$ and $N(\mu_{\mathcal{J}_*}, \sigma^2_{\mathcal{J}_*})$ onto $H_\mathcal{I}$ and $H_{\mathcal{J}_*}$ respectively.

First Iteration Hypothesis Let $X \sim N(\mu_{\mathcal{J}_*}, \sigma^2_{\mathcal{J}_*})$. We hypothesise that the respective means and variances take on values such that $Pr[X \geq \mu_\mathcal{I}] \approx 10^{-29}$.

For each repetition ι, we choose a set of points U_ι by skimming off all the indices that yield Hamming weight reductions more than $\mu_{\mathcal{I}}$:

$$U_\iota = \{i \in \mathbb{Z}_n : \phi_\iota(i) > \mu_{\mathcal{I}}\}$$

Assuming the first iteration hypothesis, $U_\iota \subseteq \alpha_\iota$ with high probability, for each repetition ι.

3.2.3 Second Iteration: Measurements and Hypothesis

We then compute:

$$\mathcal{I}'_\iota = \{(i, \phi_{\iota,U}(i) : i \in \alpha_\iota \backslash U_\iota\}$$
$$\mathcal{J}'_\iota = \{(i, \phi_{\iota,U}(i) : i \notin \alpha_\iota \backslash U_\iota\}$$

We repeat all the calculations as in the first iteration, replacing $\mathcal{I}_\iota$ and $\mathcal{J}_\iota$ by $\mathcal{I}'_\iota$ and $\mathcal{J}'_\iota$ respectively:

$$\begin{aligned} \mathcal{I}' &= \bigcup_{\iota=1}^{N} \mathcal{I}'_\iota \\ \mathcal{F}_{\mathcal{I}'} &= \{(m, \Delta) : m = |\{(i,d) \in \mathcal{I}' : d = \Delta\}|\} \\ \mathcal{J}'_* &= \bigcup_{\iota=1}^{N} \{(i,d) : d = \max \pi_2(\mathcal{J}'_\iota)\} \\ \mathcal{F}_{\mathcal{J}'_*} &= \{(m, \Delta) : m = |\{(i,d) \in \mathcal{J}'_* : d = \Delta\}|\} \end{aligned}$$

We then plot two new histograms: $H_{\mathcal{I}'}$ for $\mathcal{F}_{\mathcal{I}'}$ and $H_{\mathcal{J}'_*}$ for $\mathcal{F}_{\mathcal{J}'_*}$. We fit the normal distributions $N(\mu_{\mathcal{I}'}, \sigma^2_{\mathcal{I}'})$ and $N(\mu_{\mathcal{J}'_*}, \sigma^2_{\mathcal{J}'_*})$ onto $H_{\mathcal{I}'}$ and $H_{\mathcal{J}'_*}$ respectively.

Second Iteration Hypothesis Let $Y \sim N(\mu_{\mathcal{J}'_*}, \sigma^2_{\mathcal{J}'_*})$. We hypothesise that the respective means and variances take on values such that $Pr[Y \geq \mu_{\mathcal{I}'} - 4\sigma_{\mathcal{I}'}] \approx 10^{-28}$.

For each repetition ι, we choose a set of points U'_ι by skimming off all the indices that yield Hamming weight reductions more than $\mu_{\mathcal{I}'} - 4\sigma_{\mathcal{I}'}$:

$$U'_\iota = \{i \in \mathbb{Z}_n : \phi_{\iota,U_\iota}(i) > \mu_{\mathcal{I}'} - 4\sigma_{\mathcal{I}'}\}$$

Assuming the second iteration hypothesis, $U'_\iota \subseteq \alpha_\iota$ with high probability, for each repetition ι.

3.3 Experimental Results

3.3.1 First Iteration

We plot the histograms $H_{\mathcal{I}}$ (green portion) and $H_{\mathcal{J}_*}$ (red portion) for the first iteration. The data for $H_{\mathcal{I}}$ has a sample mean of 74.5 and sample variance of 8.64^2. The data for $H_{\mathcal{J}_*}$ has a sample mean of 25.9 and sample variance of 4.17^2.

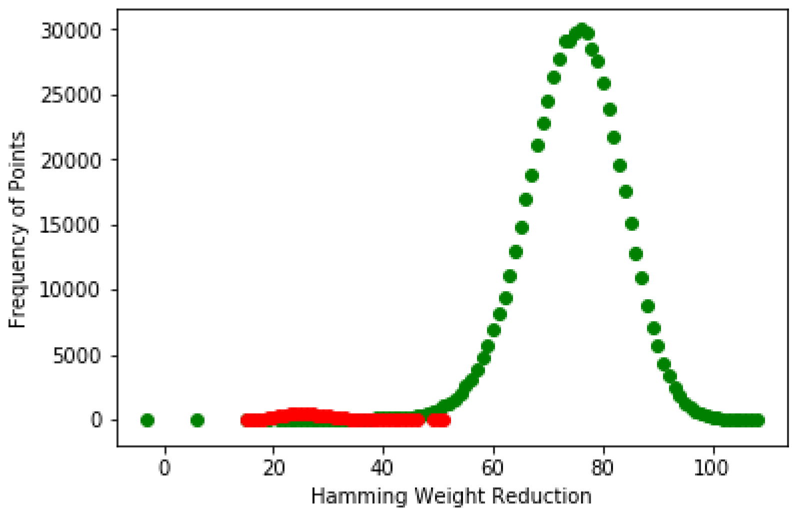
We fit normal distributions to both histograms by method of moments estimation and obtain:

$$\begin{aligned} \mu_{\mathcal{I}} &= 74.5 \\ \sigma_{\mathcal{I}} &= 8.64 \\ \mu_{\mathcal{J}_*} &= 25.9 \\ \sigma_{\mathcal{J}_*} &= 4.17 \end{aligned}$$

