A Code-Based Key Agreement Scheme using QC-MDPC Codes

By

Peck Kai Ting

A0130651U

Supervisor:

Associate Professor Ma Siu Lun

A thesis submitted in partial fulfilment of the requirement for the degree of

Bachelor of Science with Honours in Applied Mathematics

Department of Mathematics

National University of Singapore

Academic Year 2018/2019

ACKNOWLEDGEMENTS

I would like to express my utmost appreciation to my project supervisor, Associate Professor Ma Siu Lun for his guidance during the entire period of my honours project. Throughout the year, he has been most understanding towards my strengths and weaknesses, patiently helping me to clarify any of my doubts regarding mathematical concepts, and given me much useful advice at each stage of the project. I have learnt a lot from him, and without his constant guidance, this thesis would not have been possible.

On my university journey, I have met many other professors who have inspired me greatly, with their desire to learn and seek knowledge humbly. Hence, I would like to express my sincere gratitude to them, for teaching me the importance of questioning, and for shaping my academic journey in NUS into a fulfilling one.

I would also like to take this opportunity to express my deepest thanks to my family, who is ever so supportive of my endeavours. Although they may not fully understand my area of study, they have always encouraged me to go for what I think is best, and silently supported me in subtle ways and the ways they knew best.

To my friends, I would like to show my appreciation for their never-ending support while I was writing this thesis. Firstly, I would like to thank Team Hufflepuff, for being the most amazing and understanding group of friends. You are among my best gifts from Edinburgh, and I appreciate all the advice and encouragement you have given me, be it in life or academics. I would also like to thank my secondary school clique, for always brightening even my toughest days, and Shane for helping to proofread my thesis. Also, thank you, Dash and Jia Yao, for being the best listeners and coder friends, and I hope that your final-year project will be a fulfilling one. Thank you, Jia Peng, for being there since junior college and all through

my four years in Applied Mathematics.

Last but not least, I would like to show my gratitude to all the other amazing people, who have made a positive difference in my university life.

CONTENTS

1. SUMMARY

In online communication, there is often a need to encrypt contents sent and received, so that they are only readable by the intended audience. This is usually accomplished by cryptosystems and protocols built upon them, which guide message encryption and decryption. Depending on the context, different schemes are used. In particular, key agreement schemes are used to establish a common key between two (or more) parties to encrypt and decrypt messages sent and received between them; the process of doing so is called key agreement.

Commonly-used key agreement schemes, such as the Diffie-Hellman scheme, are based on problems in number theory, such as the discrete logarithm problem. This renders them vulnerable to quantum algorithms such as Shor's Algorithm – if the advancement of quantum computers reaches the stage whereby it can harness quantum algorithms to break these schemes easily, they will become obsolete.

In view of such possibilities, there is a need to search for cryptosystems resistant to quantum algorithms, leading us to post-quantum cryptography. In this thesis, we look at one such scheme: the Code-Based Algorithm for Key Encapsulation (CAKE). The thesis covers CAKE and its code construction (Chapter 4), a mathematical analysis of its feasiblity in terms of parameter choices, and an empirical investigation and discussion on parameter choices in Chapter 6. The experiments aim to find suitable code weights for CAKE to achieve better security of the scheme, given the difficulties in achieving both usefulness and security of the codes, as explained in Section 4.2.4. Circulant matrices and their properties are introduced in Chapter 2, and decoding algorithms used in the experiments in Chapter 5.

Theorem 4.3.4 and Corollary 4.3.6 are new (they are not expressed explicitly in the literature known to us). The theorem is built on existing Lemmas 4.3.1 to 4.3.3. It provides a

method to check and generate possible matrix sizes for CAKE, based on constraints from the original paper [1]. Several computer programmes have been written for investigating (empirically) the different parameter choices and the security of the scheme. These include CAKE encoding and decoding algorithms, and an algorithm which attempts to break the code. Another programme has been written for generating possible matrix sizes, based on Theorem 4.3.4 and Corollary 4.3.6. These programmes are included in the CD accompanying the thesis submission.

2. BASICS OF CIRCULANT MATRICES

In this chapter, we state and prove some basic properties of circulant matrices, which will be used for defining the code-based key encapsulation scheme CAKE.

Definition 2.0.1 (Cyclic shift of a vector)**.** *Let* $\boldsymbol{a} = (a_0, a_1, \ldots, a_{r-1})$ *be a row vector. The* ***cyclic shift*** *of* $\boldsymbol{a}$ *is the vector* $\sigma(\boldsymbol{a}) = (a_{r-1}, a_0, a_1, \ldots, a_{r-2})$.

Definition 2.0.2 (Circulant Matrix)**.** *A square matrix* $\boldsymbol{A}$ *of order* r *is a* ***circulant matrix*** *if for* $i \in \{2, 3, \ldots, r\}$*, the* i*th row of* $\boldsymbol{A}$ *is the cyclic shift of the* $(i-1)$*th row of* $\boldsymbol{A}$.

Remark 2.0.3. *If* $(a_0, a_1, \ldots, a_{r-1})$ *is the first row of* $\boldsymbol{A}$*, then* $\boldsymbol{A}$ *is a circulant matrix if and only if* $\boldsymbol{A}_{ij} = a_{j-i \pmod r}$*, for* $i, j \in \{1, 2, \ldots, r\}$*, where* $\boldsymbol{A}_{ij}$ *is the (i,j)-entry of* $\boldsymbol{A}$.

Definition 2.0.4. *Let* $\boldsymbol{a} = (a_0, a_1, \ldots, a_{r-1})$ *be the first row of a circulant matrix* $\boldsymbol{A}$ *of order* r*. The* ***Hall polynomial*** *of* $\boldsymbol{A}$ *is defined to be the polynomial*

$$a(x) = a_0 + a_1 x + \cdots + a_{r-1} x^{r-1}.$$

Without loss of generality, we may sometimes identify the row vector $\mathbf{a}$ with the polynomial $a(x)$ in the later chapters of this thesis.

Example 2.0.5. *Let* $\boldsymbol{A} = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{pmatrix}$*. The first row of* $\boldsymbol{A}$ *is* $(1, 0, 1, 1)$*, and the Hall polynomial of* $\boldsymbol{A}$ *is* $a(x) = 1 + x^2 + x^3$.

Proposition 2.0.6. *Let* $\boldsymbol{A}$ *and* $\boldsymbol{B}$ *be circulant matrices of order* r *over a field* $\mathbb{F}$ *and* $c \in \mathbb{F}$*. Let* a *and* b *be the first rows of* $\boldsymbol{A}$ *and* $\boldsymbol{B}$ *respectively.*

1. $\boldsymbol{A} + \boldsymbol{B}$ *is a circulant matrix and the Hall polynomial of* $\boldsymbol{A} + \boldsymbol{B}$ *is* $a(x) + b(x)$.

2. *$c\boldsymbol{A}$ is a circulant matrix and the Hall polynomial of $c\boldsymbol{A}$ is $ca(x)$.*

3. *$\boldsymbol{AB}$ is a circulant matrix and the Hall polynomial of $\boldsymbol{AB}$ is $a(x)b(x) \pmod{x^r - 1}$.*

4. *$\boldsymbol{A}^T$ is a circulant matrix and the Hall polynomial of $\boldsymbol{A}^T$ is $a(x^{-1}) \pmod{x^r - 1}$.*

5. *If $\boldsymbol{A}$ is invertible, then $\boldsymbol{A}^{-1}$ is a circulant matrix. If $\boldsymbol{d}$ is the first row of $\boldsymbol{A}^{-1}$, then $a(x)d(x) \equiv 1 \pmod{x^r - 1}$.*

Proof. Note that since $\mathbf{A}$ and $\mathbf{B}$ are circulant matrices, $\mathbf{A}_{ij} = a_{j-i \pmod r}$ and $\mathbf{B}_{ij} = b_{j-i \pmod r}$, for $i, j \in \{1, 2, \ldots, r\}$. We follow the notation in Definition 2.0.4. The first row of $\mathbf{A}$ and $\mathbf{B}$ are $(a_0, a_1, \ldots, a_{r-1})$ and $(b_0, b_1, \ldots, b_{r-1})$ respectively. The respective Hall polynomials are $a(x) = a_0 + a_1 x + \cdots + a_{r-1}x^{r-1}$ and $b(x) = b_0 + b_1 x + \cdots + b_{r-1}x^{r-1}$.

1. Denote $\mathbf{c} = (c_0, c_1, \ldots, c_{r-1})$ as the first row of $\mathbf{A} + \mathbf{B}$, and let $c(x)$ be the corresponding Hall polynomial.

   For $k \in \{1, 2, \ldots, r\}$, $c_{k-1} = (\mathbf{A} + \mathbf{B})_{1k} = \mathbf{A}_{1k} + \mathbf{B}_{1k} = a_{k-1} + b_{k-1}$. Therefore,

   $$\begin{aligned}(\mathbf{A} + \mathbf{B})_{ij} &= \mathbf{A}_{ij} + \mathbf{B}_{ij} \\ &= a_{j-i \pmod r} + b_{j-i \pmod r} \\ &= c_{j-i \pmod r}\end{aligned}$$

   By Remark 2.0.3, $\mathbf{A} + \mathbf{B}$ is a circulant matrix.

   Furthermore,

   $$\begin{aligned}c(x) &= c_0 + c_1 x + \cdots + c_{r-1}x^{r-1} \\ &= (a_0 + b_0) + (a_1 + b_1)x + \cdots + (a_{r-1} + b_{r-1})x^{r-1} \\ &= (a_0 + a_1 x + \cdots + a_{r-1}x^{r-1}) + (b_0 + b_1 x + \cdots + b_{r-1}x^{r-1}) \\ &= a(x) + b(x)\end{aligned}$$

2. Denote $\mathbf{d} = (d_0, d_1, \ldots, d_{r-1})$ as the first row of $c\mathbf{A}$, and let $d(x)$ be the corresponding Hall polynomial.

For $k \in \{1, 2, \ldots, r\}$, $d_{k-1} = (c\mathbf{A})_{1k} = c\mathbf{A}_{1k} = ca_{k-1}$. Therefore,

$$\begin{aligned}(c\mathbf{A})_{ij} &= c\mathbf{A}_{ij} \\ &= ca_{j-i \pmod r} \\ &= d_{j-i \pmod r}\end{aligned}$$

By Remark 2.0.3, $c\mathbf{A}$ is a circulant matrix.

Furthermore,

$$\begin{aligned}d(x) &= d_0 + d_1 x + \cdots + d_{r-1}x^{r-1} \\ &= (ca_0) + (ca_1)x + \cdots + (ca_{r-1})x^{r-1} \\ &= c(a_0 + a_1 x + \cdots + a_{r-1}x^{r-1}) \\ &= ca(x)\end{aligned}$$

3. Denote $\mathbf{c} = (c_0, c_1, \ldots, c_{r-1})$ as the first row of $\mathbf{AB}$, and let $c(x)$ be the corresponding Hall polynomial.

For $p \in \{1, 2, \ldots, r\}$, $c_{p-1} = (\mathbf{AB})_{1p} = \sum_{k=1}^{r} \mathbf{A}_{1k}\mathbf{B}_{kp} = \sum_{k=1}^{r} a_{k-1}b_{p-k} = \sum_{m=0}^{r-1} a_m b_{p-m-1}$.

i.e. For $p \in \{0, 1, \ldots, r-1\}$, $c_p = \sum_{m=0}^{r-1} a_m b_{p-m}$. Therefore,

$$\begin{aligned}(\mathbf{AB})_{ij} &= \sum_{k=0}^{r-1} a_{k-i \pmod r} b_{j-k \pmod r} \\ &= \sum_{m=0}^{r-1} a_m b_{(j-i)-m \pmod r} \\ &= c_{j-i \pmod r}\end{aligned}$$

By Remark 2.0.3, $\mathbf{AB}$ is a circulant matrix.

Furthermore,

$$\begin{aligned}
c(x) &= c_0 + c_1x + \cdots + c_{r-1}x^{r-1} \\
&= (\sum_{m=0}^{r-1} a_m b_{-m}) + (\sum_{m=0}^{r-1} a_m b_{1-m})x + \cdots + (\sum_{m=0}^{r-1} a_m b_{r-1-m})x^{r-1} \\
&= \sum_{n=0}^{r-1}(\sum_{m=0}^{r-1} a_m b_{n-m})x^n \\
&\equiv \sum_{i=0}^{r-1}(\sum_{j=0}^{r-1} a_i b_j x^{i+j}) \pmod{x^r - 1} \\
&= (\sum_{i=0}^{r-1} a_i x^i)(\sum_{j=0}^{r-1} b_j x^j) \\
&= a(x)b(x)
\end{aligned}$$

4. Denote $\mathbf{c} = (c_0, c_1, \ldots, c_{r-1})$ as the first row of $\mathbf{A}^T$, and let $c(x)$ be the corresponding Hall polynomial.

   For $k \in \{1, 2, \ldots, r\}$, $c_{k-1} = (\mathbf{A}^T)_{1k} = \mathbf{A}_{k1} = a_{1-k \pmod r}$.

   i.e. For $k \in \{0, 1, \ldots, r-1\}$, $c_k = a_{-k \pmod r}$. Therefore,

$$\begin{aligned}
(\mathbf{A}^T)_{ij} &= \mathbf{A}_{ji} \\
&= a_{i-j \pmod r} \\
&= c_{j-i \pmod r}
\end{aligned}$$

   By Remark 2.0.3, $\mathbf{A}^T$ is a circulant matrix.

