PALOMA: Binary Separable Goppa-based KEM[1]

Dong-Chan Kim Chang-Yeol Jeon Yeonghyo Kim Minji Kim

Future cryptography Design Lab., Kookmin University

dckim@kookmin.ac.kr

Updated: October 31, 2022

[1]This work is submitted to 'Korean Post-Quantum Cryptography Competition' (www.kpqc.or.kr).

Contents

List of Figures

List of Algorithms

List of Tables

Chapter 1

Introduction

1.1 Design Rationale

PALOMA is a code-based key encapsulation mechanism that has the following features.

(1) Trapdoor based on SDP(syndrome decoding problem)

(2) IND-CCA2-secure KEM(Key Encapsulation Mechanism) based on FO(Fujisaki-Okamoto) transformation

(3) Parameters supporting 128/192/256-bit security strength

1.1.1 Trapdoor

1.1.1.1 Syndrome Decoding Problem

SDP is a problem finding the preimage vector with a specific Hamming weight for a given random binary parity-check matrix and a syndrome. In 1978, SDP was proven to be NP-hard because it is equivalent to the 3-dimensional matching problem[9, 3]. McEliece and Niederreiter cryptosystems are designed with the trapdoor based on SDP[15, 17]. However, because the public key of a SDP-based trapdoor is a random-looking matrix, the public key is larger than that of other ciphers. Therefore, there have been attempts to reduce the size of a public key through cryptographic design using SDP-variant, such as rank metric-based SDP and quasi-cyclic code-based SDP. However, SDP-variants assume the problem's difficulty because one cannot guarantee the NP-hard property of SDP.

Post Quantum Cryptography is not a cryptographic scheme that provides additional functionality but an alternative to the current cryptosystem against quantum attacks. Therefore, we design PALOMA based on SDP with a conservative perspective because SDP is NP-hard and it is judged that the analysis method is sufficiently mature.

1.1.1.2 Niederreiter-type Code Scrambling.

In general, code-based cryptographic schemes use the information of a scrambled code $\widehat{\mathcal{C}}$, which is an equivalent code of the base code $\mathcal{C}$, as a public key, and the decoding information for $\mathcal{C}$ as a private key. Similar to the Niederreiter cryptosystem, PALOMA uses the parity check matrix $\widehat{\mathbf{H}}$ of a scrambled code $\widehat{\mathcal{C}}$ that is defined by $\mathbf{SHP}$ where $\mathbf{H}$ is the parity-check matrix of $\mathcal{C}$, $\mathbf{S}$ and $\mathbf{P}$ are an invertible matrix and a permutation matrix, respectively. $\mathbf{P}$ is randomly chosen. However, to reduce the size of a public key, the invertible matrix $\mathbf{S}$ is obtained from the reduced row echelon form procedure applying to $\mathbf{HP}$, so that $\widehat{\mathbf{H}}$ is the form of systematic, i.e., $\widehat{\mathbf{H}} = [\mathbf{I} \mid \mathbf{M}]$. PALOMA uses the submatrix $\mathbf{M}$ of $\widehat{\mathbf{H}}$ as a public key like Classic McEliece[4]. Figure 1.1 shows the trapdoor framework of PALOMA.

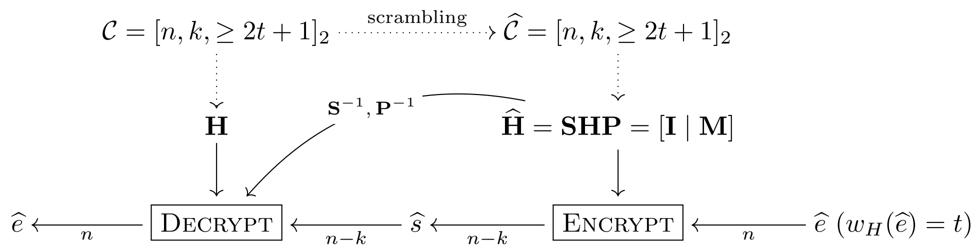


Figure 1.1: PALOMA: Trapdoor Framework

The Niederreiter cryptosystem needs to convert messages into vectors with a specific Hamming weight for decoding. This conversion performs a large amount of computation, which significantly affects encryption/decryption performance. However, PALOMA is designed to work without this conversion.