Letting $X \sim N(\mu_{\mathcal{J}_*}, \sigma^2_{\mathcal{J}_*})$ we use the Gaussian tail bound to obtain:

$$\begin{aligned}
Pr[X \geq \mu_{\mathcal{I}}] &= Pr[X - \mu_{\mathcal{J}_*} \geq \mu_{\mathcal{I}} - \mu_{\mathcal{J}_*}] \\
&= Pr[X - \mu_{\mathcal{J}_*} \geq 74.5 - 25.9] \\
&= Pr[X - \mu_{\mathcal{J}_*} \geq 48.6] \\
&\leq \exp\left(-\frac{48.6^2}{2\sigma^2_{\mathcal{J}_*}}\right) \\
&= \exp\left(-\frac{48.6^2}{2 \times 4.17^2}\right) \\
&\leq 3.20 \times 10^{-30}
\end{aligned}$$

This confirms the first iteration hypothesis.



3.3.2 Second Iteration

We plot the histograms $H_{\mathcal{I}'}$ (green portion) and $H_{\mathcal{J}'_*}$ (red portion) for the first iteration. The data for $H_{\mathcal{I}'}$ has a sample mean of 87.5 and sample variance of 6.65^2. The data for $H_{\mathcal{J}'_*}$ has a sample mean of 11.9 and sample variance of 4.35^2.

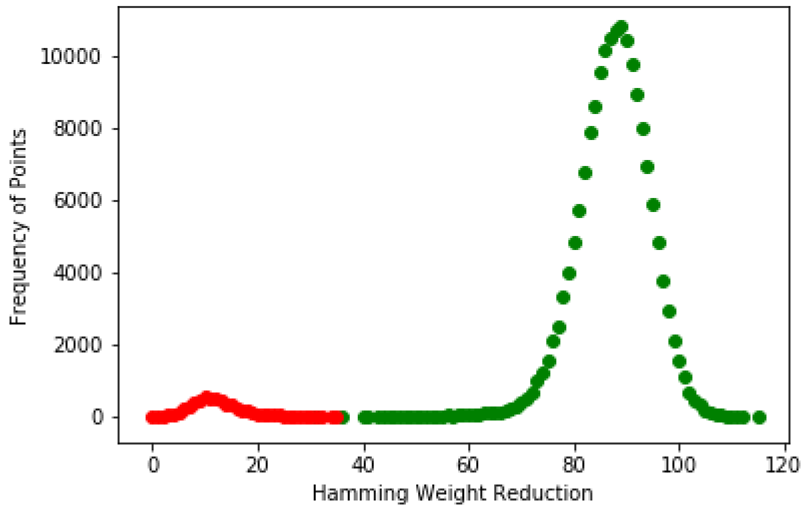
We fit normal distributions to both histograms by method of moments estimation and obtain:

$$\begin{aligned}
\mu_{\mathcal{I}'} &= 87.5 \\
\sigma_{\mathcal{I}'} &= 6.65 \\
\mu_{\mathcal{J}'_*} &= 11.9 \\
\sigma_{\mathcal{J}'_*} &= 4.35
\end{aligned}$$

Letting $Y \sim N(\mu_{\mathcal{J}'_*}, \sigma^2_{\mathcal{J}'_*})$ we use the Gaussian tail bound to obtain:

$$\begin{aligned}
Pr[Y \geq \mu_{\mathcal{I}'} - 4\sigma_{\mathcal{I}'}] &= Pr[Y - \mu_{\mathcal{J}'_*} \geq \mu_{\mathcal{I}'} - 4\sigma_{\mathcal{I}'} - \mu_{\mathcal{J}'_*}] \\
&= Pr[Y - \mu_{\mathcal{J}'_*} \geq 87.5 - 4 \times 6.65 - 11.9] \\
&= Pr[Y - \mu_{\mathcal{J}'_*} \geq 49.0] \\
&\leq \exp\left(-\frac{49.0^2}{2\sigma^2_{\mathcal{J}'_*}}\right) \\
&= \exp\left(-\frac{49.0^2}{2 \times 4.35^2}\right) \\
&\leq 2.80 \times 10^{-28}
\end{aligned}$$

This confirms the second iteration hypothesis.



3.4 Deducing the Algorithm

At this point, we have experimentally verified the two hypotheses we made earlier. We can now use these hypotheses to obtain an algorithm $\chi_{n,h}$ to extract (A, B) given D:

$$\chi_{n,h} : \mathcal{B}_n \times \mathcal{H}_h^n \times \mathcal{H}_h^n \to \mathcal{H}_h^n \times \mathcal{H}_h^n$$
$$\chi_{n,h} = (AF + BG, F, G) \mapsto (A, B)$$

3.4.1 Proposed Algorithm

Suppose it is known that $A, B, F, G \in \mathcal{H}_h^n$, where F and G is known to Bob while A and B is not. Suppose Bob has $D = AF + BG \in \mathcal{B}_n$ and wishes to compute $(A, B) \in \mathcal{H}_h^n \times \mathcal{H}_h^n$. We propose an procedure EXTRACT-MSG to do so:

```
1 EXTRACT-MSG(n, h, D, F, G):
2     A := 0
3     B := 0
```

```

    for k in {0, 1}
        maintain a loop counter $k$

        for i: $\mathbb{Z}_n$ in {0, .., n-1}:
            if HAMMING-WEIGHT-DECREASE(D, F, i) >= $\delta_k$:
                set bit i of A
            if HAMMING-WEIGHT-DECREASE(D, G, i) >= $\delta_k$:
                set bit i of B

        D := D - (A * F + B * G)

    if Ham(A) != h or Ham(B) != h:
        error

    return (A, B)
```