   Furthermore,

$$\begin{aligned}
c(x) &= c_0 + c_1x + \cdots + c_{r-1}x^{r-1} \\
&= a_0 + a_{r-1}x + \cdots + a_1x^{r-1} \\
&= a_0 + a_1x^{r-1} + \cdots + a_{r-1}x \\
&\equiv a(x^{-1}) \pmod{x^r - 1}
\end{aligned}$$

   since $a(x^{-1}) = a_0 + a_1x^{-1} + \cdots + a_{r-1}x^{-(r-1)} \equiv a_0 + a_1x^{r-1} + \cdots + a_{r-1}x \pmod{x^r - 1}$

5. The characteristic polynomial of $\mathbf{A}$ is defined as $p(\lambda) = det(\lambda\mathbf{I} - \mathbf{A})$. Let the character-

istic polynomial of $\mathbf{A}$ be $p(\lambda) = c_0 + c_1\lambda + c_2\lambda^2 + \cdots + c_n\lambda^n$.

By the Cayley-Hamilton Theorem, $p(\mathbf{A}) = c_0\mathbf{I} + c_1\mathbf{A} + c_2\mathbf{A}^2 + \cdots + c_n\mathbf{A}^n = \mathbf{0}$. (Note that $c_0 = (-1)^n det(\mathbf{A})$.)

Hence, if $\mathbf{A}$ is invertible, $\mathbf{A}^{-1} = -c_0^{-1}(c_1\mathbf{I} + c_2\mathbf{A} + \cdots + c_n\mathbf{A}^{n-1})$.

If $\mathbf{A}$ is a circulant matrix, by Results 1 to 3, $\mathbf{A}^{-1} = -c_0^{-1}(c_1 I + c_2\mathbf{A} + \cdots + c_n\mathbf{A}^{n-1})$ is also a circulant matrix.

Let $I(x)$ be the Hall polynomial for the identity matrix $\mathbf{I} = \mathbf{A}\mathbf{A}^{-1}$ and $d(x)$ be the Hall polynomial of $\mathbf{A}^{-1}$. By Result 3,

$$1 = I(x) \equiv a(x)d(x) \pmod{x^r - 1}.$$

□

3. INTRODUCTION TO KEY AGREEMENT

In this chapter, we introduce the basic concepts of key agreement (sometimes also referred to as key establishment) schemes (KAS). Key agreement is a method in cryptography by which two (or more) parties employ an active protocol to establish a session key, which is used to encrypt messages to be sent and decrypt received messages [2]. We first provide some background on public key and symmetric key cryptosystems.

3.1 Public Key and Symmetric Key Cryptosystems

3.1.1 Introduction to Public and Symmetric Key Cryptosystems

In cryptography, there are two main types of cryptosystems, namely public key cryptosystems and symmetric key cryptosystems.

Public key cryptosystems (also referred to as asymmetric key cryptosystems) are based on a pair of keys, called the private and public keys. In a typical public key cryptosystem, each user has their own pair of private and public keys. The public key is generally known to everyone, and is used for encryption of data. On the contrary, the corresponding private key is only known to the user, and is used for decryption. The guiding principle behind the cryptosystem is that the user can decrypt the encrypted message easily using the private key, but anyone who does not possess the private key will not be able to (or at least, will find it extremely difficult to) break the encryption and retrieve the original data. A simple (and ideal) scenario is as follows. Suppose Alice is a user in the cryptosystem with her own pair of private and public keys. Her private key is kept secret (only Alice knows her own private key), whereas her public key is made public. Since her public key is known to everyone, anyone possessing the public key would be able to encrypt data for transmission to Alice. On the other hand, as she is the only person in knowledge of her own private key, only she can decrypt

the encrypted data.

Such cryptosystems are usually based upon mathematical theories in number theory. Their security is therefore determined by the difficulty in solving the relevant mathematical problems. For example, the commonly-cited example of the Rivest-Shamir-Adleman (RSA) cryptosystem is based on the hardness of the (large) prime number factoring problem i.e. the problem of factoring a number $n = pq$, where p and q are large primes. The fastest solving algorithm for this problem at present has a sub-exponential time complexity, with respect to the number of bits of n. Since anyone not in possession of the private key will have to first solve this problem to break the encryption, the high time complexity translates to extreme difficulty in breaking the encryption, in terms of the amount of time required. The current recommended key size for RSA is 2048 bits, which would take over 6.4 quadrillion years to break on standard desktop computing power [3].

Symmetric key cryptosystems use the same key for encryption and decryption. Since this is the case, a secure communication channel is needed when establishing this common key, as any interceptor who manages to obtain the key would be able to decrypt communication between the originally-intended users. An example of a symmetric key cryptosystem is the Advanced Encryption Standard (AES).

3.1.2 Comparison between Public and Symmetric Key Cryptosystems

Advantages and Disadvantages of Public Key Cryptosystems

Public key cryptosystems hold an advantage over symmetric key cryptosystems, as there is no need to exchange a secret key before further communication can take place. This means that public key cryptosystems do not require a secure communication channel to exchange a secret key, unlike symmetric key cryptosystems.

However, public key cryptosystems also have their disadvantages. Firstly, they tend to be slower than symmetric key cryptosystems, as their encryption and decryption processes are usually significantly more computationally intensive [4]. Secondly, they are less secure than

symmetric key cryptosystems. In order to achieve the same level of security as symmetric keys, a longer key length is required for public keys, which also contributes to the higher computational complexity of public key cryptosystems. For instance, in terms of security strength, the RSA equivalent of 128-bit AES at present is 3072-bit RSA [5].

Advantages and Disadvantages of Symmetric Key Cryptosystems

Symmetric key cryptosystems are typically faster in terms of computational complexity and more secure than public key cryptosystems, as described above.

However, a disadvantage of symmetric key cryptosystems is that the communicating parties will have to first agree on a secret key before further communication can ensue.

Taking into account the various advantages and disadvantages of the two types of cryptosystems, when two parties, say Alice and Bob, wish to communicate securely with each other, they will first establish a common secret key K via a key agreement protocol. From then on, they will be able to communicate securely and efficiently using a symmetric key cryptosystem based on K. As the focus of this thesis is not on key agreement protocols, but rather the mathematical theories behind a particular type of key agreement scheme, we will not elaborate on the intricacies of the protocols.

In the next section, we will use the well-known Diffie-Hellman Key Agreement Scheme (KAS) to illustrate a basic form of key agreement. The Diffie-Hellman KAS uses an asymmetric method of establishing a symmetric key.

3.2 Diffie-Hellman Key Agreement Scheme

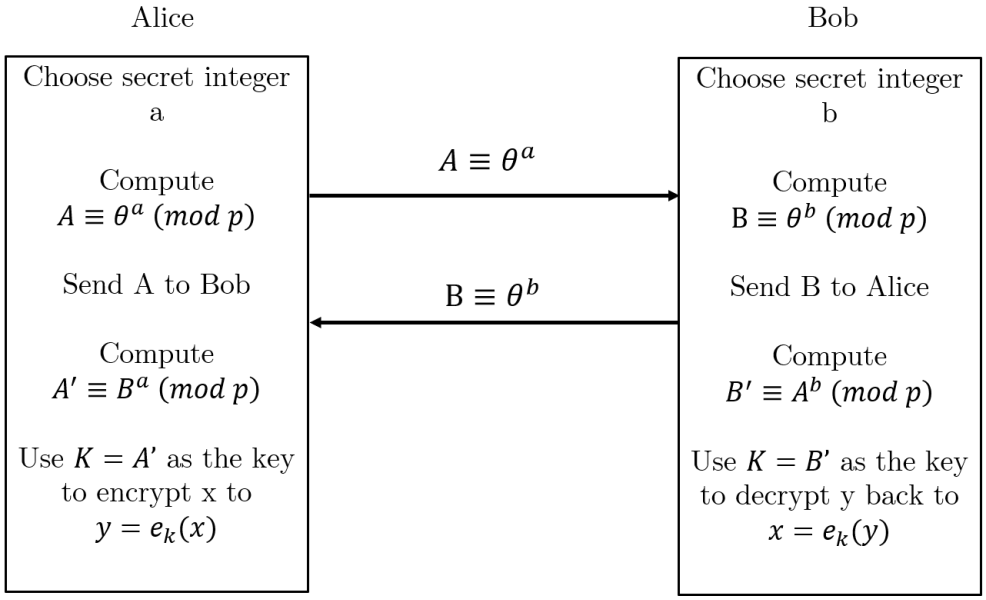
Definition 3.2.1 (Discrete Logarithm Problem)**.** *Given a large prime p and two integers c and d, find an integer x such that $c^x \equiv d \pmod{p}$*

The Diffie-Hellman Key Agreement Scheme is based upon the Discrete Logarithm Problem. In the scheme, the relevant parties Alice and Bob agrees on a large prime p and integer $\theta \not\equiv 0 \pmod{p}$. This information is made public. Each of them also chooses a secret integer

a and b respectively. Alice computes $A = \theta^a$ and sends it to Bob, and Bob sends $B = \theta^b$ to Alice. With A and B, Alice and Bob would each be able to compute the shared secret key $K = \theta^{ab} = \theta^{ba} = A^b = B^a$. So long as the Discrete Logarithm Problem is sufficiently difficult to solve, an interceptor, say Oscar, would not be able to compute the key K, even if he has knowledge of the values A and B.

The scheme is illustrated in Figure 3.1.

Fig. 3.1: Diffie-Hellman Key Agreement Scheme

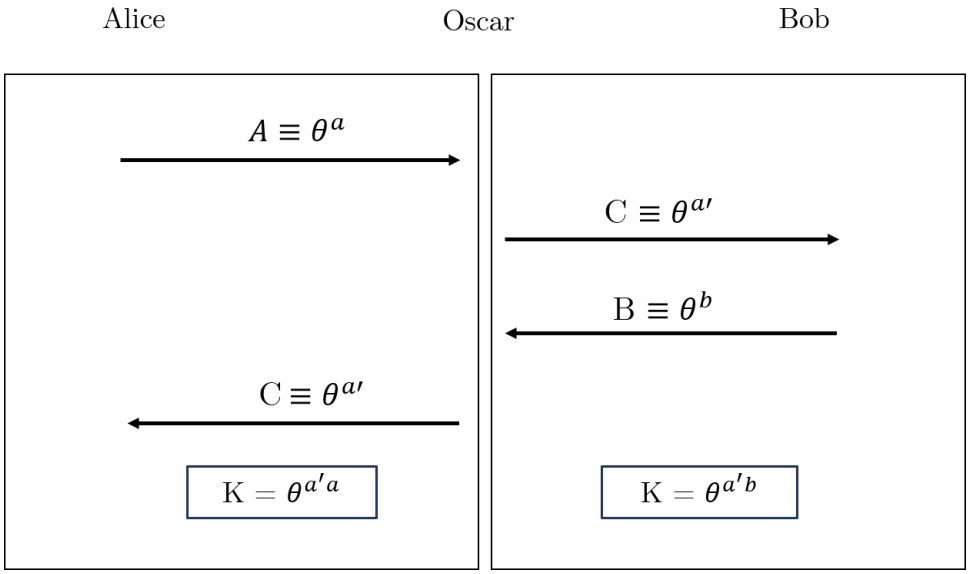


The Diffie-Hellman Key Agreement Scheme is a basic scheme, and is by itself susceptible to many types of attacks, such as the Man-in-the-middle (MITM) attack. One form of MITM attack occurs whereby Oscar attempts to establish a key with both Alice and Bob, by intercepting communication from Alice and Bob and sending its own messages to Alice and Bob instead. The detailed attack is illustrated in Figure 3.2. However, note that the Diffie-Hellman KAS only serves as a basic structure upon which more sophisticated protocols are built. Such protocols would be able to fend off more attacks, including the MITM attack described. For instance, the Transport Layer Security (TLS) handshake incorporates the Diffie-Hellman and RSA schemes in its protocol.

Another point worthy of note is that schemes such as the Diffie-Hellman scheme and RSA scheme is based on a problem related to number theory, which makes it vulnerable to quantum

algorithms. In particular, Shor's Algorithm, which is a quantum algorithm, would only take polynomial time to solve an integer factorization problem [6]. Hence, should there be advances in quantum computing to the point that it becomes viable to use quantum algorithms to solve these problems, such schemes would be rendered obsolete. In view of this possibility, there is a need to develop new key agreement schemes that are secure against quantum algorithms. Hence, research for the use of post-quantum cryptography in key agreement schemes is ongoing. In this thesis, we will study one of these methods, which is based on code-based cryptography, a particular area of post-quantum cryptography.

Fig. 3.2: An example of a Man-in-the-middle attack on the Diffie-Hellman KAS



4. A CODE-BASED ALGORITHM FOR KEY ENCAPSULATION (CAKE) SCHEME

In this chapter, the code-based key encapsulation algorithm CAKE (as described in the original literature in [1]) will be introduced. In defining the scheme, we will adopt the polynomial notation for circulant matrices as defined in Definition 2.0.2. Before elaborating on CAKE, we will first introduce some important definitions about quasi-cyclic codes, as defined in [1] and [7].

4.1 Quasi-cyclic Codes

4.1.1 Definitions

By convention, the notation for vectors will refer to row vectors, unless otherwise stated.