1.1.1.3 Binary Separable Goppa Code.

There are no critical attacks on cryptographic schemes based on an SDP defined with a binary separable Goppa code[7], for example, McEliece cryptosystem, which is the first code-based cipher[15]. Many researchers have tried to design code-based ciphers using various codes such as GRS and RM to increase efficiency in terms of public key size and decryption speed, but most of them have been attacked due to their structural properties, and the rest still need more rigid security proof[20, 16]. Therefore, PALOMA chooses a binary separable Goppa code that has no attack even though it has been studied for a long time with a conservative perspective.

A binary separable Goppa code $\mathcal{C} = [n, k, \geq 2t+1]_2$ is defined with a support set L and a Goppa polynomial $g(X)$ that is separable. Because every irreducible polynomial is separable, an irreducible polynomial is chosen as a Goppa polynomial, in general. However, since the algorithms generating irreducible polynomials are probabilistic, i.e., non-constant time. PALOMA defines a support set and a Goppa polynomial with randomly chosen $n+t$ elements of $\mathbb{F}_{2^{13}}$ as follows:

$$[\alpha_0, \alpha_1, \ldots, \alpha_{2^m-1}] \leftarrow \textsc{Shuffle}(\mathbb{F}_{2^m}), \quad L \leftarrow [\alpha_0, \alpha_1, \ldots, \alpha_{n-1}], \quad g(X) \leftarrow \prod_{j=n}^{n+t-1} (X - \alpha_j).$$

After shuffling of all $\mathbb{F}_{2^m}$ elements, the set of the first n elements is defined as a support set and the next t elements are the root of a Goppa polynomial with degree t. Note that $g(X)$ is separable but not irreducible in $\mathbb{F}_{2^{13}}[X]$. Thus, PALOMA generates a binary separable Goppa code efficiently within constant time.

Patterson and Berlekamp-Massey are decoding algorithms of a binary separable Goppa[18, 2, 12]. Patterson seem to be better than Berlekamp-Massey in terms of speed performance, however, it operates when a Goppa polynomial $g(X)$ is irreducible. So, PALOMA adapts the extended Patterson decoding to deal with a non-irreducible Goppa polynomial[5].

1.1.2 KEM structure

In IND-CCA2 security game(INDistinguishability against Adaptive Chosen-Ciphertext Attack) for KEM $= (\textsc{GenKeyPair}, \textsc{Encap}, \textsc{Decap})$, the challenger sends a challenge (key, ciphertext) pair to the attacker, and the attacker guesses if the pair is right or not. ("right" means the pair (key, ciphertext) is an output of Encap) Here it is allowed for the attacker to query the Decap oracle except for the challenge. We say KEM is IND-CCA2-secure when the winning probability of any polynomial time attackers in IND-CCA2 game is negligible. Figure 1.2 shows the IND-CCA2 game.

Challenger		Adversary $\mathcal{A}$
$(pk, sk) \leftarrow \textsc{GenKeyPair}(1^n)$	$\xrightarrow{O^{\textsc{Decap}(sk;*)}}$	$\mathcal{A}$ queries $O^{\textsc{Decap}(sk;*)}$
$(k, c) \leftarrow \textsc{Encap}(pk)$		
$b \xleftarrow{\$} \{0,1\}$		
If $b = 0$, then $k \xleftarrow{\$} \{0,1\}^l$	$\xrightarrow{(k,c)}$	$\mathcal{A}$ queries $O^{\textsc{Decap}(sk;*)}$ except c
If $b = b'$, then $\mathcal{A}$ wins, else $\mathcal{A}$ loses.	$\xleftarrow{b'}$	If $\mathcal{A}$ thinks that k is right, then $b' \leftarrow 1$, else $b' \leftarrow 0$

Figure 1.2: Security Game for IND-CCA2 KEM

In general, IND-CCA2-secure schemes are constructed with OW-CPA-secure trapdoors and hash functions that are considered random oracles. FO transformation is a representative IND-CCA2-secure scheme design method, which is also proven to be IND-CCA2-secure in QROM(Quantum Random Oracle Model)[6, 8, 22]. PALOMA guarantees IND-CCA2-secure since it is designed by the FO-variant transformation $\mathsf{KEM}^{\not\perp}$, introduced in [8].