The procedure EXTRACT-MSG depends on a threshold δ_k for $k \in \{0, 1\}$. From the experimental results used to confirm the iteration hypotheses, we set:

$$\begin{aligned}\delta_0 &= \mu_{\mathcal{I}} \\ &= 74.5 \\ \delta_1 &= \mu_{\mathcal{I}'} - 4\sigma_{\mathcal{I}'} \\ &= 87.5 - 4 \times 6.65 \\ &= 60.9\end{aligned}$$

The procedure EXTRACT-MSG also depends on a subprocedure:

```
HAMMING-WEIGHT-DECREASE(D, X, i):
    return HAM(D) - HAM(D - X * 2^i)
```

3.4.2 Probabilistic Correctness of Algorithm

The algorithm has two main iterations, with the main body of each iteration taking place in lines 6-14. We analyse what happens in those two iterations, as well as the error check that takes place in lines 16-17.

Let α and β be the (unknown) sets, both of size h, such that:

$$\begin{aligned}A &= \sum_{i \in \alpha} 2^i \\ B &= \sum_{j \in \beta} 2^j\end{aligned}$$

To avoid unwieldy prose in the paragraphs that follow, we define the Hamming weight decrease function (the same one as in HAMMING-WEIGHT-DECREASED) by:

$$\begin{aligned}&\phi : \mathcal{B}_n \times \mathcal{H}_h^n \times \mathbb{Z}_n \to \mathbb{Z} \\ &\phi = (D, X, i) \mapsto Ham(D) - Ham(D - X * 2^i)\end{aligned}$$

Probability of Wrong Index Being Chosen We first analyse the probability that the algorithm chooses a wrong index to set in either A or B. This wrong choice can happen in either of the two iterations of the loop in Line 5.

Let $\alpha_0, \beta_0 \subseteq \mathbb{Z}_n$ be the indices chosen and set in the first iteration for A and B respectively. Let $E_{A,0}$ and $E_{B,0}$ be the events that $\alpha_0 \subseteq \alpha$ and $\beta_0 \subseteq \beta$ respectively. By the first iteration hypothesis, we have:

$$Pr[\overline{E_{A,0}}] \approx 10^{-29}$$
$$Pr[\overline{E_{B,0}}] \approx 10^{-29}$$

Let α_1, β_1 be the indices chosen and set in the second iteration for A and B respectively. Let $E_{A,1}$ and $E_{B,1}$ be the events that $\alpha_1 \subseteq \alpha \backslash \alpha_0$ and $\beta_1 \subseteq \beta \backslash \beta_0$ respectively. By the second iteration hypothesis, we have:

$$Pr[\overline{E_{A,1}} | \overline{E_{A,0}}] \approx 10^{-28}$$
$$Pr[\overline{E_{B,1}} | \overline{E_{A,0}}] \approx 10^{-28}$$

Let E_A and $_B$ be the events that a wrong index is chosen for A and B respectively. We have:

$$\begin{aligned} Pr[E_A] &= Pr[E_{A,0}] + Pr[\overline{E_{A,0}}] \times Pr[E_{A,1} | \overline{E_{A,0}}] \\ &\approx 10^{-29} + (1 - 10^{-29}) \times 10^{-28} \\ &\approx 10^{-28} \\ Pr[E_B] &= Pr[E_{B,0}] + Pr[\overline{E_{B,0}}] \times Pr[E_{B,1} | \overline{E_{B,0}}] \\ &\approx 10^{-28} \text{ (by similar reasoning)} \end{aligned}$$

Let E be the event that the algorithm chooses the wrong index at any time. Then $Pr[E] \leq Pr[E_A] + Pr[E_B] < 2 \times 10^{-28}$. We have therefore established that the algorithm picks the wrong index with extremely low probability. $\square$

Probability of Insufficient Number of Indices being Chosen Assuming that no incorrect point has been selected, we note that the error condition in line 16 triggers when $\alpha_0 \cup \alpha_1 \neq \alpha$, that is, when there are indices that remain unselected.

Let E' be the probability that at least one point that should have been selected but was not. Let $Z \sim N(\mu_{\mathcal{I}'}, \sigma^2_{\mathcal{I}'})$. Then we have:

$$\begin{aligned}
Pr[E'|\overline{E}] &= Pr[Z < \mu_{\mathcal{I}'} - 4\sigma_{\mathcal{I}'}] \\
&= Pr[\frac{Z - \mu_{\mathcal{I}'}}{\sigma_{\mathcal{I}'}} < -4] \\
&= \int_{-\infty}^{-4} \frac{1}{\sqrt{2\pi}} \exp{(-\frac{x^2}{4})} dx \\
&< 1.34 \times 10^{-4}
\end{aligned}$$

We therefore have that the probability of an insufficient number of points is somewhat high (order of 10^{-4}) compared to the probability of error. This, however, is not a cause for concern since we can simply repeat the loop in line 5 as many times as needed in the rare that an insufficient number of points are selected. □

We have seen that the message extraction algorithm presented in this section computes the correct (A, B) from D with very high probability (with a small, non-negligible but acceptable chance of needing to run for more than two iterations). We now turn to the runtime of the algorithm.