Definition 4.1.1 (Linear Code)**.** *Let C be a code of length n over a finite field $\mathbb{F}$. Then C is a **linear code** over $\mathbb{F}$ if it is a subspace of $\mathbb{F}^n$. i.e. C is a non-empty subset of $\mathbb{F}^n$ such that for all $\boldsymbol{x}, \boldsymbol{y} \in C$ and $a, b \in \mathbb{F}, a\boldsymbol{x} + b\boldsymbol{y} \in C$. Suppose $dim(C) = k$. Then C is an $[n, k]$-**(linear) code** over $\mathbb{F}$.*

Definition 4.1.2 (Dual Code of a Linear Code)**.** *Let $\mathbb{F}$ be a finite field. Let $\boldsymbol{a}$, $\boldsymbol{b} \in \mathbb{F}^n$. Define the dot product of $\boldsymbol{a}$ and $\boldsymbol{b}$ as follows:*

$$\boldsymbol{a} \cdot \boldsymbol{b} = \boldsymbol{a}\boldsymbol{b}^T$$

Let C be a linear code over $\mathbb{F}$. The dual code $C^\perp$ of C is then defined as

$$C^\perp = \{\boldsymbol{x} \in \mathbb{F} \mid \boldsymbol{x} \cdot \boldsymbol{c} = 0, \forall \boldsymbol{c} \in C\}.$$

Definition 4.1.3 (Hamming weight)**.** *Let $\boldsymbol{x}$ be a word of length n over $\mathbb{F}$. The **Hamming weight** of $\boldsymbol{x}$, denoted by $w(\boldsymbol{x})$, is defined to be the number of nonzero entries in $\boldsymbol{x}$.*

Theorem 4.1.4 (Dimension Theorem for Linear Codes). *Let $C \subseteq \mathbb{F}^n$ be a linear code. Then $dim(C) + dim(C^{\perp}) = n = dim(\mathbb{F}^n)$.*

Definition 4.1.5 (Cyclic Code). *A code $C \subseteq \mathbb{F}^n$ over a finite field $\mathbb{F}$ is called **cyclic** if*

(i) it is a linear code over $\mathbb{F}$

(ii) for any codeword $\boldsymbol{c} = c_0c_1c_2 \dots c_{n-1}$ in C, its cyclic shift $\sigma(\boldsymbol{c}) = c_{n-1}c_0c_1 \dots c_{n-2}$ is also a codeword in C.

Definition 4.1.6 (Quasi-cyclic (QC) Code). *A code $C \subseteq \mathbb{F}^n$ over a finite field $\mathbb{F}$ is called a **quasi-cyclic (QC) code of index** n_0 if*

(i) it is a linear code over $\mathbb{F}$

(ii) for any codeword $\boldsymbol{c} = c_0c_1c_2 \dots c_{n-1}$ in C, a word generated by a cyclic shift by n_0 positions $\sigma(\boldsymbol{c})^{n_0} = c_{-n_0 \pmod n}c_{1-n_0 \pmod n}c_{2-n_0 \pmod n} \dots c_{n-1-n_0 \pmod n}$ is also a codeword in C.

Definition 4.1.7 (Generator matrix). *Let C be an $[n,k]$-linear code over $\mathbb{F}$. Take a basis $\{\boldsymbol{g}_1, \boldsymbol{g}_2, \dots, \boldsymbol{g}_k\}$ for C. Define a $k \times n$ matrix $\boldsymbol{G} = \begin{pmatrix} \boldsymbol{g}_1 \\ \boldsymbol{g}_2 \\ \vdots \\ \boldsymbol{g}_k \end{pmatrix}$. Then, $\boldsymbol{G}$ is known as a **generator matrix** for the linear code C and $C = \{\boldsymbol{aG} \mid \boldsymbol{a} \in \mathbb{F}^k\}$.*

Definition 4.1.8 (Parity-check matrix). *Let C be an $[n,k]$-linear code over $\mathbb{F}$, and $\boldsymbol{H}$ be a generator matrix for the dual code $C^{\perp}$. Then $\boldsymbol{H}$ is called a **parity-check matrix** for C.*

Definition 4.1.9 (Quasi-cyclic Moderate Density Parity-Check (QC-MDPC) Codes). *An $[n,k]$-MDPC code is a linear code of length n and dimension k such that it is defined by a parity-check matrix with row weight w, where w is of the order $O(\sqrt{n \log n})$.*

*Hence, an $[n,k]$-**QC-MDPC code** is defined as a quasi-cyclic code defined by a parity-check matrix with row weight w, where w is of the order $O(\sqrt{n \log n})$.*

4.1.2 Construction of Quasi-cyclic (QC) Codes

In this section, we present a way of constructing quasi-cyclic codes, as used in a public key cryptosystem in [8]. We first observe the following.

Observation 4.1.10. *Let $n = n_0 r$. For $\boldsymbol{a} = (a_0, a_1, \ldots, a_{n-1}) \in \mathbb{F}^n$ and $i = 0, 1, \ldots, n_0 - 1$, let $\boldsymbol{a}^{(i)} = (a_i, a_{n_0+i}, \ldots, a_{r-1(n_0)+i})$.*

Define a mapping

$$\begin{aligned} \psi : \mathbb{F}^n &\to \mathbb{F}^n \\ \boldsymbol{a} &\mapsto (\boldsymbol{a}^{(0)}, \boldsymbol{a}^{(1)}, \ldots, \boldsymbol{a}^{(n_0-1)}) \end{aligned}$$

It is clear that ψ is an isomorphism.

Furthermore, observe that $\psi(\sigma^{n_0}(\boldsymbol{a})) = (\sigma(\boldsymbol{a}^{(0)}), \sigma(\boldsymbol{a}^{(1)}), \ldots, \sigma(\boldsymbol{a}^{(n_0-1)}))$. More generally, we have that for $j \in \mathbb{Z}^+$, $\psi((\sigma^{n_0})^j(\boldsymbol{a})) = \psi((\sigma^j)^{n_0}(\boldsymbol{a})) = (\sigma^j(\boldsymbol{a}^{(0)}), \sigma^j(\boldsymbol{a}^{(1)}), \ldots, \sigma^j(\boldsymbol{a}^{(n_0-1)}))$.

Construction 4.1.11 (Constructing a Quasi-cyclic Code)**.** *Let $n = n_0 r$. Construct an $r \times n$ matrix*

$$\boldsymbol{H} = \begin{pmatrix} \boldsymbol{H}_0 & \boldsymbol{H}_1 & \cdots & \boldsymbol{H}_{n_0-1} \end{pmatrix}$$

where $\boldsymbol{H}_0, \boldsymbol{H}_1, \ldots, \boldsymbol{H}_{n_0-1}$ are invertible circulant matrices. By the choice of $\boldsymbol{H}_i$, $rank(\boldsymbol{H}) = r$.

Let $D = \{\boldsymbol{a} \in \mathbb{F} \mid \boldsymbol{H}\boldsymbol{a}^T = \boldsymbol{0}\}$. i.e. D is the nullspace of $\boldsymbol{H}$. Note that by the Dimension Theorem for matrices, $dim(D) = nullity(\boldsymbol{H}) = n - rank(\boldsymbol{H}) = n - r$.

Define $C = \psi^{-1}(D)$. Then C is an $[n, n-r]$ quasi-cyclic code of index n_0.

It is clear that C is a linear code over $\mathbb{F}$. To show that C is an $[n, n-r]$ quasi-cyclic code of index n_0, it remains to show that for any codeword $\boldsymbol{a} \in C$, its cyclic shift by n_0 positions $\sigma^{n_0}(\boldsymbol{a})$ is also a codeword in C.

Since ψ is an isomorphism, and $\psi(\boldsymbol{a}) = (\boldsymbol{a}^{(0)}, \boldsymbol{a}^{(1)}, \ldots, \boldsymbol{a}^{(n_0-1)}) \in D$, this is equivalent to showing that $\psi(\sigma^{n_0}(\boldsymbol{a})) = (\sigma(\boldsymbol{a}^{(0)}), \sigma(\boldsymbol{a}^{(1)}), \ldots, \sigma(\boldsymbol{a}^{(n_0-1)})) \in D$. We will prove this in the next

proposition.

Proposition 4.1.12. *Given a parity check matrix* $\boldsymbol{H}$ *defined as in the above construction, let* $D = \{\boldsymbol{a} \in \mathbb{F} \mid \boldsymbol{H}\boldsymbol{a}^T = \boldsymbol{0}\}$. *If* $(\boldsymbol{a}^{(0)}, \boldsymbol{a}^{(1)}, \ldots, \boldsymbol{a}^{(n_0-1)}) \in D$ *then* $(\sigma(\boldsymbol{a}^{(0)}), \sigma(\boldsymbol{a}^{(1)}), \ldots, \sigma(\boldsymbol{a}^{(n_0-1)})) \in D$.

Proof. For each of the circulant matrices $\mathbf{H}_i, i \in \{1, 2, \ldots, n_0 - 1\}$, let h_i be the first row of matrix. Since $(\mathbf{a}^{(0)}, \mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(n_0-1)}) \in D$,

$$
\begin{aligned}
\mathbf{0} = \mathbf{H}\mathbf{a}^T = (\mathbf{H}_0, \mathbf{H}_1, \ldots, \mathbf{H}_{n_0-1}) &\begin{pmatrix} \mathbf{a}^{(0)^T} \\ \mathbf{a}^{(1)^T} \\ \vdots \\ \mathbf{a}^{(n_0-1)^T} \end{pmatrix} \\
\implies & \mathbf{H}_0\mathbf{a}^{(0)^T} + \mathbf{H}_1\mathbf{a}^{(1)^T} + \cdots + \mathbf{H}_{n_0-1}\mathbf{a}^{(n_0-1)^T} = \mathbf{0} \\
\implies & \mathbf{a}^{(0)}\mathbf{H}_0^T + \mathbf{a}^{(1)}\mathbf{H}_1^T + \cdots + \mathbf{a}^{(n_0-1)}\mathbf{H}_{n_0-1}^T = \mathbf{0}^T \\
\implies & \sum_{i=0}^{n_0-1} a^{(i)}\mathbf{H}_i^T = \sum_{i=0}^{n_0-1} a^{(i)}(h_i, \sigma(h_i), \ldots, \sigma^{r-1}(h_i)) = \mathbf{0} \\
\implies & (\sum_{i=0}^{n_0-1} a^{(i)}h_i, \sum_{i=0}^{n_0-1} a^{(i)}\sigma(h_i), \ldots, \sum_{i=0}^{n_0-1} a^{(i)}\sigma^{r-1}(h_i)) = \mathbf{0} \\
\implies & \sum_{i=0}^{n_0-1} a^{(i)}h_i = 0, \ \sum_{i=0}^{n_0-1} a^{(i)}\sigma(h_i) = 0, \ \ldots, \ \sum_{i=0}^{n_0-1} a^{(i)}\sigma^{r-1}(h_i) = 0 \qquad (*)
\end{aligned}
$$

On the other hand,

$$
\begin{aligned}
\sigma(\mathbf{a}^{(0)})\mathbf{H}_0^T + \sigma(\mathbf{a}^{(1)})\mathbf{H}_1^T + \cdots + \sigma(\mathbf{a}^{(n_0-1)})\mathbf{H}_{n_0-1}^T &= \sum_{i=0}^{n_0-1} \sigma(\mathbf{a}^{(i)})\mathbf{H}_i^T \\
&= \sum_{i=0}^{n_0-1} (\sigma(\mathbf{a}^{(i)})h_i, \sigma(\mathbf{a}^{(i)})\sigma(h_i), \ldots, \sigma(\mathbf{a}^{(i)})\sigma^{r-1}(h_i)) \\
&= \sum_{i=0}^{n_0-1} (\mathbf{a}^{(i)}\sigma^{r-1}(h_i), \mathbf{a}^{(i)}h_i, \ldots, \mathbf{a}^{(i)}\sigma^{r-2}(h_i)) \\
&= \mathbf{0}
\end{aligned}
$$

where the last equality follows from $(*)$. Hence, $(\sigma(\mathbf{a}^{(0)}), \sigma(\mathbf{a}^{(1)}), \ldots, \sigma(\mathbf{a}^{(n_0-1)})) \in D$. □

By Proposition 4.1.12, C is an $[n, n-r]$-quasi cyclic code of index n_0. The code used in the CAKE Scheme is a particular case of the codes in Construction 4.1.11. Since C and D are

equivalent codes, we shall only refer to the code D in our discussion, without transforming it using the isomorphism ψ.

4.2 Introduction to the CAKE Scheme

The Code-based Algorithm for Key Encapsulation (CAKE) scheme [5] is a key agreement scheme based on the hardness of the decoding problem. In fact, this problem is NP-complete [9], and are currently-known to be resistant against quantum computer attacks. This makes it a possible replacement option for existing key encapsulation systems based on number theory problems, should these problems be one day breakable using quantum computers.

Intrinsically, the codes are not guaranteed to be decoded successfully every time, even with knowledge of the private key. This gives rise to an associated decoding probability for each parameter setting used to generate the codes used in the scheme. Ideally, the decoding probability has to be **sufficiently high** (above a certain threshold probability) with knowledge of the private key (so that the code is **useful**), and **low** (under a certain threshold probability) when one does not have knowledge of the private key, so that the code is **secure** against eavesdroppers or malicious attackers. This is heavily dependent on the parameter setting of the code (to be discussed further in Section 4.2.4). Hence, there is a need to investigate the behaviour of the codes under different parameter settings, as we will later see in Chapter 6.

4.2.1 Key Generation

Let r, d_v and $t \in \mathbb{Z}$ such that the following conditions hold.

(i) r is a prime such that $(x^r - 1)/(x - 1) \in \mathbb{F}_2[x]$ is an irreducible polynomial.

(ii) d_v is odd.

(iii) t is the error-correcting capacity of the code (i.e. how many errors can be corrected with sufficiently high decoding probability, usually set to be at least 90%)

Remark 4.2.1. *Note that in the above set of conditions, $d_v < r$ is a small number relative to r. However, the exact range of d_v is not specified, and that one of the aims of our experiments*

in Chapter 6 will be to determine a suitable range for d_v, based on the set of other parameters used.

Input: r, d_v and $t \in \mathbb{Z}$
Output: Sparse private key $(\mathbf{h}_0, \mathbf{h}_1)$ and dense public key $(\mathbf{g}_0, \mathbf{g}_1)$

Define $\mathcal{R}$ to be the ring $\mathbb{F}_2[x]/\langle x^r - 1\rangle$. Let $h_0(x), h_1(x), g(x) \in \mathcal{R}$ be polynomials sampled uniformly at random from the ring $\mathcal{R}$. Let the polynomials be the respective Hall polynomials of the circulant matrices $\mathbf{H}_0, \mathbf{H}_1, \mathbf{G}'$ of order r, and represent the corresponding first rows of the matrices by $\mathbf{h}_0$, $\mathbf{h}_1, \mathbf{g}$. Note that the Hall polynomials can be identified with the corresponding first rows of the circulant matrices.