1.1.3 Parameter Sets

The security of PALOMA is evaluated by the number of bit computations of generic attacks to SDP because there are no known attacks on binary separable Goppa codes. ISD(Information Set Decoding) is the most powerful generic attack of an SDP. The complexity of ISD has been improved by changing the specific conditions for the information set[19, 10, 11, 21, 13, 1, 14] and birthday-type search algorithms. PALOMA evaluated the security strength level in computational complexity for the most effective attack.

PALOMA provides three parameter sets: PALOMA-128, PALOMA-192 and PALOMA-256, which are 128-bit, 192-bit, and 256-bit security strength level, respectively. Each parameter was selected as a parameter satisfying the following conditions regarding implementation efficiency.

- (i) Binary separable Goppa codes are defined in $\mathbb{F}_{2^{13}}$
- (ii) $n \equiv k \equiv t \equiv 0 \bmod 64$
- (iii) $n + t \leq 2^{13}$
- (iv) $k/n > 0.7$

1.2 Advantages and Limitations

PALOMA is a KEM designed by combining an NP-hard SDP-based trapdoor using binary separable Goppa codes and FO transformation that guarantees IND-CCA2-secure in ROM and QROM both, which are strongly considered safe in cryptographic communities. Therefore, we believe that PALOMA provides sufficiently reliable security in classical computers and quantum computers.

Since PALOMA is an SDP-based trapdoor, the public key size is essentially over 300 KB. In addition, the generation of a public key that is the parity check matrix of the scrambled Goppa code is relatively slow compared to other post-quantum ciphers. So, in the server-client protocol, generating ephemeral keys can burden the server. Therefore, PALOMA is suitable for server-to-client protocols that use static keys and client-to-client protocols, such as E2EE.

Chapter 2

Mathematical Background

In this chapter, we introduce the mathematical background needed to figure out the operating principles of PALOMA.

2.1 Syndrome Decoding Problem

2.1.1 Binary Linear Codes

A k-dimensional binary linear code $\mathcal{C}$ of length n defined in a binary finite field $\mathbb{F}_2$ is a k-dimension subspace of the n-dimensional vector space $\mathbb{F}_2^n$. It means that $\mathcal{C}$ is the solution space of the following $n-k$ linear equations.

$$\begin{array}{rcrcccrcl} h_{0,0}X_0 & + & h_{0,1}X_1 & + & \cdots & + & h_{0,n-1}X_{n-1} & = & 0, \\ h_{1,0}X_0 & + & h_{1,1}X_1 & + & \cdots & + & h_{1,n-1}X_{n-1} & = & 0, \\ \vdots & & & & & & & & \\ h_{n-k-1,0}X_0 & + & h_{n-k-1,1}X_1 & + & \cdots & + & h_{n-k-1,n-1}X_{n-1} & = & 0. \end{array}$$

Therefore, a binary linear code $\mathcal{C}$ can be expressed as follows.

$$\mathcal{C} = \{c \in \mathbb{F}_2^n : \mathbf{H}c = 0^{n-k}\},$$

where 0^{n-k} is a zero vector in $\mathbb{F}_2^{n-k}$ and

$$\mathbf{H} = [h_{i,j}] := \begin{pmatrix} h_{0,0} & h_{0,1} & \cdots & h_{0,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ h_{n-k-1,0} & h_{n-k-1,1} & \cdots & h_{n-k-1,n-1} \end{pmatrix} \in \mathbb{F}_2^{(n-k)\times n}.$$

Note that all vectors are considered as column vectors in this paper. The vector $c \in \mathcal{C}$ and the matrix $\mathbf{H}$ are called a codeword and a parity check matrix of $\mathcal{C}$, respectively.

2.1.2 Syndrome Decoding Problem

For a vector $r \in \mathbb{F}_2^n$, $\mathbf{H}r \in \mathbb{F}_2^{n-k}$ is called the syndrome of r. If a syndrome is 0^{n-k}, the vector r is the codeword of $\mathcal{C}$. For any codeword $c \in \mathcal{C}$ and an arbitrary vector $e \in \mathbb{F}_2^n$, the vector $r = c + e$

satisfies the following.

$$\mathbf{H}r = \mathbf{H}(c+e) = \mathbf{H}c + \mathbf{H}e = \mathbf{H}e.$$

SDP is the problem of finding a preimage vector of a syndrome that has a specific Hamming weight. The formal definition of SDP is as follows:

Definition 2.1.1 (Syndrome Decoding Problem, SDP)**.** Given a parity check matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$ of a random binary linear code $\mathcal{C} = [n,k]_2$, a syndrome $s \in \mathbb{F}_2^{n-k}$ and $w \in \{1,2,\ldots,n\}$, find the vector $e \in \mathbb{F}_2^{(n-k)\times n}$ that satisfies the following two conditions.

$$\mathbf{H}e = s \quad \text{and} \quad w_H(e) = w.$$

SDP is proven as an NP-hard problem because it is equivalent to the 3-dimensional matching problem in 1978[9, 3].