3.4.3 Runtime Of Algorithm

In Subsection 3.4.1, we gave an implementation of EXTRACT-MSG and HAMMING-WEIGHT-DECREASE. We first give the time complexity of EXTRACT-MSG in terms of the time complexity of HAMMING-WEIGHT-DECREASE, which we call $T(n)$.

Lemma 3.4.1. EXTRACT-MSG runs in time $O(nT(n) + nh)$.

Proof We see that lines 8-14 runs in $O(nT(n))$, since setting bits is $O(1)$ and the loop body (which takes $O(T(n)+1) = O(T(n))$ is run n times in the loop defined on line 8. Line 14 has multiplications $A * F$ and $B * G$ that take $O(nh)$ since F and G have Hamming weight h and we can simply perform $A * F = \sum_{k=1}^{h} A \times 2^{f_k}$ and $B * G = \sum_{k=1}^{h} B \times 2^{g_k}$ where $f_1, \ldots f_h$ are the bits set in F and $g_1, \ldots, g_h$ are the bits set in G.

The body of the loop in line 5 therefore runs in $O(nT(n)+nh)$, so the entire loop (which makes only a constant number of iterations) runs in $O(nT(n) + nh)$. The check on lines 16-17 run in $O(n)$ since they perform a Hamming weight computation, while the initialisation code on lines 2-3 take $O(n)$ as they allocate memory for bitstrings of length n.

We can therefore conclude that EXTRACT-MSG runs in $O(nT(n) + nh)$.
□

We note that, from our current implementation, HAMMING-WEIGHT-DECREASE runs in time $O(n)$. This is because all the steps (Hamming weight computation, subtraction, bit shifting) are all linear in the number of bits operated on, which is n. This makes the time complexity of EXTRACT-MSG $O(n^2 + nh) = O(n^2)$ since $n > 2h^2$. With $n = 86243$, the number of operations to be performed is on the order of 10^9, which is rather large. We therefore propose a functionally-equivalent but more efficient implementation of HAMMING-WEIGHT-DECREASE.

```
HAMMING-WEIGHT-DECREASE'(D, X, i):
    compute Y := X * $2^i$

    // ensure D > Y, this makes the result off by at most 1
```

```
pad D with an additional bit on the MSB side

//can view D as a bit sequence $d_r \dots d_1$
//can view Y as an ordered set of indices $\{y_{j_1}, \dots, y_{j_h}\}$

ham_wt_reduction := 0
stop := set bit in D with smallest index greater than $y_{j_h}$

for k from h downto 1:
    if $y_{j_k}$ = 0:
        if $d_{j_k}$ = 1:
            stop := $j_k$
        continue
    ham_wt_reduction -= 1
    if $d_{j_k}$ = 1:
        $d_{j_k}$ := 0
    else:
        c := stop - i
        ham_wt_reduction += c
        stop = i
return ham_wt_reduction
```

The idea behind HAMMING-WEIGHT-DECREASE' is to count the number of places where the Hamming weight could change during subtraction, rather than actually performing the subtraction.

- If both the minuend and subtrahend have a bit set at a given index, then the Hamming weight is guaranteed to decrease by exactly one at that index.
    - e.g. 110 - 010 causes a Hamming weight decrease of one at index

2

  - e.g. 11011 - 1000 causes a Hamming weight decrease of one at index 4

- If the subtrahend has a bit set at the index but the minuend does not, then the Hamming weight is guaranteed to increase by one less than the number of zeros in the minuend starting from that index and counting upward until the next set bit.

  - e.g $10000 - 1$ will cause a Hamming weight increase of $4 - 1 = 3$ since the result becomes 01111
  - e.g. $10000 - 10$ will cause a Hamming weight increase of $3 - 1 = 2$ since the result becomes 01110

The algorithm searches (the loop in Line 13) all the indices of bits set in the subtrahend (at most h many) and keeps track of the Hamming weight changes when each particular bit of the subtrahend is subtracted from the minuend. When the algorithm terminates, it will return the total Hamming weight change when the entire subtrahend has been subtracted from the minuend, which is what we want.

Claim 3.4.1. HAMMING-WEIGHT-DECREASE' runs in $O(h)$.

Proof. Line 2 runs in $O(h)$ since X has Hamming weight h. Lines 5 and 10 can be done in $O(1)$. Line 11 can be done in $O(1)$ on average by starting at d_{j_h} and searching upwards, since D is expected to be somewhat dense and no more than a very small number of positions are searched on average. Every step in Lines 14-24 takes $O(1)$, so the entire loop from Lines 13-24 takes $O(h)$. □

Therefore, using HAMMING-WEIGHT-DECREASE' in place of HAMMING-WEIGHT-DECREASE, EXTRACT-MSG has time complexity $O(nh)$. With

$n = 86243, h = 128$, this is of order 10^6, which is very fast on a modern computer.

Having obtained an efficient algorithm to compute (A, B) given D, we now turn to constructing the key-exchange protocol in the following chapter.

Chapter 4

Public Key Cryptosystem

4.1 Overview

Before we give a formal specification of MersennePKC, we first give an illustration of how it is expected to work intuitively.

Let $n, h \in \mathbb{N}$ such that $n > 2h^2$ and $2^n - 1$ is a Mersenne prime. Suppose we have an extractor $Ext : \mathcal{H}_h^n \times \mathcal{H}_h^n \to \mathbb{F}_2^\lambda$, for some $\lambda \in \mathbb{N}$.