The polynomials are chosen so that $\mathbf{h}_0$, $\mathbf{h}_1, \mathbf{g}$ have the following properties:

(i) $w(\mathbf{h}_0) = w(\mathbf{h}_1) = d_v$.

(ii) $w(\mathbf{g})$ is odd, and $w(\mathbf{g}) \approx \frac{r}{2}$.

Let $\mathbf{g}_0$ be the first row of the circulant matrix $\mathbf{G}'\,\mathbf{H}_1^T$, and $\mathbf{g}_1$ be the first row of the circulant matrix $\mathbf{G}'\,\mathbf{H}_0^T$. Set $\mathbf{G}_0 = \mathbf{G}'\,\mathbf{H}_1^T, \mathbf{G}_1 = \mathbf{G}'\,\mathbf{H}_0^T$. Note that all these circulant matrices are of order r.

Remark 4.2.2. *Using the notation in Definition 2.0.4 and by Proposition 2.0.6, $\boldsymbol{G}'\boldsymbol{H}_i^T$ is circulant and its Hall polynomial is $g(x)h_i(x^{-1}) \pmod{x^r - 1}$, for $i \in \{0, 1\}$.*

Define the parity-check matrix $\mathbf{H} = \begin{pmatrix} \mathbf{H}_0 & \mathbf{H}_1 \end{pmatrix}$ as in Construction 4.1.11, and the generator matrix $\mathbf{G} = \begin{pmatrix} \mathbf{G}_0 & \mathbf{G}_1 \end{pmatrix}$.

Remark 4.2.3. *It can be shown that $\boldsymbol{G}$ is indeed a generator matrix for the code defined by $\boldsymbol{H}$. Let $c(x)$ be the first row of $\boldsymbol{G}\boldsymbol{H}^T = \boldsymbol{G}_0\boldsymbol{H}_0^T + \boldsymbol{G}_1\boldsymbol{H}_1^T$. Then*

$$\begin{aligned} c(x) &\equiv g(x)h_1(x^{-1})h_0(x^{-1}) + g(x)h_0(x^{-1})h_1(x^{-1}) \pmod{x^r - 1} \\ &= 2g(x)h_0(x^{-1})h_1(x^{-1}) \\ &= 0 \end{aligned}$$

Hence, $\boldsymbol{GH}^T = \boldsymbol{0}$*. Thus, every codeword* $\boldsymbol{aG}$ *is contained in the nullspace of* $\boldsymbol{H}$*. By the dimension argument below,* $\boldsymbol{G}$ *indeed generates all words in the nullspace of* $\boldsymbol{H}$*.*

Let C be the QC-MDPC code generated by $\mathbf{G}$. Note that $\mathbf{H}$ and $\mathbf{G}$ have to be of full rank, so that $dim(C^{\perp}) + dim(C) = rank(\mathbf{H}) + rank(\mathbf{G}) = r + r = 2r = n$, which concurs with Theorem 4.1.4, where n is the length of the code. This condition is satisfied by the choice of $h_0(x), h_1(x), g(x)$, as we will later show in Section 4.3.2.

4.2.2 Code Encapsulation

Let $\mathbf{K} : \{0,1\}^n \to \{0,1\}^{l_k}$ be the hash function used for both code encapsulation and decapsulation, where l_k is the final desired symmetric key length. Upon obtaining the public key $(\mathbf{g}_0, \mathbf{g}_1)$ and private key $(\mathbf{h}_0, \mathbf{h}_1)$, we can generate a cryptogram $\mathbf{c}$ and encapsulated key K using the following procedure.

1. Generate random vectors $\mathbf{e}_0, \mathbf{e}_1, \mathbf{m} \in \mathcal{R}$, where $w(\mathbf{e}_0) + w(\mathbf{e}_1) = t$.
2. Let the error pattern be $\mathbf{e} = (\mathbf{e}_0, \mathbf{e}_1)$.
3. Compute $\mathbf{c} = (\mathbf{c}_0, \mathbf{c}_1) = (\mathbf{mG}_0 + \mathbf{e}_0, \mathbf{mG}_1 + \mathbf{e}_1)$.
4. Compute the encapsulated key $K = \mathbf{K}(\mathbf{e})$.

Remark 4.2.4. *In step 3, for* $i \in \{0,1\}$*, we can also calculate the corresponding polynomial of* $\boldsymbol{c}_i$*,* $c_i(x) \equiv m(x)g_i(x) + e_i(x) \pmod{x^r - 1}$.

Remark 4.2.5. *The matrices* $\boldsymbol{G}$ *and* $\boldsymbol{H}$ *are invertible, by the choice of the polynomials* $\boldsymbol{h}_0, \boldsymbol{h}_1$*, and* $\boldsymbol{g}$*. The invertibility of the circulant matrices represented by these three polynomials are shown in Section 4.3.2.*

This invertibility property of the matrices are required, to ensure a bijective mapping for m and c. This means that we do not have $\boldsymbol{m}$ *and* $\boldsymbol{m}'$ *mapping to the same codeword* $\boldsymbol{c}$*, if* $\boldsymbol{m} \neq \boldsymbol{m}'$*. For if such a scenario occurs, it will be difficult to know if we have indeed decoded* $\boldsymbol{c}$ *to the correct original vector* $\boldsymbol{m}^*$*, or another vector m' which also maps to c.*

4.2.3 Code Decapsulation

Given the sparse private key $(\mathbf{h}_0, \mathbf{h}_1)$ and the cryptogram $\mathbf{c}$, we obtain the decapsulated key K, the procedure is as follows:

1. Decode the cryptogram $\mathbf{c}$ to obtain an error vector $\mathbf{e}' = (\mathbf{e}'_0, \mathbf{e}'_1)$. If $w(\mathbf{e}') \neq t$, or decoding fails, return a message indicating decoding failure and terminate.

2. Compute the decapsulated key $K' = \mathbf{K}(\mathbf{e}')$.

For Step 1, the code $\mathbf{c}$ can be decoded using common MDPC code decoding algorithms, such as the Bit-flipping and Sum Product algorithm. The details are expounded upon in Section 5.1. It is also desired that $K' = K$. Later, we will see in our empirical results in Chapter 6 that this is indeed often the case.

4.2.4 Parameter Settings of CAKE and its security

As highlighted at the start of this section, in code-based cryptography schemes, the (randomly) generated codes are not guaranteed to be decoded, even when one has knowledge of the private key $\mathbf{h}$. This gives rise to an associated decoding probability for each parameter setting of CAKE. The decoding probability can be defined as the probability that a code is correctly decoded. We provide a more formal definition of this in Chapter 6 when we run experiments using this as a metric.

Ideally it is required that the decoding probability is high with knowledge of the private key, and low otherwise. The former is required so that the code **remains useful in practice** (if decoding probability using the private key is too low, there will be too many instances whereby the original vector $\mathbf{m}$ cannot be recovered), while the latter ensures **security of the code**, since anyone not in possession of the private key will not be able to break the code easily, allowing the code to be secure against malicious attackers. However, achieving both of them is not trivial.

For higher security of the code, it is usually desired for t and d_v to be high, since

- Codes with low-weight parity-check matrices can be easily broken by finding dual low weight codewords to build a sparse generator matrix for the dual code (the parity-check matrix for the actual code) [8], since the generator matrix for the dual code is the parity-check matrix for the actual code. Hence, d_v has to be sufficiently high for a secure code.
- With a higher value of t, this means that more errors are introduced into the code. Higher values of t usually lead to higher security.

Hence, to achieve low decoding probability without knowledge of the private key, we want to be able to introduce more errors (higher value of t) and higher code-weight (higher d_v). However, doing so will lead to low decoding probability even with knowledge of the private key. If decoding probability falls below a certain threshold even with knowledge of the private key, the code will not be useful since the probability of recovering the original message is low.

On the other hand, suppose we enforce that decoding probability with knowledge of the private key is kept above a certain threshold. As we will see later in Chapter 6, for decoding probability to stay above a certain threshold for higher values of t, usually a lower (Hamming) weight is required. However, as explained above, if the weight of the code is too low, security will be compromised. If we increase d_v, then t will be forced to decrease in order to maintain a high decoding probability (using the private key).

Hence, the weight of the code is a crucial factor in determining the security of the code. It is determined by d_v, and can neither be too large nor too small. If d_v is too large, very few errors can be corrected, and t is forced to be low in order to keep decoding probability using the private key sufficiently high. Consequently, a lower value of t makes it easier for an attacker to break the code. If d_v is too small, t can be allowed to be high while keeping decoding probability using the private key high as well. However, the security of the code will also be compromised this way.

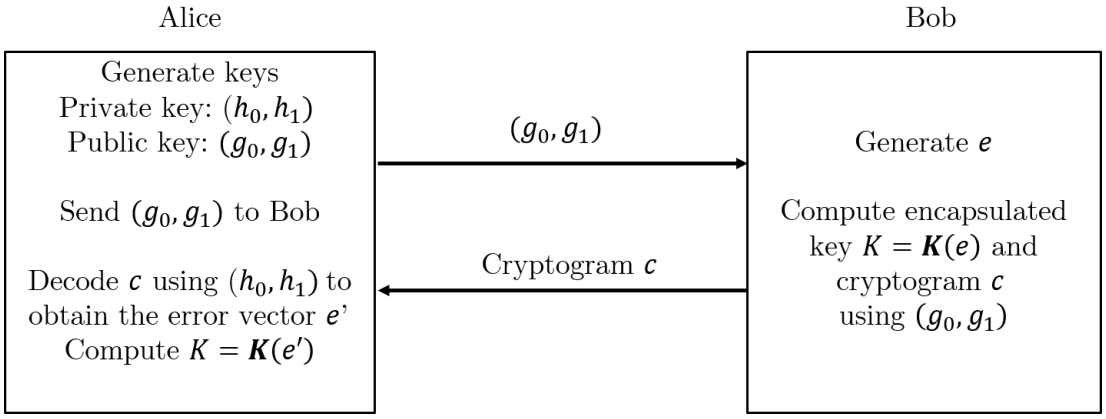
As there is no exact definition for the weight of MDPC codes, suitable weights for the codes used in CAKE have to be determined through the running of experiments and observing the decoding probability, as seen in Chapter 6. We aim to find a parameter setting of d_v that will

balance the trade-offs as discussed above.

4.2.5 An Illustration of the CAKE Scheme

The CAKE Scheme is a combination of the procedures in Section 4.2.1, 4.2.2 and 4.2.2. In Figure 4.1, we present an illustration of this scheme.

Fig. 4.1: An Illustration of the CAKE Scheme



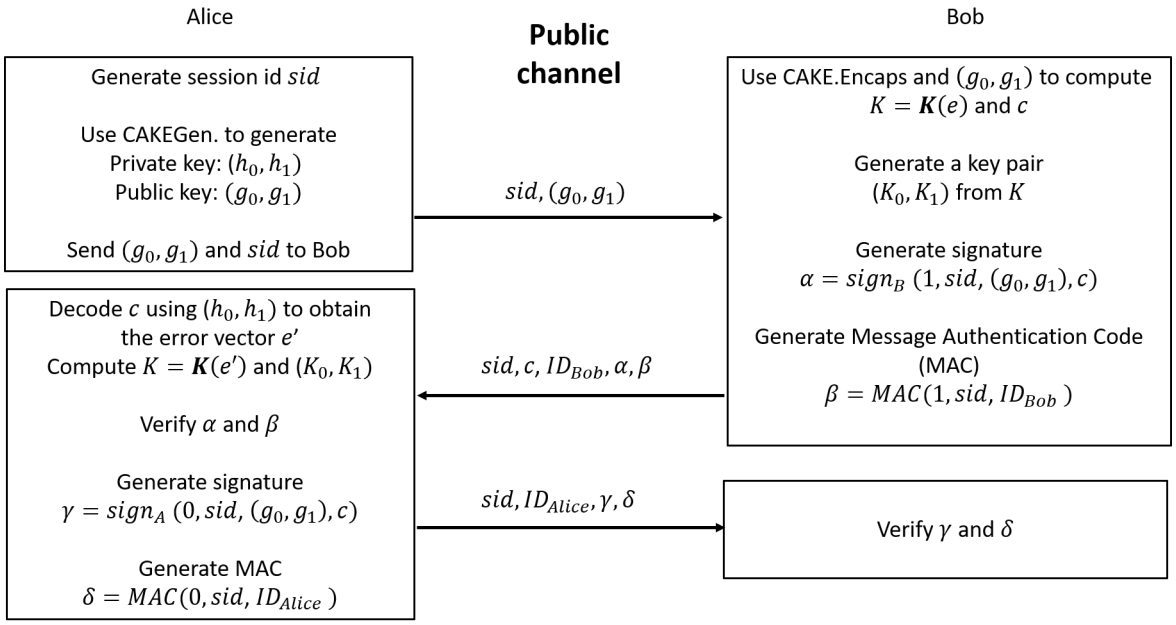
Under the CAKE scheme, ephemeral (one-time-use) keys are generated, i.e. each key is only used once and discarded. This makes the scheme secure against known attacks such as the Guo, Johansson and Stankovski (GJS) attack, which makes use of decoding patterns obtained from the encoding and decoding of many messages using the same key. In terms of time complexity and memory usage, compared to some other encoding schemes which require computation of inverses and storing of the entire generator and parity-check matrix, CAKE takes a shorter time for encoding since there is no need to calculate the inverse of circulant matrices, and requires less memory as we can store just the Hall polynomial of each circulant matrix, instead of entire matrices.