2.1.2.1 Number of Roots of SDP.

Hamming weight $w_H(v)$ of a vector $v = (v_0,\ldots,v_{n-1}) \in \mathbb{F}_2^n$ is defined as $|\{j : v_j \neq 0\}|$. Hamming distance $d_H(u,v)$ of the two vectors $u, v \in \mathbb{F}_2^n$ is defined as $w_H(u+v)$. Assume that there are two distinct vectors $v_1, v_2 \in \mathbb{F}_2^n$ with Hamming weight of $\lfloor \frac{d-1}{2} \rfloor$ having same syndrome where d is the minimum Hamming distance of the linear code $\mathcal{C}$, i.e., $d(= \min_{c\in\mathcal{C}\setminus\{0^n\}} w_H(c))$. Since $\mathbf{H}(v_1+v_2) = 0^{n-k}$, it becomes $v_1 + v_2 \in \mathcal{C}$. However, since the minimum distance of $\mathcal{C}$ is d, the following contradiction occurs.

$$d \leq |\mathrm{supp}\,(v_1+v_2)| \leq |\mathrm{supp}\,(v_1)| + |\mathrm{supp}\,(v_2)| \leq 2\left\lfloor \frac{d-1}{2} \right\rfloor \leq d-1,$$

where $\mathrm{supp}\,(v) := \{j : v_j \neq 0\}$. Therefore, the preimage vector with Hamming weight less than equal to $\lfloor \frac{d-1}{2} \rfloor$ is unique. Generally, in SDP-based schemes, the Hamming weight condition w of SDP is set to $\lfloor \frac{d-1}{2} \rfloor$ for the uniqueness of root and root candidates more that 2^{256}.

2.2 Binary Separable Goppa Code

Binary separable Goppa codes are special cases of algebraic-geometric codes proposed by V. D. Goppa in 1970[7]. Many code-based ciphers, such as McEliece and Classic McEliece, use it as the base codes. The formal definition of a binary separable Goppa code over $\mathbb{F}_2$ is as follows.

Definition 2.2.1 (Binary Separable Goppa code)**.** For a set of distinct $n(\leq 2^m)$ elements $L = [\alpha_0, \alpha_1, \ldots, \alpha_{n-1}]$ of $\mathbb{F}_{2^m}$ and a separable polynomial $g(X) \in \mathbb{F}_{2^m}[X]$ of degree t that all elements of L are not roots of $g(X)$, i.e., $g(\alpha) \neq 0$ for all $\alpha \in L$, a binary separable Goppa code of length n over $\mathbb{F}_2$ is the subspace $\mathcal{C}_{L,g}$ of $\mathbb{F}_2^n$ defined by

$$\mathcal{C}_{L,g} := \left\{ (c_0,\ldots,c_{n-1}) \in \mathbb{F}_2^n : \sum_{j=0}^{n-1} c_j (X-\alpha_j)^{-1} \equiv 0 \pmod{g(X)} \right\},$$

where $(X-\alpha)^{-1}$ is the polynomial of degree $t-1$ satisfying the following.

$$(X-\alpha)^{-1}(X-\alpha) \equiv 1 \pmod{g(X)}.$$

L and $g(X)$ are called a support set and a Goppa polynomial, respectively. $\mathcal{C}_{L,g}$ is called a binary irreducible Goppa code when $g(X)$ is an irreducible polynomial in $\mathbb{F}_{2^m}[X]$. The dimension k and the minimum Hamming distance d of $\mathcal{C}_{L,g}$ satisfy the following inequalities.

$$k \geq n - mt, \quad d \geq 2t+1.$$

PALOMA set the dimension k of $\mathcal{C}_{L,g}$ to $n-mt$ and the Hamming weight condition of the SDP to t for uniqueness of root.