We assume that we have two agents Alice and Bob. Suppose Bob has private key $(F, G) \in \mathcal{H}_h^n \times \mathcal{H}_h^n$ and public key $H = \frac{F}{G} \in \mathcal{B}_n$. Then an Alice can send a message $m \in \mathbb{F}_2^\lambda$ to Bob via the following protocol:

Alice		**Bob**
$A, B \sim \mathcal{U}(\mathcal{H}_h^n)$		
$C := AH + B$		
$c := Ext(A, B) \oplus m$		
send (C, c) to Bob	$\longrightarrow$	receive (C, c) from Alice
		$D := CG$
		$(A, B) := \chi_{n,h}(D, F, G)$
		$m := Ext(A, B) \oplus c$

In the protocol above, Alice first uniformly and randomly samples A and B from $\mathcal{H}_h^n$. She then sends the pair $(C, c) = (AH + B, Ext(A, B) \oplus m)$ over an insecure channel to Bob, where H is Bob's public key.

Bob receives (C, c) from Alice and computes $D = CG$ using the G component of his private key. He then uses the message extraction algorithm described in Chapter 3 to recover A and B from D. He finally computes $Ext(A, B) \oplus c = Ext(A, B) \oplus Ext(A, B) \oplus m = m$, thereby retrieving the plaintext.

Having illustrated the functioning of MersennePKC, we proceed to its formal specification.

4.2 Specification

Definition 4.2.1. MersennePKC Let $n, h \in \mathbb{N}$ be fixed parameters such that $n \geq 2h^2$ and $p = 2^n - 1$ is a Mersenne prime. Let $Ext : \mathcal{H}_h^n \times \mathcal{H}_h^n \to \mathbb{F}_2^\lambda$ be an extractor. MersennePKC-(n, h, Ext) is the tuple $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ where:

$$\begin{aligned}
&\mathcal{P} = \mathbb{F}_2^\lambda \\
&\mathcal{C} = \mathcal{B}_n \times \mathbb{F}_2^\lambda \\
&\mathcal{K} = \{(F, G, H) \in \mathcal{H}_h^n \times \mathcal{H}_h^n \times \mathcal{B}_n : H = \frac{F}{G}\} \\
&\mathcal{E} = \{e_k : \mathcal{H}_h^n \times \mathcal{H}_h^n \times \mathcal{P} \to \mathcal{C} : \\
&\qquad k = (F, G, H) \in \mathcal{K}, \\
&\qquad e_k(A, B, m) = (AH + B, Ext(A, B) \oplus m)\} \\
&\mathcal{D} = \{d_k : \mathcal{C} \to \mathcal{P} : \\
&\qquad k = (F, G, H) \in \mathcal{K}, \\
&\qquad d_k(C, c) = Ext(\chi_{n,h}(CG, F, G)) \oplus c\}
\end{aligned}$$

In the definition above, the cryptosystem explicitly depends on the choices of n, h and the extractor Ext, and also implicitly depends on λ via Ext.

- $\mathcal{P}$ is the set of plaintexts.
- $\mathcal{C}$ is the set of ciphertexts.
- $\mathcal{K}$ is the set of keys. Each key has the form (F, G, H), where (F, G) is the private key while H is the public key.
- $\mathcal{E}$ is the set of encryption functions. Each encryption function e_k depends on a key k, and is of the form $\mathcal{H}_h^n \times \mathcal{H}_h^n \times \mathcal{P} \to \mathcal{C}$. The extra input from $\mathcal{H}_h^n \times \mathcal{H}_h^n$ in addition to the plaintext is meant for the encryptor to supply uniform random choices to allow for semantic security.
- $\mathcal{D}$ is the set of decryption functions. Each decryption function d_k depends on a key k, and uses χ_{h_h} from Chapter 3 as part of its decryption.

4.3 Security

We can see from the illustration of MersennePKC that any adversary intercepting the insecure communications channel between Alice and Bob is able to obtain the following information:

- cryptosystem parameters n, h, known publicly
- Bob's public key, H, known publicly
- ciphertext (C, c) that Alice sends to Bob over the channel during the key exchange

We want to see how secure MersennePKC is under chosen plaintext attacks (IND-CPA) and chosen ciphertext attacks (IND-CCA).

For the rest of this section, we assume that $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ is a MersennePKC-(n, h, Ext) instance, where $Ext : \mathcal{H}_h^n \times \mathcal{H}_h^n \to \mathbb{F}_2^\lambda$ is an extractor, h is the security parameter and $\lambda = h$.

For each key $k = (F, G, H) \in \mathcal{K}$, we define two oracles.

- Let $\mathcal{O}_k^{\mathcal{E}}$ be an encryption oracle that encrypts a plaintext $m \in \mathcal{P}$ to $(AH + B, Ext(A, B) \oplus m) \in \mathcal{C}$, where A and B are uniformly and randomly chosen from $\mathcal{H}_h^n$.
- Let $\mathcal{O}_k^{\mathcal{D}}$ be a decryption oracle that decrypts a ciphertext $(C, c) \in \mathcal{C}$ to $Ext(A, B) \oplus c \in \mathcal{P}$, where $C = A\frac{F}{G} + B$.

4.3.1 Chosen Plaintext Attacks

MersennePKC is secure under chosen plaintext attacks (IND-CPA), subject to some assumptions.