Note that this above-illustrated scheme is only a basic skeleton for the key agreement scheme. For real-life implementation, there is a need to introduce a few more authentication steps to fend off attacks on the protocol and allow for user authentication, such as the MITM attack. In the next section, we will describe a more sophisticated protocol which allows for authenticated key agreement.

4.2.6 A Key Agreement Protocol based on the CAKE Scheme

As described at the end of the previous section, the CAKE scheme presented in Section 4.2.5 forms the basis for more complicated protocols. One such protocol described in the original literature uses a combination of the SiGn-and-MAc (SIGMA) protocol and the CAKE scheme to establish a common key between Alice and Bob. This is a more sophisticated protocol compared to the CAKE scheme by itself, and facilitates authenticated key establishment. However, as the protocol itself is not of much mathematical interest, we will simply present it (refer to Figure 4.2), but not delve into its details.

Fig. 4.2: An Illustration of the SIGMA protocol based on CAKE



4.3 Feasibility of the CAKE Scheme

4.3.1 Obtaining an irreducible polynomial $(x^r - 1)/(x - 1)$

The CAKE scheme detailed in Section 4.2.1 explicitly requires r to be a prime such that $(x^r - 1)/(x - 1) \in \mathbb{F}_2)[x]$ is an irreducible polynomial. It is obvious that if $(x^r - 1)/(x - 1)$ is irreducible, then r must be a prime number. But the prime number condition is not sufficient for irreducibility in some fields. For example, when $r = 7$ and $\mathbb{F} = \mathbb{F}_2$, the polynomial $(x^7 - 1)/(x - 1) = (1 + x + x^3)(1 + x^2 + x^3)$ is reducible.

Here, we present a theorem which will provide us with a criteria to select suitable r values

for the scheme. It will also be shown that a specific group of prime numbers, called the **Sophie Germain Primes**, satisfy the desired condition, under a simple criteria.

We first state the following Lemmas.

Lemma 4.3.1. *Let r be a positive integer and $\mathbb{F}$ be a finite field of order q. There exists an element of (multiplicative) order r in $\mathbb{F}$ if and only if $q \equiv 1 \pmod{r}$.*

Proof. ($\Longrightarrow$) Suppose there exists $\beta \in \mathbb{F}$ such that the order of β is r.

Since $\mathbb{F}^*$ is a group of order $q-1$ and $\beta \in \mathbb{F}^*$, by Lagrange's Theorem, $r \mid q-1$. Hence, $q \equiv 1 \pmod{r}$.

($\Longleftarrow$) Suppose $q \equiv 1 \pmod{r}$. Then $r \mid q-1$.

Let α be a primitive element of $\mathbb{F}$. Then $\alpha^{q-1} = 1$. Define $\beta = \alpha^{(q-1)/r}$. Then $\beta^r = \alpha^{q-1} = 1$. Since $\alpha, \alpha^2, \ldots, \alpha^{q-1}$ are distinct and $q-1$ is the smallest positive integer such that $\alpha^{q-1} = 1$, the order of β is r. □

Lemma 4.3.2. *If β is a root of a polynomial f(x) (i.e. $f(\beta) = 0$), then the minimal polynomial with respect to β, $M_\beta(x)$, divides f(x).*

Proof. Let $d(x) = gcd(f(x), M_\beta(x))$. Since $M_\beta(x)$ is irreducible, $d(x) = M_\beta(x)$ or $d(x) = 1$.

Suppose $d(x) = 1$. Then there exists polynomials $p(x), q(x)$ such that $p(x)f(x)+q(x)M_\beta(x) = 1$. But $1 = p(\beta)f(\beta) + q(\beta)M_\beta(\beta) = p(\beta) \cdot 0 + q(\beta) \cdot 0 = 0$, which is a contradiction.

So we must have $d(x) = M_\beta(x)$. Hence, $M_\beta(x) \mid f(x)$. □

Lemma 4.3.3. *Let β be a nonzero element of a finite field $\mathbb{F}$.*

If the order of β is $r > 1$, then the minimal polynomial of β over $\mathbb{F}$ divides the polynomial $(x^r - 1)/(x - 1) = 1 + x + \cdots + x^{r-1}$.

Proof. As $\beta^r = 1$, β is a root of $x^r - 1 = (x-1)(1 + x + \cdots + x^{r-1})$.

On the other hand, $r > 1$ implies $\beta \neq 1$ (β is not a root of $x - 1$). Hence, β is a root of $1 + x + \cdots + x^{r-1}$. By Lemma 4.3.2, the minimal polynomial of β divides the polynomial $1 + x + \cdots + x^{r-1}$. □

Theorem 4.3.4. *Let $r > 1$ be a prime and $\mathbb{F}$ be a finite field of order q, where $q \not\equiv 0 \pmod{r}$. The polynomial $(x^r - 1)/(x - 1) = 1 + x + \cdots + x^{r-1}$ is irreducible over $\mathbb{F}$ if and only if for all prime divisors p of $r - 1$, $q^{(r-1)/p} \not\equiv 1 \pmod{r}$.*

Proof. We prove the following equivalent statement:

$1 + x + \cdots + x^{r-1}$ is reducible over $\mathbb{F}$ if and only if there exists a prime divisor p of $r - 1$ such that $q^{(r-1)/p} \equiv 1 \pmod{r}$.

($\Longleftarrow$) Suppose there exists a prime divisor p of $r - 1$ such that $q^{(r-1)/p} \equiv 1 \pmod{r}$.

Let $\mathbb{K}$ be a finite extension of $\mathbb{F}$ of order $q^{(r-1)/p}$. By Lemma 4.3.1, there exists $\beta \in \mathbb{K}$ of order $r > 1$. Furthermore, by Lemma 4.3.3, the minimal polynomial $M_\beta(x)$ of β over $\mathbb{F}$ is a divisor of the polynomial $1 + x + \cdots + x^{r-1}$.

Let $m = deg(M_\beta(x))$, where $M_\beta(x) = a_0 + a_1 x + \cdots + a_m x^m$.

Define

$$\mathbb{F}(\beta) = \{b_1\beta^{m-1} + b_2\beta^{m-2} + \cdots + b_m \mid b_i \in \mathbb{F}\},$$

with $\beta^m = a_0 + a_1 x + \cdots + a_{m-1} x^{m-1}$. $\mathbb{F}(\beta)$ is a subfield of $\mathbb{K}$ of order q^m.

Hence,

$$\begin{aligned} |\mathbb{F}(\beta)| = q^m &\leq |\mathbb{K}| \\ &= q^{(r-1)/p} \\ &< q^{r-1} \\ \Longrightarrow m &< r - 1 \end{aligned}$$

So, $M_\beta(x)$ is a proper divisor of $1 + x + \cdots + x^{r-1}$ and thus $1 + x + \cdots + x^{r-1}$ is reducible over $\mathbb{F}$.

($\Longrightarrow$) Suppose $1 + x + \cdots + x^{r-1}$ is reducible over $\mathbb{F}$.

Let $\mathbb{L}$ be a finite extension of $\mathbb{F}$ of order q^{r-1}.

Since r is a prime and $q \not\equiv 0 \pmod{r}$ i.e. $r \nmid q$, by Fermat's Little Theorem, $q^{r-1} \equiv 1 \pmod{r}$.

By Lemma 4.3.1 and Lemma 4.3.3 again, there exists an element $\beta \in \mathbb{L}$ of order r, such that the minimal polynomial $M_\beta(x)$ of β over $\mathbb{F}$ is a divisor of $1+x+\cdots+x^{r-1}$. By assumption,

$1+x+\cdots+x^{r-1}$ is reducible over $\mathbb{F}$. Hence, $M_\beta(x)$ is a **proper divisor** of $1+x+\cdots+x^{r-1}$.

Let $m = deg(M_\beta(x))$. Note that since $M_\beta(x)$ is a proper divisor of $1 + x + \cdots + x^{r-1}$, $1 \leq m < r - 1$. By the same argument as in the above, $\mathbb{F}(\beta)$ is a finite field of order q^m.

Observe that $\beta^r = 1$ and $\beta \in \mathbb{F}(\beta)$. By Lemma 4.3.1,

$$q^m \equiv 1 \pmod{r} \tag{$*$}$$

Since $\mathbb{F}(\beta)$ is a subfield of $\mathbb{L}$,

$$\exists t \in \mathbb{Z} \text{ such that } q^{r-1} = |\mathbb{L}| = |\mathbb{F}(\beta)|^t = q^{mt}$$
$$\implies r - 1 = mt \tag{$**$}$$

Since $m < r - 1$, we have $t > 1$.

Take any prime divisor p of t.

Then by $(*)$ and $(**)$,

$$q^{(r-1)/p} = (q^m)^{t/p} \equiv 1^{t/p} \equiv 1 \pmod{r}$$

and we arrive at the desired result.

□

The theorem above provides us with a way to check for suitable values of r for the code-based key encapsulation scheme CAKE as detailed in Section 4.2. Namely, we can iterate through a range of candidate values of r, and check the following condition: for all prime divisors p of $r - 1$, $2^{(r-1)/p} \not\equiv 1 \pmod{r}$. Here, we use $q = 2$, since the finite field used is $\mathbb{F}_2$.

Besides the above way of obtaining suitable values of r, we can also look at a specific group of primes, termed the **Sophie Germain primes**. We will prove a result about this group of prime numbers, which will provide another possible method of obtaining appropriate r values.

Definition 4.3.5 (Sophie Germain Primes)**.** *Let s be a prime number. If $2s + 1$ is also a prime, then s is a* ***Sophie Germain prime****.*

For example, 11 *is a Sophie Germain prime, since* $11(2)+1 = 23$ *is also a prime number. Similarly,* $23, 29$ *and* 41 *are also Sophie Germain primes.*

Corollary 4.3.6. *Let* $s \geq 3$ *be a Sophie Germain prime and let* $r = 2s+1$*. Then* $(x^r-1)/(r-1)$ *is irreducible over* $\mathbb{F}_2$ *if and only if* 2 *is not a square modulo* r*.*

Proof. Let s and $r = 2s + 1$ be primes.

By Theorem 4.3.4, $(x^r - 1)/(r - 1) = 1 + x + \cdots + x^{r-1}$ is irreducible over $\mathbb{F}_2$ if and only if for all prime divisors p of $r - 1$, $q^{(r-1)/p} \not\equiv 1 \pmod{r}$.

Note that since p and s are primes, $p \mid r - 1 \iff p \mid 2s \iff p = 2$ or $p = s$. Hence we have the following equivalent condition.

$$\begin{aligned} &\text{For all primes } p \mid r-1,\ 2^{(r-1)/p} \not\equiv 1 \pmod{r} \\ \iff\ &\text{For all primes } p \mid 2s,\ 2^{(r-1)/p} \not\equiv 1 \pmod{r} \\ \iff\ &\text{For } p = 2 \text{ and } p = s,\ 2^{(r-1)/p} \not\equiv 1 \pmod{r} \end{aligned}$$

We proceed to check the two cases for $p = 2$ and $p = s$.

Case 1: p = s

For $r \geq 5$,

$$2^{(r-1)/s} \equiv 2^{2s/s} \equiv 2^2 \equiv 4 \not\equiv 1 \pmod{r}$$

Since $2^{(r-1)/s} \not\equiv 1 \pmod{r}$ for all $r \geq 5$, we only need to show that for $p = 2$, $2^{(r-1)/p} \not\equiv 1 \pmod{r}$ if and only if 2 is a not a square modulo r.

Case 2: p = 2

$$2^{(r-1)/2} \equiv 2^{2s/2} \equiv 2^s \pmod{r}$$

Let α be a primitive element of the finite field $\mathbb{F}$, where $|\mathbb{F}^*| = r - 1$.

Note that $\alpha^{2s} \equiv \alpha^{r-1} \equiv 1 \pmod{r}$. Since α is a primitive element, $\alpha, \alpha^2, \ldots, \alpha^{r-1}$ are distinct elements. Hence, $s < 2s = r - 1$ implies that $\alpha^s \not\equiv 1 \pmod{r}$, i.e. $\alpha^s \equiv -1 \pmod{r}$.

There exists $j \in \mathbb{Z}$ such that $\alpha^j \equiv 2 \pmod{r}$, and j can be odd or even.

Suppose 2 is a square modulo r, i.e. $j = 2m$ for some $m \in \mathbb{Z}^+$, and we have $2 \equiv \alpha^{2m} \pmod{r}$. Then,

$$2^s \equiv (\alpha^{2m})^s \equiv (\alpha^{2s})^m \equiv 1^{2m} \equiv 1 \pmod{r} \qquad (*)$$

Conversely, suppose 2 is not a square modulo r, i.e. $j = 2m+1$ for some $m \in \mathbb{Z}^+$, and we have $2 \equiv \alpha^{2m+1} \pmod{r}$. Then,

$$2^s \equiv (\alpha^{2m+1})^s \equiv (\alpha^{2s})^m \alpha^s \equiv 1^m(-1) \equiv -1 \pmod{r} \qquad (**)$$

Combining $(*)$ and $(**)$, 2 is a square modulo $r \iff 2^s \equiv 1 \pmod{r}$.

Hence,

$$\begin{aligned}(x^r - 1)/(r-1) \text{ is irreducible over } \mathbb{F}_2 &\iff 2^{(r-1)/p} \not\equiv 1 \pmod{r} \text{ for } p = 2 \text{ and } p = s \\ &\iff 2 \text{ is not a square modulo } r.\end{aligned}$$

□

This result provides an alternate method for finding suitable r values. More specifically, if 2 is **not** a square modulo r, where $s = (r-1)/2$ is a Sophie Germain prime, the corresponding polynomial $(x^r - 1)/(x - 1)$ is irreducible over $\mathbb{F}_2$. It is known that

$$\begin{cases} r \equiv 3 \text{ or } 5 \pmod{8} \implies 2 \text{ is not a square modulo } r \\ r \equiv 1 \text{ or } 7 \pmod{8} \implies 2 \text{ is a square modulo } r \end{cases} \qquad (4.1)$$

Condition 4.1 can hence be used for checking if 2 is a square modulo r [10].