2.2.1 Parity-check Matrix

The parity check matrix $\mathbf{H}$ of $\mathcal{C}_{L,g}$ is defined with each coefficient of the polynomial $(X-\alpha_j)^{-1}$ with degree $t-1$, and $\mathbf{H}$ can be decomposed into the product of the following matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$.

$$\mathbf{H} = \mathbf{ABC} \in \mathbb{F}_{2^m}^{t\times n},$$

where

$$\mathbf{A} := \begin{pmatrix} g_1 & g_2 & \cdots & g_t \\ g_2 & g_3 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ g_t & 0 & \cdots & 0 \end{pmatrix} \in \mathbb{F}_{2^m}^{t\times t}, \quad \mathbf{B} := \begin{pmatrix} \alpha_0^0 & \alpha_1^0 & \cdots & \alpha_{n-1}^0 \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_0^{t-2} & \alpha_1^{t-2} & \cdots & \alpha_{n-1}^{t-2} \\ \alpha_0^{t-1} & \alpha_1^{t-1} & \cdots & \alpha_{n-1}^{t-1} \end{pmatrix} \in \mathbb{F}_{2^m}^{t\times n},$$
$$\mathbf{C} := \begin{pmatrix} g(\alpha_0)^{-1} & 0 & \cdots & 0 \\ 0 & g(\alpha_1)^{-1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & g(\alpha_{n-1})^{-1} \end{pmatrix} \in \mathbb{F}_{2^m}^{n\times n}. \tag{2.1}$$

Since the matrix $\mathbf{A}$ is invertible ($g_t \neq 0$), $\mathbf{BC}$ is another parity check matrix of $\mathcal{C}_{L,g}$. Classic McEliece uses $\mathbf{BC}$ as a parity check matrix.

2.2.2 Extended Patterson Decoding for Binary Separable Goppa code

Patterson decoding is the algorithm for binary irreducible Goppa codes, not separable Goppa code. However, it can be extended for binary separable Goppa codes[18, 5]. Given a syndrome vector s, the extended Patterson decoding procedure to find the preimage vector e of s with $w_H(e) = t$ is as follows. (Note that preimage vector is called an error vector in coding theory)

Step 1. Convert the syndrome vector $s \in \mathbb{F}_2^{n-k}$ into the syndrome polynomial $s(X) \in \mathbb{F}_{2^m}[X]$.

Step 2. Derive the key equation for finding the error locator polynomial $\sigma(X) \in \mathbb{F}_{2^m}[X]$.

Step 3. Solve the key equation using the extended Euclidean algorithm.

Step 4. Calculate $\sigma(X)$ using a root of the key equation.

Step 5. Find all roots of $\sigma(X)$ and compute the preimage vector $e \in \mathbb{F}_2^n$. At this time, in order to have resistance against timing attacks, we use exhaustive search.

In the above procedure, the error locator polynomial $\sigma(X)$ is

$$\sigma(X) := \prod_{j \in E}(X - \alpha_j) \in \mathbb{F}_{2^m}[X] \text{ where } E = \{i \in [n] : e_i \neq 0\}.$$

$\sigma(X)$ satisfies the following identity.

$$\sigma(X)s(X) \equiv \sigma'(X) \pmod{g(X)}. \tag{2.2}$$

Note that since the number of errors is t, $\sigma(X)$ that satisfies (2.2) is unique. In $\mathbb{F}_{2^m}[X]$, all polynomials $f(X)$ has polynomials $a(X)$ and $b(X)$ such that

$$f(X) = a(X)^2 + b(X)^2 X \text{ where } \deg(a) \leq \left\lfloor \frac{t}{2} \right\rfloor, \ \deg(b) \leq \left\lfloor \frac{t-1}{2} \right\rfloor.$$

Thus, if $\sigma(X) = a(X)^2 + b(X)^2 X$, (2.2) can be transformed as follows.

$$b(X)^2(1 + Xs(X)) \equiv a(X)^2 s(X) \pmod{g(X)}. \tag{2.3}$$

When $g(X)$ is irreducible, $s^{-1}(X)$ and $\sqrt{s^{-1}(X) + X}$ exist in modulo $g(X)$. Patterson decoding uses the extended Euclidean algorithm to find solutions $a(X)$ and $b(X)$ of the following key equation to generate the error locator polynomial $\sigma(X)$.