Assumption 4.3.1. Let $\mathcal{A}$ be a probabilistic polynomial-time Turing machine. Let $(F, G, H) \in \mathcal{K}$. Define the following probability distributions:

$$\begin{aligned}\mathcal{X} &= \mathcal{U}(\{AH + B : A, B \in \mathcal{H}_h^n\})\\ \mathcal{Y} &= \mathcal{U}(\mathcal{B}_n)\end{aligned}$$

The task of $\mathcal{A}$ is to output 0 if its input comes from $\mathcal{X}$ and 1 if its input comes from $\mathcal{Y}$. Let $X \sim \mathcal{X}$ and $Y \sim \mathcal{Y}$ be random variables. Then $Adv^{\mathcal{A}}(X, Y) \in O(2^{-h})$. □

Assumption 4.3.2. Let $\mathcal{A}$ be a probabilistic polynomial-time Turing machine. Define the following probability distributions:

$$\begin{aligned}\mathcal{X} &= \mathcal{U}(\{Ext(A, B) : A, B \in \mathcal{H}_h^n\})\\ \mathcal{Y} &= \mathcal{U}(\mathcal{B}_n)\end{aligned}$$

The task of $\mathcal{A}$ is to output 0 if its input comes from $\mathcal{X}$ and 1 if its input comes from $\mathcal{Y}$. Let $X \sim \mathcal{X}$ and $Y \sim \mathcal{Y}$ be random variables. Then $Adv^{\mathcal{A}}(X, Y) \in O(2^{-h})$. □

Assumption 4.3.1 claims that the public keys appear uniformly random on $\mathcal{B}_n$ and do not leak information about the private key. Assumption 4.3.2 claims that $Ext(A, B)$ does not leak any information about $AH + B$ (and therefore does not leak any information about A or B).

With these assumptions, we can now show that MersennePKC is secure under chosen plaintext attacks.

Theorem 4.3.1. MersennePKC satisfies IND-CPA Fix some key $k = (F, G, H) \in \mathcal{K}$. Let $\mathcal{A}$ be a probabilistic polynomial-time adversary with access to H and $\mathcal{O}_k^{\mathcal{E}}$ who chooses $m_0, m_1 \in \mathcal{P}$. Define the following random variables:

$$Z_0 \sim \mathcal{U}(\{(AH + B, Ext(A, B) \oplus m_0) : A, B \in \mathcal{H}_h^n\})$$
$$Z_1 \sim \mathcal{U}(\{(AH + B, Ext(A, B) \oplus m_1) : A, B \in \mathcal{H}_h^n\})$$

Then $Adv^{\mathcal{A}}(Z_0, Z_1) \in O(2^{-h})$ regardless of the choice of $m_0, m_1 \in \mathcal{P}$ that $\mathcal{A}$ makes.

Proof. The goal is for $\mathcal{A}$ to output b given $(C, c) = (AH+B, Ext(A, B)\oplus m_b)$. There are two ways to do this: attempting to recover information about A and B (and hence $Ext(A, B)$ by obtaining information about the private key F and G, or using $AH + B$ to obtain information about $Ext(A, B)$.
By Assumption 4.3.1, the public key appears uniform and random to $\mathcal{A}$. Extracting any information about the private key from the public key allows $\mathcal{A}$ to distinguish the public key from a random bitstring sampled from $\mathcal{U}(\mathcal{B}_n)$, and so the advantage of $\mathcal{A}$ in doing so is bounded above by $O(2^{-h})$.
By Assumption 4.3.2, the advantage of $\mathcal{A}$ in obtaining any information about $Ext(A, B)$ from $AH + B$ is bounded above by $O(2^{-h})$.
Since $\mathcal{A}$ needs to only succeed at one of the two attempts to distinguish Z_0 from Z_1, the probability of distinguishing them is bounded above by the sum of the probabilities of distinguishing them using each approach. Therefore, for some $\alpha, \beta \in \mathbb{N}$:

$$\begin{aligned} Adv^{\mathcal{A}}(Z_0, Z_1) &\in O(2^{-h}) + O(2^{-h}) \\ &= O(2^{-h}) \end{aligned}$$

□

4.3.2 Chosen Ciphertext Attacks

MersennePKC, as given above, fails to satisfy even IND-CCA1 i.e. it is vulnerable to nonadaptive chosen ciphertext attacks.

Suppose an probabilistic polynomial-time adversary with access to $\mathcal{O}_k^{\mathcal{D}}$ chooses $m_0, m_1 \in \mathcal{P}$, is given $(C, c) = (AH + B, Ext(A, B) \oplus m_b) \in \mathcal{C}$ by the challenger and needs to determine $b \in \{0, 1\}$. The adversary simply queries $\mathcal{O}_k^{\mathcal{D}}$ with the ciphertext $(AH+B, 0^\lambda)$. The oracle gives $m' = Ext(A, B) \oplus 0^\lambda = Ext(A, B)$ to the adversary, who then computes $c \oplus m' = Ext(A, B) \oplus m_b \oplus Ext(A, B) = m_b$ and compares the result with m_0 and m_1. In this way, the adversary can determine b, so MersennePKC as given does not satisfy IND-CCA1.

To make MersennePKC secure under chosen ciphertext attacks, we use a technique similar to that in Section 7 of (Aggarwal et al., 2018): turning the MersennePKC-(n, h, Ext) instance into a key encapsulation mechanism and using a random oracle to transform the randomly selected key. We call this new key encapsulation mechanism MersenneKEM-(n, h, Ext).

Definition 4.3.1. MersenneKEM Suppose $\mathcal{H}_0$ and $\mathcal{H}_1$ are random oracles with input $\mathbb{F}_2^\lambda$ and output $\mathcal{H}_h^n$.