Using a program, we ran the first checking algorithm above. The list of primes r between 100 and 200, the irreducibility of $(x^r - 1)/(x - 1)$ over $\mathbb{F}_2$, and whether the corresponding $(r-1)/2$ are Sophie Germain primes are as listed in Table 4.1.

Out of 21 primes in the range, for 10 of them, $(x^r - 1)/(x - 1)$ is irreducible over $\mathbb{F}_2$, and out of these 10 primes, only two of them are Sophie Germain primes.

Tab. 4.1: Table detailing primes from 100 and 200 and their suitability for the CAKE Scheme

r	**Is** $(r-1)/2$ **a Sophie Germain prime?**	**Is** $(x^r-1)/(x-1)$*irreducible*?
101	F	T
103	F	F
107	T	T
109	F	F
113	F	F
127	F	F
131	F	T
137	F	F
139	F	T
149	F	T
151	F	F
157	F	F
163	F	T
167	T	F
173	F	T
179	T	T
181	F	T
191	F	F
193	F	F
197	F	T
199	F	F

4.3.2 *Choosing invertible matrices*

In Section 4.2.1, $\mathbf{h}_0, \mathbf{h}_1$ and $\mathbf{g}$ are chosen to be of odd (Hamming) weights, from the ring $\mathbb{F}_2[x]/\langle x^r - 1\rangle$. The details regarding the choice of r have been described in Section 4.3.1. In this subsection, we will prove that such a choice for the first row of the circulant matrices $\mathbf{H}_0, \mathbf{H}_1$ and $\mathbf{G}'$ will give us the desired invertibility property of the matrices.

Theorem 4.3.7. *Let $\boldsymbol{A}$ be a non-zero circulant matrix of order r over a finite field $\mathbb{F}$. Then $\boldsymbol{A}$ is invertible if and only if the Hall polynomial of $\boldsymbol{A}$ is relatively prime to $x^r - 1$.*

Proof. Given a circulant matrix $\mathbf{A}$, we define $w(\mathbf{A})$ to be the weight of the first row of $\mathbf{A}$, i.e. the number of nonzero entries in the first row. Note that the Hall polynomial of $\mathbf{I}, I(x) = 1$. Hence, letting the Hall polynomial of $\mathbf{A}$ be $a(x)$,

$$\begin{aligned} \mathbf{A} \text{ is invertible} &\iff \text{there exists a circulant matrix } \mathbf{B} \in \mathbb{F}^r \text{ such that } \mathbf{AB} = \mathbf{I} \\ &\iff \text{there exists } b(x) \text{ such that } a(x)b(x) \equiv I(x) \equiv 1 \pmod{x^r - 1} \\ &\iff a(x) \text{ is relatively prime to } x^r - 1 \end{aligned}$$

where the second "if and only if" statement holds by Proposition 2.0.6. □

Corollary 4.3.8. *Let $\mathbb{F}$ be a field and r a positive integer such that $(x^r - 1)/(x - 1) = 1 + x + \cdots + x^{r-1}$ is irreducible over $\mathbb{F}$. Let $\boldsymbol{A}$ be a non-zero circulant matrix of order r, with Hall polynomial $a(x)$.*

Then $\boldsymbol{A}$ is invertible if and only if $a(1) \neq 0$ and $\boldsymbol{A} \neq c\boldsymbol{J}$, where $c \in \mathbb{F}^*$ and $\boldsymbol{J}$ is the square matrix of order r such that all its entries are 1.

In particular, if $\mathbb{F} = \mathbb{F}_2$, then $\boldsymbol{A}$ is invertible if and only if $w(\boldsymbol{A})$ is an odd number less than r. Here, $w(\boldsymbol{A})$ refers to the Hamming weight of the word represented by the first row of $\boldsymbol{A}$.

Proof. By Theorem 4.3.7, a circulant matrix $\mathbf{A}$ of order r is invertible iff $a(x)$ is relatively prime to $x^r - 1$. Since $a(x)$ is relatively prime to $x^r - 1 = (x - 1)(1 + x + \cdots + x^{r-1})$ iff $x - 1 \nmid a(x)$ and $1 + x + \cdots + x^{r-1} \nmid a(x)$, it is sufficient to prove the following:

(i) $x - 1 \mid a(x) \iff a(1) = 0$, and

(ii) $1 + x + \cdots + x^{r-1} \mid a(x) \iff \mathbf{A} = c\mathbf{J}$.

Since $x - 1 \mid a(x) \iff 1$ is a root of $a(x) \iff a(1) = 0$, (i) is true. It remains to prove (ii).

($\Longleftarrow$) Suppose $\mathbf{A} = c\mathbf{J}$ for some $c \in \mathbb{F}^*$. Then the first row of $\mathbf{A}$ is $(c, c, \ldots, c)$.

Hence, $a(x) = c(1 + x + \cdots + x^{r-1})$. i.e. $1 + x + \cdots + x^{r-1} \mid a(x)$.

($\Longrightarrow$) Suppose $1 + x + \cdots + x^{r-1} \mid a(x)$.

Then there exists a polynomial $p(x) = p_0 + p_1 x + \cdots + p_{r-1}x^{r-1}$ such that $a(x) \equiv p(x)(1 + x + \cdots + x^{r-1}) \pmod{x^r - 1}$.

Hence, we have

$$\begin{aligned} a(x) &\equiv p(x)(1 + x + \cdots + x^{r-1}) \\ &\equiv (p_0 + p_1 x + \cdots + p_{r-1}x^{r-1})(1 + x + \cdots + x^{r-1}) \\ &\equiv c(1 + x + \cdots + x^{r-1}) \pmod{x^r - 1} \end{aligned}$$

where $c = p_0 + p_1 + \cdots + p_{r-1}$ is a constant and hence $\mathbf{A} = c\mathbf{J}$.

If $\mathbb{F} = \mathbb{F}_2$, $c = 1 \in \mathbb{F}^*$. Furthermore, $a(1) = w(\mathbf{A}) \not\equiv 0 \pmod 2 \iff w(\mathbf{A})$ is odd. Hence, $\mathbf{A}$ is invertible iff $w(\mathbf{A})$ is odd and $w(\mathbf{A}) < r$ (i.e. $\mathbf{A} \neq \mathbf{J}$).

□

Hence, by choosing $\mathbf{h}_0, \mathbf{h}_1$ and $\mathbf{g}$ to be of odd (Hamming) weight, by Corollary 4.3.8, the corresponding circulant matrices will be invertible.

5. DECODING MODERATE DENSITY PARITY-CHECK (MDPC) CODES

In this chapter, we will describe two common algorithms used for decoding Moderate Density Parity-Check (MDPC) codes. Following that, we will explore their usage in decoding the codes generated in the CAKE scheme.

5.1 Decoding Algorithms

A popular algorithm used for decoding MDPC codes is the Bit-Flipping Algorithm, as described in [8]. Another method commonly used for decoding of similar codes is the Sum Product Algorithm. In this section, we will describe both methods of decoding and how they work. The algorithm description and relevant proofs are based on the algorithms presented in [7].

Since decoding is an NP-complete problem, the algorithms here do not always output the original codeword. Hence, when using them for decoding CAKE codes, one of our concerns is the time required and the decoding probability achieved using each of these methods, which will be further discussed in Chapter 6.

5.1.1 Tanner Graph

In both the Bit-Flipping and Sum Product Algorithm, a special type of bipartite graph, called the **Tanner graph** is used for the decoding. Therefore, we will first describe the Tanner graph.

Definition 5.1.1 (Tanner Graph)**.** *Let $\boldsymbol{H}$ be an $m \times n$ parity-check matrix of a binary MDPC code.*

For $i = 1, 2, \ldots, m$, let $r_{i1}c_1 + r_{i2}c_2 + \cdots + r_{in}c_n = 0$ be the parity-check equations, i.e. $\boldsymbol{r}_i = (r_{i1}, r_{i2}, \ldots, r_{in})$ is the ith row of $\boldsymbol{H}$.

*The **Tanner graph** is defined to be a bipartite graph, with two disjoint types of nodes: n bit nodes and m check nodes respectively. The n bit nodes represent the n bits of a word, whereas the check nodes represent the m parity-check equations. An edge e_{ij} exists between the ith bit node and the jth check node if and only if $r_{ij} = 1$.*

5.1.2 Bit-flipping Algorithm

Let $\mathbb{F} = \mathbb{F}_2$, $\mathbf{H}$ be the parity-check matrix, $\mathbf{x} = (x_0, x_1, \ldots, x_{n-1})$ be the original codeword, $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$ be the word received and N be the limit on the number of iterations. The guiding steps and principle behind the algorithm is as follows.

Step 1: Initialize $t = 1$. For $j = 1, 2, \ldots, n$, set the value of the jth bit node to be c_j.

Step 2: Send the values of the bit nodes to their neighbouring check nodes through the edges.

Set the value of each check node to be the sum (modulo 2) of values received from the bit nodes. If the values of all the check nodes are zero, then $\mathbf{c}$ is a valid codeword. Stop and decode $\mathbf{c}$ to $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, where x_j is the current value of the jth bit node.

Step 3: If $\mathbf{c}$ is not a codeword, i.e. at least one of the check nodes have a non-zero value, send the values of check nodes back to their neighbouring bit nodes through the edges. Note that for each bit node, this is also the number of violated parity-check equations involving that bit.

Step 4: For each bit node, if majority of the values received from the adjacent check nodes are 1, then flip its value from 0 to 1 or from 1 to 0. If $t < N$, then set $t = t + 1$ and go to step 2. If $t = N$, stop and we fail to decode $\mathbf{c}$.

Remark 5.1.2. *Note that a word $\mathbf{c}$ satisfies all the parity-check equations if and only if it is a valid codeword.*

In step 2, using the definition of the Tanner graph in Definition 5.1.1, the sum $r_{i1}c_1 + r_{i2}c_2 + \cdots + r_{in}c_n$ corresponding to the ith check node is the sum of the values of each of its neighbouring bit nodes (recall that for each of its neighbouring jth bit nodes, $r_{ij} = 1$). Hence, $\mathbf{c}$ is a codeword if and only if the values of all the check nodes are zero.

Although the algorithm does not guarantee successful decoding (for example, it may get trapped in an infinite loop if there are short cycles in the Tanner graph), it has been shown empirically to perform well as a decoding algorithm.

The detailed Bit-Flipping Algorithm is presented in Algorithm 1 in the Appendix.

5.1.3 Sum Product Algorithm

Let $\mathbb{F} = \mathbb{F}_2$, $\mathbf{H}$ be the parity-check matrix, $\mathbf{X} = (x_0, x_1, \ldots, x_{n-1})$ be the original word and $\mathbf{C} = (c_0, c_1, \ldots, c_{n-1})$ be the word received.

For $j = 1, 2, \ldots, n$, define $L(X_j \mid N \text{ and } \mathbf{C}) = \ln\left(\frac{Pr[x_j=0|N \text{ and } \mathbf{C}]}{Pr[x_j=1|N \text{ and } \mathbf{C}]}\right)$, where N is the event that all m parity-check equations are satisfied by $\mathbf{X}$. For each jth bit node, define A_j to be the set of check nodes adjacent to the jth bit node. For each ith check node, define A_i to be the set of bit nodes adjacent to the ith check node.

In each iteration of the algorithm,

- For $j = 1, 2, \ldots, n$, a message M_{ij}, which is an estimate of $L(X_j \mid N \text{ and } \mathbf{C})$ is sent from the jth bit node to each of its adjacent check nodes $i \in A_j$.
- For $i = 1, 2, \ldots, m$, a message E_{ij}, which is an estimate for $\ln\left(\frac{Pr[N_i|X_j=0 \text{ and } \mathbf{C}]}{Pr[N_i|X_j=1 \text{ and } \mathbf{C}]}\right)$ is sent from the ith check node back to the jth bit node, for each $j \in B_i$.

The Sum product algorithm is a probabilistic decoding algorithm, which is employed to compute $L(X_j \mid N \text{ and } \mathbf{C})$.

The general steps are as follows, and the exact algorithm is presented in the Appendix in Algorithm 2. More mathematical details can also be found in the Appendix. Let N be the limit on the number of iterations.

Step 1: If $\mathbf{C}$ is a codeword, stop and decode $\mathbf{C}$ to $\mathbf{C}$. Else, set $t = 1$.

Step 2: For $j = 1, 2, \ldots, n$, compute $r_j = L(X_j \mid C_j = c_j)$ and send $M_{ij} = r_j$ to the ith check node for each $i \in A_j$.

Step 3: For $i = 1, 2, \ldots, m$, compute $E_{ij} = \ln\left(\frac{1+\prod_{t\in B_i\setminus\{j\}} \tanh(\frac{1}{2}M_{it})}{1-\prod_{t\in B_i\setminus\{j\}} \tanh(\frac{1}{2}M_{it})}\right)$ and send it to the jth bit node for each $j \in B_i$.

Step 4: For $j = 1, 2, \ldots, n$, compute $L_j = r_j + \sum_{s\in A_j} E_{sj}$. Set $y_j = \begin{cases} 0 \text{ if } L_j > 0 \\ 1 \text{ if } L_j < 0 \end{cases}$

If $\mathbf{y} = (y_0, y_1, \ldots, y_n)$ is a codeword, terminate and decode $\mathbf{C}$ to $\mathbf{y}$.

Step 5: If $t < N$, set $t = t + 1$.