$$b(X)\sqrt{(s^{-1}(X) + X)} \equiv a(X) \pmod{g(X)}, \quad \deg(a) \leq \left\lfloor \frac{t}{2} \right\rfloor, \ \deg(b) \leq \left\lfloor \frac{t-1}{2} \right\rfloor.$$

However, if $g(X)$ is separable, the existence of $s^{-1}(X)$ cannot be guaranteed because $g(X)$ and $s(X)$ are unlikely to be relatively prime. We define

$$s^*(X) := 1 + Xs(X), \quad g_1(X) := \gcd(g(X), s(X)), \quad g_2(X) := \gcd(g(X), s^*(X)).$$

Since $\gcd(s(X), s^*(X)) = \gcd(s(X), s^*(X) \bmod s(X)) = \gcd(s(X), 1) \in \mathbb{F}_{2^m} \setminus \{0\}$, we know

$$\begin{aligned} g \mid b^2 s^* + a^2 s \quad &\Rightarrow \quad g_1 \mid b^2 s^* + a^2 s \quad \Rightarrow \quad g_1 \mid b^2 s^* \quad \Rightarrow \quad g_1 \mid b^2 \quad \Rightarrow \quad g_1 \mid b, \\ g \mid b^2 s^* + a^2 s \quad &\Rightarrow \quad g_2 \mid b^2 s^* + a^2 s \quad \Rightarrow \quad g_2 \mid a^2 s \quad \Rightarrow \quad g_2 \mid a^2 \quad \Rightarrow \quad g_2 \mid a. \end{aligned}$$

Therefore, the following polynomial can be defined.

$$b_1(X) := \frac{b(X)}{g_1(X)}, \quad a_2(X) := \frac{a(X)}{g_2(X)}, \quad g_{12}(X) := \frac{g(X)}{g_1(X)g_2(X)},$$
$$s_2^*(X) := \frac{s^*(X)}{g_2(X)}, \quad s_1(X) := \frac{s(X)}{g_1(X)}.$$

(2.3) can be expressed as follows.

$$b(X)^2 s^*(X) \equiv a(X)^2 s(X) \pmod{g(X)}$$

$$\Rightarrow \quad b_1^2(X)g_1^2(X)s_2^*(X)g_2(X) \equiv a_2^2(X)g_2^2(X)s_1(X)g_1(X) \pmod{g_{12}(X)g_1(X)g_2(X)}$$
$$\Rightarrow \quad b_1^2(X)g_1(X)s_2^*(X) \equiv a_2^2(X)g_2(X)s_1(X) \pmod{g_{12}(X)}.$$

We know $\gcd(g_2(X)s_1(X), g_{12}(X)) \in \mathbb{F}_{2^m}$ because of $\gcd(g_2(X), g_{12}(X)), \gcd(s_1(X), g_{12}(X)) \in \mathbb{F}_{2^m}$. Therefore, there exists the inverse of $g_2(X)s_1(X)$ modulo $g_{12}(X)$, and we have the following equation.

$$b_1^2(X)u(X) \equiv a_2^2(X) \pmod{g_{12}(X)} \text{ where } u(X) := g_1(X)s_2^*(X)(g_2(X)s_1(X))^{-1}.$$

Since $u(X)$ has a square root modulo $g_{12}(X)$ (Remark 2.2.1), $a(X) = a_2(X)g_2(X)$ and $b(X) = b_1(X)g_1(X)$ are obtained by calculating $a_2(X)$ and $b_1(X)$ that satisfy the following equations using the extended Euclidean algorithm.

$$b_1(X)\sqrt{u(X)} \equiv a_2(X) \pmod{g_{12}(X)}, \quad \deg(a_2) \le \left\lfloor \frac{t}{2} \right\rfloor - \deg(g_2),\ \deg(b_1) \le \left\lfloor \frac{t-1}{2} \right\rfloor - \deg(g_1).$$

Remark 2.2.1. Since all elements of $\mathbb{F}_{2^{13}}$ are roots of the equation $X^{2^{13}} - X = 0$, we know

$$g_{12}(X) \mid X^{2^{13}} - X \quad \Rightarrow \quad X^{2^{13}} \equiv X \pmod{g_{12}(X)} \quad \Rightarrow \quad \sqrt{X} \equiv X^{2^{12}} \bmod g_{12}(X).$$