- Let $\overline{\mathcal{E}}$ be an algorithm that takes as input a public key H, samples $K \sim \mathcal{U}(\mathbb{F}_2^\lambda)$, computes $A = \mathcal{H}_0(K), B = \mathcal{H}_1(K)$ and outputs $(AH + B, Ext(A, B) \oplus K$.
- Let $\overline{\mathcal{D}}$ be an algorithm that as input a private key (F, G) and ciphertext (C, c), retrieves (A', B') from CG using the message extraction algorithm from Chapter 3, computes $K' = Ext(A', B') \oplus c$, and outputs K' if $A'H + B' = C$ or gives an error otherwise.

MersenneKEM-(n, h, Ext) is the tuple $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \overline{\mathcal{E}}, \overline{\mathcal{D}})$.

Theorem 4.3.2. MersenneKEM satisfies IND-CCA Fix some key $k = (F, G, H) \in \mathcal{K}$. Let $\mathcal{A}$ be a probabilistic polynomial-time adversary with access to H and an oracle $\mathcal{O}_k^{\overline{\mathcal{D}}}$ to perform decapsulation. Let $X = (C, K_0) \sim \overline{\mathcal{E}}(F, G)$ and $Y = (C, K_1)$ where K_1 is uniformly and randomly sampled from $\mathcal{P}$. Then $Adv^{\mathcal{A}}(X, Y) \in O(2^{-h})$.

The proof of Theorem 4.3.2 is identical to that of Theorem 2 in (Aggarwal et al., 2018), although the key exchange protocols are different.

4.4 Extractors

MersennePKC and MersenneKEM are defined with respect to an extractor function, in addition to n and h. Since we set the security parameter λ to h, we expect the extractors to have the type $Ext : \mathcal{H}_h^n \times \mathcal{H}_h^n \to \mathbb{F}_2^h$.

Since we set $h = 128$, we can generally use any cryptographic hash function that outputs a 128-bit string as an extractor. However, in the interest of efficiency in encryption and decryption, we opt to use a simpler extractor involving the inner product. Before we take an inner product, we perform a sequence of transformations to remove any unnecessary bits not contributing to the entropy of the source strings used in the extractor.

Suppose we have $A \in \mathcal{H}_h^n$, which is a bitstring that typically consists of clusters of 0s separated by a small number of 1s. Since A has Hamming weight h, there are at most $h+1$ clusters. Each cluster has at most n elements since the entire string A has only n bits. We can therefore identify the sequence of clusters with a vector in $\mathbb{Z}_n^{h+1}$. We define a map $f : \mathcal{H}_h^n \to \mathbb{Z}_n^{h+1}$ to represent this identification.

We then define the map $g : \mathbb{Z}_n^{h+1} \to \mathbb{F}_2^{(h+1)\log n}$ in the following manner: given an input $x \in \mathbb{Z}_n^{h+1}$, concatenate the bit representations of all the elements in the vector without the leading one (which does not contribute to the entropy of the result) into a single bitstring y. Since each element of x has at most $\log n$ bits (being an element of $\mathbb{Z}_n$) and x has $h+1$ elements, it follows that $y \in \mathbb{F}_2^{(h+1)\log n}$.

We finally define the map $h : \mathbb{F}_2^{(h+1)\log n} \to \mathbb{F}_{2^h}^m$, for the appropriate choice of m, by regarding each element of $\mathbb{F}_2^{(h+1)\log n}$ as an element in $\mathbb{F}_{2^h}^m$. This m will be sligtly greater than $\log n$, and the bitstrings in $\mathbb{F}_2^{(h+1)\log n}$ may be padded if necessary.

Definition 4.4.1. Inner Product Extractor Define $Ext : \mathcal{H}_h^n \times \mathcal{H}_h^n \to \mathbb{F}_{2^h}$ in the following way:

$$Ext(A, B) =< h \circ g \circ f(A), h \circ g \circ f(B) >$$

We then identify $\mathbb{F}_{2^h}$ with $\mathbb{F}_2^h$ by identifying elements in $\mathbb{F}_{2^h}$ with their bit representations, and obtain an extractor $Ext : \mathcal{H}_h^n \times \mathcal{H}_h^n \to \mathbb{F}_2^h$.

The inner product extractor appears uniform and random to an adversary without access to A or B: this is a standard result from (Chor & Goldreich, 1988). This extractor is not secure under information leakage; however, it may be transformed accordingly using methods from (Dziembowski & Faust, 2011) if needed.

We note that the extractor defined here runs in $O(n)$. This is clear since none of the intermediate representations require more than n bits (the most is $(h + 1)\log n$, which is still less than $n \geq 2h^2$), and all the intermediate steps require making only a single pass over the data.

4.5 Algorithmic Efficiency

For the key exchange protocol to be usable in practice, it is necessary that its operations are efficient. We consider the complexity of the following operations:

- key generation
- encryption

- decryption

4.5.1 Key Generation

Key generation involves Bob randomly generating $F, G \in \mathcal{H}_h^n$ and computing $\frac{F}{G}$.

Generating F (and G) involves sampling h numbers $i_1, \dots i_h$ without replacement from $\mathcal{U}(\mathbb{Z}_n)$ and computing $F = \sum_{k=1}^n 2^{i_k}$ (similarly for G). An algorithm is given in (Gupta & Bhattacharjee, 1984) that performs this with time complexity $O(h \log h) = O(\sqrt{n} \log n) \subseteq O(n)$.