Send $M_{ij} = r_j + \sum_{s\in A_j\setminus\{i\}} E_{sj}$ to the ith check node for each $i \in A_j$, then go to step 3. If $t = N$, terminate and output a decoding failure message for $\mathbf{C}$.

Remark 5.1.3. *In step 2, $L(X_j \mid C_j = c_j)$ is used as an initial estimate for $L(X_j \mid N$ and $\boldsymbol{C})$. Letting the probability of error be p, i.e.$p = Pr[C_j = 0 \mid X_j = 1] = Pr[C_j = 1 \mid X_j = 0]$, and under the assumption $Pr[X_j = 0] = Pr[X_j = 1]$,*

$$L(X_j \mid C_j = c_j) = \begin{cases} \ln(\frac{1-p}{p}) \text{ if } c_j = 0 \\ \ln(\frac{p}{1-p}) \text{ if } c_j = 1 \end{cases}$$

6. EXPERIMENTS

In Chapter 4, we outlined and described the CAKE scheme. The actual construction of the code, however, involves several parameters, such as the Hamming weight of $\mathbf{h}$, d_v, number of errors introduced in the code t and the value of r. As discussed in Section 4.3 of the same chapter, there are certain guidelines that may be followed to choose a suitable r and hence the polynomial ring used for the code, and the r values can be infinitely large as long they satisfy the required conditions.

All the experiments are run on a Windows 64-bit laptop machine. Due to computational limitations, we will let $r = 1019$. Note that this choice of r will give us an irreducible polynomial ring, and $\frac{r-1}{2}$ is also a Sophie Germain Prime.

In Section 4.2.4, we highlighted the importance of the parameter d_v, which determines the weight of the code. This value can neither be too large or too small, so as to get a balance between high decoding probability using the private key (usefulness of the code), and low decoding probability without knowledge of the private key (security of the code). Taking note of this, here, we will use empirical data to estimate suitable ranges for the parameter d_v, with varying t values and run a small-scale experiment on the security of the code.

6.1 Finding Suitable Ranges for d_v

In the first part of our experiments, we try to determine the possible ranges of d_v, while varying the values of t from 1 to 20. To do so, we will generate keys and encode generated messages in the same way as described in Section 4.2.1 and 4.2.2. Then, we will attempt to decode the messages using both the bit-flipping and sum product algorithm, and compare them against

the original message sent. The decoding rate out of 100 trials will be recorded. The decoding rate p_{decode} is defined as such:

$$p_{\text{decode}} = \frac{\text{number of correctly decoded messages}}{\text{total number of trials}}$$

Note that since there may be messages that are decoded by the algorithm but do not match the original message sent (incorrectly decoded), we will only count the correctly decoded messages. However, later we will also see that instances of incorrectly decoded messages are extremely rare.

In order to ensure that communicating parties in the key agreement scheme are able to establish a common secret key K, the decoding rate has to be sufficiently high. In other words, the decoding failure rate has to be very low. Hence, our approach towards finding suitable ranges for the parameter d_v, for each value of $t \in \{1, 2, \ldots, 20\}$ is as follows:

Step 1: Set $d_v^{\text{high}} = r, d_v^{\text{low}} = 1$ and fix t. For both decoding algorithms, fix $N = 20$. We explain the choice of N in 6.3.

Step 2: Run 100 trials of encoding and decoding using d_v^{high}. Record the decoding rate p_{decode}. If $p_{\text{decode}} < 0.9$ for both decoding algorithms, half d_v^{high} and repeat the trials, until $d_v^{\text{high}} = 1$ or $p_{\text{decode}} \geq 0.9$.

Step 3: Run 100 trials of encoding and decoding using d_v^{low}. Record the decoding rate p_{decode}. If $p_{\text{decode}} < 0.9$ for both decoding algorithms, double d_v^{low} and repeat the trials, until $d_v^{\text{low}} = 1$ or $p_{\text{decode}} \geq 0.9$.

Step 4: Output d_v^{high} and d_v^{low}.

In Tables 6.1 and 6.2, the decoding rate for one of the t values, $t = 5$ is shown. In addition, we also display the number of iterations and time taken for each of the decoding algorithms, taking into account only the correctly decoded messages. Note that while the value of $r = 1019$, the number of iterations taken on average for correctly decoded words is less than 2.

Table 6.3 shows the ranges of d_v for each t, for which the decoding rate is at least 0.9 for either the bit-flipping or sum product algorithm.

Tab. 6.1: Table of results for $r = 1019$, $t = 5$ using the Bit-Flipping Algorithm

d_v	**Correctly decoded**	**Incorrectly decoded**	**Failed to be decoded**	**Average number of iterations**	**Average time taken (s)**
1019	0	0	100	NA	NA
509	0	0	100	NA	NA
255	0	0	100	NA	NA
127	49	0	51	1.77551	0.47558
2	96	0	4	1.03125	0.00674
1	0	0	100	NA	NA

Tab. 6.2: Table of results for $r = 1019$, $t = 5$ using the Sum Product Algorithm

d_v	**Correctly decoded**	**Incorrectly decoded**	**Failed to be decoded**	**Average number of iterations**	**Average time taken (s)**
1019	0	0	100	NA	NA
509	0	0	100	NA	NA
255	2	0	98	3	5.97411
127	100	0	0	1.29	1.53364
2	97	0	3	1.05155	0.02897
1	0	0	100	NA	NA

Tab. 6.3: Ranges of d_v for which decoding rate is at least 0.9, for $r = 1019$ and $t = 1$ to 20

t	d_v^{low}	d_v^{high}	d_v^{low} **achieved by bit-flipping**	d_v^{low} **achieved by Sum Product algorithm**	d_v^{high} **achieved by bit-flipping**	d_v^{high} **achieved by sum product algorithm**
1	2	255	Y	Y	Y	Y
2	2	255	Y	Y	Y	Y
3	2	127	Y	Y	Y	Y
4	2	127	Y	Y	Y	Y
5	2	127	Y	Y	N	Y
6	2	127	Y	Y	N	Y
7	2	63	Y	Y	N	Y
8	2	63	Y	Y	N	Y
9	2	63	Y	Y	N	Y
10	2	63	Y	Y	N	Y
11	2	63	Y	Y	N	Y
12	2	63	Y	Y	N	Y
13	2	31	Y	Y	N	Y
14	2	31	Y	Y	N	Y
15	2	31	Y	Y	N	Y
16	2	31	N	Y	N	Y
17	2	31	N	Y	Y	Y
18	2	31	N	Y	Y	N
19	2	31	N	Y	N	Y
20	2	31	N	Y	N	Y

Based on the above empirical observations, the range of values d_v for the code to be a t-correcting code narrows down as t increases. This is not surprising, considering that the likelihood of having short cycles in the Tanner graph increases as d_v increases, and the Tanner-graph based decoding algorithms tend to perform less optimally with many short cycles.

Note that $n = n_0 r = 2 * 1019 = 2038$, hence $\sqrt{n \log n} \approx 124$. If the row weight of the code is $\sqrt{n \log n}$, then we have $d_v \approx 62$. Based on results in Table 6.3, for d_v in this range, the code achieves a relatively high decoding rate of at least 0.9, for $t = 1$ to 12.

The results above give a range for d_v^{low} and d_v^{high}. To investigate further, we can zoom into a specific range. Based on the ranges of d_v, and taking into consideration that we mainly focus on MDPC codes, we can look at more specific (odd) values in the range $d_v = 30$ to 64. Fixing $t = 15$, we get the results in Table 6.4 when we let $d_v \in [30, 64]$. From the empirical results, it is clear that the error-correcting capability decreases as d_v increases within the range, as expected. For $t = 15$, it would still be possible to let $d_v \approx 51$, based on the results.

Tab. 6.4: Table of results for $r = 1019$, $t = 15$, odd $d_v \in [30, 64]$

d_v	$p_{\mathbf{decode}}$ **(bit-flipping algorithm)**	**Average time taken (s)**	$p_{\mathbf{decode}}$ **(sum product algorithm)**	**Average time taken (s)**
31	0.36	0.13098	1	0.45109
33	0.3	0.13387	1	0.50379
35	0.33	0.15808	1	0.55868
37	0.17	0.15377	1	0.61879
39	0.12	0.17154	0.99	0.62393
41	0.09	0.18232	1	0.75836
43	0.05	0.20347	0.97	0.78343
45	0.05	0.24110	0.96	0.81582
47	0.05	0.21245	0.96	0.93493
49	0.02	0.21143	0.92	3.17920
51	0.01	0.23838	0.96	1.18821
53	0	NA	0.93	1.13766
55	0	NA	0.89	1.73540
57	0	NA	0.82	1.19643
59	0	NA	0.72	3.21050
61	0	NA	0.55	15.84148
63	0	NA	0.59	3.16606

6.2 Brute-force Decoding

In 5, we discussed two ways of decoding CAKE. The decoding typically requires possession of the private key, which is the parity-check matrix $\mathbf{H}$. Suppose an eavesdropper somehow manages to obtain the cryptogram $\mathbf{c}$, but has only the public key, the generator matrix $\mathbf{G}$. Besides achieving a high decoding probability using the private key, it is also important to ensure that any eavesdropper possessing the $\mathbf{c}$ and $\mathbf{G}$ will not be able to break the code easily.

To test this, we ran a computer program on a code of circulant matrix size $r = 1019$, as before, but using a brute-force method of decoding which makes use of the public key $\mathbf{G}$ to derive another parity-check matrix $\mathbf{H}'$. Then, we applied the bit-flipping and sum product algorithm on $\mathbf{H}'$. In this brute-force method, we do not assume possession of the private key $\mathbf{H}$.

This method of decoding uses the structure of $\mathbf{G}$. Since $\mathbf{G} = \begin{pmatrix} \mathbf{G}_0 & \mathbf{G}_1 \end{pmatrix}$, we have

$$\mathbf{G}_0^{-1} \begin{pmatrix} \mathbf{G}_0 & \mathbf{G}_1 \end{pmatrix} = \begin{pmatrix} \mathbf{I} & \mathbf{G}_0^{-1}\mathbf{G}_1 \end{pmatrix}$$

and after rearrangement and taking transpose of $\mathbf{G}_0^{-1}\mathbf{G}_1$ and multiplying it by a factor of -1, we have a new parity-check matrix

$$\mathbf{H}' = \begin{pmatrix} -(\mathbf{G}_0^{-1}\mathbf{G}_1)^T & \mathbf{I} \end{pmatrix}.$$

$\mathbf{H}'$ can then be used as the parity check matrix for decoding as before, using the bit-flipping and sum product algorithm. Note that we only need to compute the Hall polynomial $g_0^{-1}(x)$, rather than the whole inverse matrix $\mathbf{G}_0^{-1}$, due to the circulant matrix properties detailed in Chapter 2.

The program was run using a simple parameter choice of $r = 1019$, $n = 2r = 2038$, $dv \in \{61, 63, 65\} \in O(n \log n)$, $t = 5$ and the iteration limit for the decoding $N = 2038 = n$. As we expect the decoding to take considerable time, only 10 trials were run. Results are as shown in Table 6.5.

Even with a low number of errors $t = 5$, $p_{\text{decode}} \leq 0.2$. For the codes that could be decoded, more than 65 iterations were needed. Part of the bottleneck could be due to the time required to compute the polynomial $g_0^{-1}(x)$. Another possible contributing factor is the high row weight of $\mathbf{H}$, which was shown to be in the order of $\frac{1}{4}n \approx 500$. A higher row weight typically results in more short cycles, and hence lower decoding probability.

This is an optimistic result, as it shows that even with a low value of t relative to r, the

code takes a much longer time to decode by brute-force. With larger matrix sizes and larger values of possible t, or better parameter choices, the code is likely to be much more difficult to break.

Tab. 6.5: Table of results for $r = 1019$, $t = 5$, odd $d_v \in \{61, 63, 65\}$

d_v	**Correctly decoded out of 10 trials (bit-flipping algorithm)**	**Average number of iterations**	**Correctly decoded out of 10 trials (sum product algorithm)**	**Average number of iterations**
61	2	267.5	0	NA
63	2	67	0	NA
65	0	NA	0	NA

6.3 A Comparison between the two Decoding Algorithms

In our decoding experiments above, both the bit-flipping and sum product algorithms were used on the same code. Based on the empirical data, there are a few key observations. Firstly, for almost all the parameter settings, the sum product algorithm achieved a higher decoding rate compared to the bit-flipping algorithm. For example, for the parameter setting of $r = 1019$ and $t = 5$ in Tables 6.1 and 6.2, the sum product algorithm achieves a much higher decoding rate of 1, compared to the bit-flipping algorithm with only 0.49. However, secondly, the sum product algorithm also tends to take a longer time for decoding. Using the same example, the decoding rate is similar for both algorithms when $d_v = 2$, however, the sum product algorithm takes more than 4 times longer to decode the same code (on average), compared to the bit-flipping algorithm. This difference in time complexity can become very significant when the size of the code gets larger.

In practical applications, the order of the code is much larger, but in order for the protocol to be efficient, the decoding time has to be minimized as well. Hence, based on the two algorithms used, there is an inevitable trade-off between time taken and decoding rate.

Thirdly, the sum product seems to achieve a more stable performance, based on results in Section 6.1 and 6.2. As mentioned above, when the private key $\mathbf{H}$ is used, the algorithm tends to achieve a higher decoding rate than the bit-flipping algorithm. However, when we attempted to break the randomly-generated codes by brute force, the sum product algorithm achieves a zero decoding rate, which is the ideal case.

Another observation is that the number of iterations required for successful decoding is usually a low number relative to the order of the code, in the case whereby the code is correctly decoded. This number is usually less than 10, based on our empirical data. If this is also the case for larger code sizes, then the time required for decoding is not likely to grow too large, making it useful for key agreement in practical applications.