A polynomial $u(X) = \sum_{i=0}^{l} u_i X^i \in \mathbb{F}_{2^{13}}[X]$ of degree l can be written as follows.

$$u(X) = \left(\sum_{i=0}^{\lfloor \frac{l}{2} \rfloor} \sqrt{u_{2i}} X^i \right)^2 + \left(\sum_{i=0}^{\lfloor \frac{l-1}{2} \rfloor} \sqrt{u_{2i+1}} X^i \right)^2 X.$$

where $\sqrt{a_j} = (a_j)^{2^{12}}$ for all j. Thus, the square root $\sqrt{u(X)}$ of $u(X)$ modulo $g_{12}(X)$ is

$$\sqrt{u(X)} \equiv \left(\sum_{i=0}^{\lfloor \frac{l}{2} \rfloor} \sqrt{u_{2i}} X^i \right) + \left(\sum_{i=0}^{\lfloor \frac{l-1}{2} \rfloor} \sqrt{u_{2i+1}} X^i \right) \sqrt{X} \bmod g_{12}(X).$$

We give the sage code for a binary separable Goppa code used in PALOMA in Appendix A.

Chapter 3

Specification

3.1 Definitions

The notations, symbols and functions used throughout this paper are listed below.

Notation

$[l]$	integer set $\{0, 1, \ldots, l-1\}$
$[l_1 : l_2]$	integer set $\{l_1, l_1+1, \ldots, l_2-1\}$
$\{0,1\}^l$	set of all l-bit strings
$a\|b$	concatenation of two bit strings a and b
$a_{[l]}$	first l-bit string $a_0\|a_1\|\cdots\|a_{l-1}$ of a bit string $a = a_0\|a_1\|\cdots$
$a_{[i:j]}$	substring $a_i\|a_{i+1}\|\cdots\|a_{j-1}$ of a bit string $a = a_0\|a_1\|\cdots$
$\mathbb{F}_q$	finite field with q elements
$\mathbb{F}_q^{m\times n}$	set of all $m \times n$ matrices over a field $\mathbb{F}_q$
$\mathbb{F}_q^l$	set of all $l \times 1$ matrices over a field $\mathbb{F}_q$, i.e., $\mathbb{F}_q^l := \mathbb{F}_q^{l\times 1}$ ($v \in {\mathbb{F}_q}^l$ is considered as a column vector)
0^l	zero vector with length l
v_I	subvector $(v_j)_{j\in I} \in \mathbb{F}_q^{\|I\|}$ of a vector $v = (v_0, v_1, \ldots, v_{l-1}) \in \mathbb{F}_q^l$
$\mathrm{supp}\,(e)$	function that returns the non-zero position set of a given vector e
$w_H(e)$	function that returns Hamming weight of a given vector e
$d_H(u,v)$	function that returns Hamming distance of given two vectors u, v
$\mathbf{M}^{-1}$	the inverse matrix of a matrix $\mathbf{M}$
$\mathbf{M}^T$	the transposed matrix of a matrix $\mathbf{M}$
$\mathbf{I}_l$	$l \times l$ identity matrix
$\mathbf{M}_I$	submatrix $[m_{r,c}]_{c\in I}$ of a matrix $\mathbf{M} = [m_{r,c}]$ where r and c are row index and column index, respectively
$\mathbf{M}_{I\times J}$	submatrix $[m_{r,c}]_{r\in I,\ c\in J}$ of a matrix $\mathbf{M} = [m_{r,c}]$ where r and c are row index and column index, respectively
$[\mathbf{A} \mid \mathbf{B}]$	concatenated matrix of two matrices $\mathbf{A}$ and $\mathbf{B}$
$\mathcal{P}_l$	set of all $l \times l$ permutation matrices
$[n,k,d]_2$	linear code over $\mathbb{F}_2$ with length n, dimension k and minimum distance d

$A \bmod B$	function that returns the remainder after dividing A by B
$div(A, B)$	function that returns the quotient and the remainder after dividing A by B
$\deg(f)$	degree of a given polynomial f
$\gcd(f(X), g(X))$	function that returns the monic greatest common divisor polynomial of $f(X)$ and $g(X)$
$x \xleftarrow{\$} X$	x randomly chosen in a set X