Computing $\frac{F}{G}$ can then be done by obtaining $G^{-1} \mod 2^n - 1$. This can be done by using the extended Euclidean algorithm to compute $\gcd(G, 2^n - 1)$, which runs in time $O(\log 2^n - 1) = O(n)$. We then compute $F \times G^{-1}$ using an algorithm given by (Schönhage & Strassen, 1971), which has time complexity $O(n \log n \log \log n)$. The process of computing $\frac{F}{G}$ therefore takes $O(n \log n \log \log n)$.

Key generation can therefore be done in $O(n \log n \log \log n)$ time.

4.5.2 Encryption

The encryption stage involves Alice randomly generating $A, B \in \mathcal{H}_h^n$ and computing $AH + B$, where H is Bob's public key.

Generating A and G involves the same process of sampling without replacement from $\mathcal{H}_h^n$ used to compute F and G in the key generation stage, using the algorithm given in (Gupta & Bhattacharjee, 1984), so we already know that this takes $O(n)$.

Computing $AH + B$ involves a multiplication step and an addition step. We use the algorithm given by (Schönhage & Strassen, 1971) to compute AH in $O(n \log n \log \log n)$, and then add B by the usual bitwise addition algorithm that runs in $O(n)$.

Computing $Ext(A, B)$ and $Ext(A, B) \oplus m$ takes $O(n)$.

The entire encryption stage therefore runs in $O(n \log n \log \log n)$.

4.5.3 Decryption

The decryption stage involves Bob computing $D = CG$, and then using the algorithm we gave in Section 3.4 to retrieve (A, B).

Computing $D = CG$ involves a multiplication, so we know this step takes $O(n \log n \log \log n)$ using the algorithm given in (Schönhage & Strassen, 1971). Retrieving (A, B) from D takes $O(nh) = O(n\sqrt{n})$.

Computing $Ext(A, B)$ and $Ext(A, B) \oplus c$ takes $O(n)$.

The entire decryption step therefore takes $O(n\sqrt{n})$ time.

Chapter 5

Conclusion

5.1 Tasks Accomplished

In this report, we have presented MersennePKC, a modified version of the cryptosystem given in (Aggarwal et al., 2018). We have experimentally determined an algorithm for an essential part of MersennePKC and given justification for its correctness, efficiency and security.

We have also implemented MersennePKC: the implemented code is available at `https://github.com/thenaesh/mersenne-pkc/` and is written in Rust (compiler version 1.34.0, nightly build at the time of writing). The implementation provides a library for performing encryption and decryption with MersennePKC, together with utilities that were used to generate the graphs in Chapter 3 and a sample program to demonstrate a proof of concept encryption and decryption.

We tested the proof of concept on an Intel Core i7-4770MQ machine (with 4 physical cores and 8 virtual cores) running Linux Mint 19.1 and measured the time taken to run a full sequence of key generation, encryption and decryption. We observed that the entire sequence took approximately 0.8s, making the MersennePKC implementation feasible for real-world use.

5.2 Improvements and Future Work

There remains areas for improvement, however. Throughout this report, we have made several assumptions, such as Assumption 4.3.1 and Assumption 4.3.2, in order to complete the proofs. The message extraction algorithm in Chapter 3 also required an experimental investigation to determine. Ideally, we would like to prove these assumptions and prove the correctness of the message extraction algorithm in Chapter 3 if possible.

With regards to the implementation, though we have demonstrated that MersennePKC is fast enough for real-world use, it is not yet sufficiently hardened for real-world use and is vulnerable to various side-channel attacks. We hope to harden the implementation in the future so that it may actually be used for real-world cryptography.

5.3 Concluding Remarks

We have presented a post-quantum cryptosystem based on the hardness of distinguishing quotients of two numbers with low Hamming weight in a field of Mersenne prime order. There are other approaches to post-quantum cryptography, such as lattice-based cryptosystems based on the hardness of finding short vectors in a lattice (Regev, 2009).

All of these approaches to post-quantum cryptography are in their infancy, but research into post-quantum cryptosystems are a necessity due to the possibility of feasible quantum computers becoming available to adversaries in the near future.

References

Aggarwal, D., Joux, A., Prakash, A., & Santha, M. (2018). A new public-key cryptosystem via mersenne numbers. In *Annual international cryptology conference* (pp. 459–482).

Chor, B., & Goldreich, O. (1988). Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM Journal on Computing*, *17*(2), 230–261.

Dziembowski, S., & Faust, S. (2011). Leakage-resilient cryptography from the inner-product extractor. In *International conference on the theory and application of cryptology and information security* (pp. 702–721).

Gupta, P., & Bhattacharjee, G. P. (1984). An efficient algorithm for random sampling without replacement. In M. Joseph & R. Shyamasundar (Eds.), *Foundations of software technology and theoretical computer science* (pp. 435–442). Berlin, Heidelberg: Springer Berlin Heidelberg.

Nielsen, M. A., & Chuang, I. (2002). *Quantum computation and quantum information.* AAPT.

Regev, O. (2009). On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, *56*(6), 34.

Samorodnitsky, G. (1991). Probability tails of gaussian extrema. *Stochastic processes and their applications*, *38*(1), 55–84.

Schönhage, A., & Strassen, V. (1971). Schnelle multiplikation grosser zahlen. *Computing*, *7*(3-4), 281–292.

Shor, P. W. (1994). Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science* (pp. 124–134).

Sloane, N. J., et al. (2008). *The on-line encyclopedia of integer sequences.*