Last but not least, although these decoding algorithms typically do not guarantee 100% decoding success, from our empirical observations, the two decoding algorithms usually outputs either a correctly decoded message or a decoding failure, and cases where it outputs an incorrectly decoded message is very rare. This makes it suitable for decoding such codes, as communicating parties under the CAKE protocol will be able to achieve their purpose of agreement on a common secret key K. In addition, this is likely to be partly attributed to the invertibility of the circulant matrices used in CAKE as described in Remark 4.2.5, which allow for a bijective mapping between $\mathbf{c}$ and $\mathbf{m}$, so that the scenario whereby a codeword is mapped from two different $\mathbf{m}$ vectors does not occur.

7. FURTHER WORK

In this thesis, the CAKE scheme and a mathematical analysis of its feasibility was introduced. Several computer programmes were run to investigate the impact of different parameter choices on the decoding probability of the codes. Due to the computational limitations of a desktop computer, the feasible matrix size was limited to approximately 1000. With better computational power, larger matrix sizes can be tested, and this is likely to give a clearer perspective on the actual performance of much larger code sizes used in practice.

From our empirical data, the method of computing $\mathbf{G}_0^{-1}$ and multiplying it to the generator matrix $\mathbf{G}$ does not work well as a way of decoding. This supports the security of the scheme, since the codes would not be easily breakable using merely the public key, and hence would not be vulnerable to attacks by eavesdroppers. Yet, this does not necessarily mean that the same decoding method certainly does not work for a computer with higher computational power. Hence, this will also serve as an area for future work, and will require more computational resources.

8. CONCLUSION

This thesis introduced the CAKE scheme, based on [1], from a mathematical perspective. The CAKE scheme has several advantages compared to some other code-based schemes, in terms of time and space complexity, which makes it a potential post-quantum cryptography candidate for key agreement. In this thesis, we proved a theorem which provides for a way to check and generate circulant matrix sizes for the codes, based on the scheme's parameter constraints in the original paper. Based on the results, a program was also run to generate possible values for the matrix size. Hence, this reduces the user's effort on finding a suitable ring for the scheme.

Next, two decoding algorithms, namely the bit-flipping and sum product algorithm, were introduced. These two algorithms are commonly used for decoding, and the bit-flipping algorithm was proposed as the decoding algorithm for MDPC codes described in [8]. The two algorithms are non-deterministic, and the decoding procedure has an associated decoding probability. Later in Chapter 6, we outline the results of several experiments run using these decoding algorithms with varying parameter choices, while observing the decoding probability.

From the empirical discussion, the parameter setting of $d_v = \sqrt{n \log n}$ for the proposed CAKE codes gives a fair balance between high decoding probability using the private key and low decoding probability **without** knowledge of the private key. Note that the larger row weights in MDPC codes makes it **more secure** than Low Density Parity Check (LDPC) codes [8]. Although this also causes the error-correcting capability of CAKE to be lower relative to LDPC codes, from our empirical results, a **fair level of errors** can still be corrected for CAKE while keeping decoding probability above a certain threshold. Hence, such a parameter setting allows for a balance between security and usefulness of the code, based on the discussion in Section 4.2.4.

A comparison between the two decoding algorithms was also presented. Based on our empirical results, the sum product algorithm tends to have a more stable performance than the bit-flipping algorithm, however the downside is that it also takes a longer decoding time. Hence, the decoder will have to find a balance between error-correcting capability and time complexity.

APPENDIX

A. APPENDIX

In the bit-flipping algorithm below, v_j, w_i represents the values of bit nodes and check nodes respectively. y_j represents the number of parity-check equations containing the jth bit, which are violated. N is an upper bound on the number of iterations allowed in the algorithm.

Algorithm 1: Bit-flipping Algorithm

input : **H**, **c**
output: decoded codeword or $\perp$

Function *Bit-Flipping Algorithm(**H**, **c**)*
  Set up Tanner graph
  Set $t = 0$
  for $j = 0, 1, \ldots, n-1$ **do**
    $v_j \leftarrow c_j$
  end
  while $t < N$ **do**
    for $i = 0, 1, \ldots, m-1$ **do**
      $w_i \leftarrow \sum_{k=0}^{n-1} v_k \pmod 2$
    end
    if $w_i = 0$ *for all* i
      return **v**
    else
      for $j = 0, 1, \ldots, n-1$ **do**
        $y_j \leftarrow \sum_{k=0}^{m-1} w_k$
        if $y_j > m/2$
          flip value of v_j
        else
          do nothing
      end
    $t = t + 1$
  end

Proposition A.0.1. *Let a Tanner graph be given. We randomly assign 0 and 1 to the bit nodes such that their values are stochastically independent from one another. Suppose z_t is the value of the tth bit node and $Pr[z_t = 1] = p_t$. Let N_i be the event that the ith parity-check equation is satisfied, and B_i be the set of bit nodes adjacent to the ith check node.*

Assuming that the jth bit node is adjacent to the ith check node, i.e. $j \in B_i$, we have $ln\left(\frac{Pr[N_i|z_j=0]}{Pr[N_i|z_j=1]}\right) = ln\left(\frac{1+\prod_{t\in B_i\setminus\{j\}}\tanh(\frac{1}{2}m_t)}{1-\prod_{t\in B_i\setminus\{j\}}\tanh(\frac{1}{2}m_t)}\right)$, *where* $m_t = \ln\left(\frac{Pr[z_t=0]}{Pr[z_t=1]}\right)$.

Proof. (i) Define $f_C(x) = \prod_{t\in C}[(1-p_t)+p_t x]$. Observe that the coefficient of x^k in $f_C(x)$ is equal to $a_k = \sum_{\substack{t\subseteq C\\|T|=k}}\Big(\prod_{t\in T}p_t\Big)\Big(\prod_{t\in C\setminus T}(1-p_t)\Big)$. On the other hand, the probability that exactly k bit nodes in C are assigned the value 1 is equal to $\sum_{\substack{t\subseteq C\\|T|=k}} Pr[z_t = 1 \text{ if } t \in T, \text{ and } z_t = 0 \text{ if } t \in C\setminus T] = a_k$, since bit node values are stochastically independent.

(ii) Note that the ith parity check equation is satisfied (event N_i) iff $\sum_{t\in B_i} z_t \equiv 0 \pmod 2$. If $z_j = 1$, then $\sum_{t\in B_i\setminus\{j\}} z_t \equiv 1 \pmod 2$, i.e. there are an odd number of bit nodes adjacent to the ith check node having value 1. Hence, by (i),

$$\begin{aligned} Pr[N_i \mid z_j = 1] &= \sum_{k \text{ is odd}} \text{coefficient of } x^k \text{ in } f_{B_i\setminus\{j\}}(x) \\ &= \frac{1}{2}[f_{B_i\setminus\{j\}}(1) - f_{B_i\setminus\{j\}}(-1)] \\ &= \frac{1}{2}\Big(1 - \prod_{t\in B_i\setminus\{j\}}(1-2p_t)\Big) \\ &= \frac{1}{2} - \frac{1}{2}\prod_{t\in B_i\setminus\{j\}}(1-2p_t) \end{aligned}$$

Similarly, $Pr[N_i \mid z_j = 0] = \frac{1}{2} + \frac{1}{2}\prod_{t\in B_i\setminus\{j\}}(1-2p_t)$.

Since $m_t = ln\left(\frac{Pr[z_t=0]}{Pr[z_t=1]}\right) = ln\left(\frac{1-p_t}{p_t}\right)$, we have

$$\tanh(\frac{1}{2}m_t) = \frac{e^{m_t}-1}{e^{m_t}+1} = \frac{\frac{1-p_t}{p_t}-1}{\frac{1-p_t}{p_t}+1} = 1-2p_t \qquad (*)$$

$$\implies ln\left(\frac{Pr[N_i \mid z_t = 0]}{Pr[N_i \mid z_t = 1]}\right) = ln\left(\frac{1+\prod_{t\in B_i\setminus\{j\}}\tanh(\frac{1}{2}m_t)}{1-\prod_{t\in B_i\setminus\{j\}}\tanh(\frac{1}{2}m_t)}\right)$$

where the last equality holds by (ii) and $(*)$. □

Proposition A.0.2. *Adopting the notations and definitions in Proposition A.0.1, suppose the Tanner graph has no 4-cycle. Assume that given z_j (for the jth bit node), the events N_i are conditionally independent. Let A_j be the set of check nodes adjacent to the jth bit node, and N be the event that all parity-check equations are satisfied. Then,* $\ln\left(\frac{Pr[z_j=0|N]}{Pr[z_j=1|N]}\right) \approx$

$ln\left(\frac{Pr[z_j=0]}{Pr[z_j=1]}\right) + \sum_{i\in A_j} ln\left(\frac{Pr[N_i|z_j=0]}{Pr[N_i|z_j=1]}\right)$.

Proof. Note that if $i \notin A_j$, then $j \notin B_i$. This means that values of z_j and $\sum_{t\in B_i} z_t$ are stochastically independent, so $\frac{Pr[N_i|z_j=0]}{Pr[N_i|z_j=1]} = \frac{Pr[N_i]}{Pr[N_i]} = 1$. Thus,

$$\begin{aligned}
\frac{Pr[z_j = 0 \mid N]}{Pr[z_j = 1 \mid N]} &= \frac{Pr[z_j = 0]Pr[N \mid z_j = 0]/Pr[N]}{Pr[z_j = 1]Pr[N \mid z_j = 1]/Pr[N]} \\
&= \frac{Pr[z_j = 0]Pr[N \mid z_j = 0]}{Pr[z_j = 1]Pr[N \mid z_j = 1]} \\
&= \frac{Pr[z_j = 0]}{Pr[z_j = 1]} \prod_i \frac{Pr[N_i \mid z_j = 0]}{Pr[N_i \mid z_j = 1]} \\
&= \frac{Pr[z_j = 0]}{Pr[z_j = 1]} \prod_{A_j} \frac{Pr[N_i \mid z_j = 0]}{Pr[N_i \mid z_j = 1]} \\
&= \frac{Pr[z_j = 0]}{Pr[z_j = 1]} + \sum_{A_j} \ln\left(\frac{Pr[N_i \mid z_j = 0]}{Pr[N_i \mid z_j = 1]}\right)
\end{aligned}$$

□

Remark A.0.3. *Using Proposition A.0.1 and A.0.2, and letting* $\boldsymbol{c}$ *be the codeword sent,* $p_t = Pr[c_t = 1 \mid \boldsymbol{y}$ *is received* $]$*, and events* N_i *stochastically independent, this gives us the updates used in the sum-product decoding algorithm in Section 5.1.3.*

Algorithm 2: Sum Product Algorithm

input : $\mathbf{H}$, $\mathbf{C}$
output: decoded codeword or $\perp$

Function *Sum Product Algorithm(**H**, **C**)*
  Set up Tanner graph
  if ***C** is a codeword*
    return $\mathbf{C}$
  else
    set $t = 0$
    for $j = 0, 1, \ldots, n-1$ **do**
      $r_j \leftarrow L(X_j \mid C_j = c_j)$
      for each $i \in A_j$ **do**
        $M_{ij} \leftarrow r_j$
      end
    end
    while $t < N$ **do**
      for $i = 0, 1, \ldots, m-1$ **do**
        for each $j \in B_i$ **do**
          $E_{ij} \leftarrow \ln\left(\frac{1+\prod_{t\in B_i\setminus\{j\}} \tanh(\frac{1}{2}M_{it})}{1-\prod_{t\in B_i\setminus\{j\}} \tanh(\frac{1}{2}M_{it})}\right)$
          Send E_{ij} to the jth bit node
        end
      end
      for $j = 0, 1, \ldots, n-1$ **do**
        $L_j \leftarrow r_j + \sum_{s\in A_j} E_{sj}$
        if $L_j > 0$
          $x_j \leftarrow 0$
        elif $L_j < 0$
          $x_j \leftarrow 1$
      end
      if ***C** is a codeword*
        return $\mathbf{C}$
      else
        set $t = t + 1$
        for each $i \in A_j$ **do**
          $M_{ij} = r_j + \sum_{s\in A_j\setminus\{i\}} E_{sj}$
          Send M_{ij} to the ith check node
        end
    end

BIBLIOGRAPHY

[1] P. S. Barreto, S. Gueron, T. Gueneysu, R. Misoczki, E. Persichetti, N. Sendrier, and J.-P. Tillich, "CAKE: Code-Based Algorithm for Key Encapsulation," in *IMA International Conference on Cryptography and Coding*. Springer, 2017, pp. 207–226.

[2] D. R. Stinson, *Cryptography: theory and practice*. CRC press, 2005.

[3] Digicert, "Check our Numbers - The Math behind Estimations to break a 2048-bit Certificate," [Online; accessed: 2018-09-02]. [Online]. Available: https://www.digicert.com/TimeTravel/math.htm

[4] P. Prajapati, N. Patel, R. Macwan, N. Kachhiya, and P. Shah, "Comparative analysis of DES, AES, RSA encryption algorithms," *International Journal of Engineering and Management Research*, vol. 4, no. 1, pp. 292–294, 2014.

[5] E. Barker and Q. Dang, "NIST Special Publication 800–57 Part 1, Revision 4," 2016.

[6] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.

[7] S. L. Ma, "Coding and Cryptography," Lecture notes, 2018.

[8] R. Misoczki, J.-P. Tillich, N. Sendrier, and P. S. Barreto, "MDPC-McEliece: New McEliece variants from moderate density parity-check codes," in *Information Theory Proceedings (ISIT), 2013 IEEE International Symposium on*. IEEE, 2013, pp. 2069–2073.

[9] E. Berlekamp, R. McEliece, and H. Van Tilborg, "On the inherent intractability of certain coding problems (Corresp.)," *IEEE Transactions on Information Theory*, vol. 24, no. 3, pp. 384–386, 1978.

[10] J. H. Silverman, *A friendly introduction to number theory*. Pearson Education, Inc., 2012.