Symbols

pk	public key
sk	secret key
$e, \widehat{e}$	error vectors
$s, \widehat{s}$	syndrome vectors
$r, \widehat{r}$	random bit string
L	support set
$g(X)$	Goppa polynomial
$\mathcal{C}, \mathcal{C}_{L,g}$	binary separable Goppa code generated by a support set L and a Goppa polynomial $g(X)$
$\mathbf{H}$	parity-check matrix of $\mathcal{C}$
$\widehat{\mathcal{C}}$	scrambled code of $\mathcal{C}$
$\widehat{\mathbf{H}}$	parity-check matrix of $\widehat{\mathcal{C}}$

Functions

GenKeyPair	function that returns a public key and a secret key pair
Encrypt	function that returns the syndrome vector of a given error vector with a public key
Decrypt	function that returns the error vector of a given syndrome vector with a secret key
Encap	function that returns a key and a ciphertext with a public key
Decap	function that returns a key of a given ciphertext with a secret key
Lsh	512-bit hash function LSH-512, the national standard of South Korea (KS X 3262), that returns an 512-bit hash value of a given bit string
Rref	function that returns the reduced row echelon form of a given matrix

3.2 Parameter Sets

The followings are the parameters of PALOMA.

m	degree of a binary field extension, i.e., $m = [\mathbb{F}_{2^m} : \mathbb{F}_2]$
t	number of correctable errors
n	length of a codeword ($n \le 2^m - t$)
k	dimension of a code ($k = n - mt$)

PALOMA consists of PALOMA-128, PALOMA-192, PALOMA-256 with 128/192/256-bit security strength, respectively. Table 3.1 shows each parameter set.

Table 3.1: Parameter Sets of PALOMA

Parameter	m	t	$n^\dagger$	$k^\ddagger$
PALOMA-128	13	64	3904	3072
PALOMA-192	13	128	5568	3904
PALOMA-256	13	128	6592	4928

† $n \le 2^m - t$, ‡ $mt = n - k$

Finite field $\mathbb{F}_{2^{13}}$ used in PALOMA is $\mathbb{F}_2[z]/\langle f(z)\rangle$ where $f(z)$ is an irreducible polynomial $f(z) = z^{13} + z^7 + z^6 + z^5 + 1 \in \mathbb{F}_2[z]$.

3.3 Key Generation

The trapdoor of PALOMA is designed with SDP based on a scrambled code $\widehat{\mathcal{C}}$ of a binary separable Goppa code $\mathcal{C}$. In PALOMA, the public key is the submatrix of the systematic parity-check matrix of $\widehat{\mathcal{C}}$, and the private key is the information for decoding and scrambling of $\mathcal{C}$. The key generation of PALOMA is as follows. (Algorithm 1 shows the pseudo-code of the key generation)

Step 1. Generation of a random binary separable Goppa code $\mathcal{C}$. (Algorithm 2)

Generate a support set $L \subseteq \mathbb{F}_{2^{13}}$, a Goppa polynomial $g(X) \in \mathbb{F}_{2^{13}}[X]$ for a Goppa code $\mathcal{C}_{L,g}$, and compute the parity check matrix $\mathbf{H} \in \mathbb{F}_2^{13t\times n}$ of $\mathcal{C}_{L,g}$.

(i) Reorder elements of $\mathbb{F}_{2^{13}}$ with a random 256-bit string r using the Shuffle, defined in Algorithm 4.

$$\mathbb{F}_{2^{13}} = [0, 1, z, z+1, z^2, \ldots, z^{12} + \cdots + 1] \xrightarrow{\textsc{Shuffle with } r} [\alpha_0, \ldots, \alpha_{2^m-1}].$$

(ii) Set a support set $L = [\alpha_0, \ldots, \alpha_{n-1}]$.

(iii) Set a separable Goppa polynomial $g(X)$ with degree t whose roots are $\alpha_n, \ldots, \alpha_{n+t-1}$, i.e.,

$$g(X) = \sum_{j=0}^{t} g_j X^j = \prod_{j=n}^{n+t-1} (X - \alpha_j) \in \mathbb{F}_{2^{13}}[X].$$

(iv) Compute the parity check matrix $\mathbf{H} = \mathbf{ABC}$ where $\mathbf{A}, \mathbf{B}, \mathbf{C}$ are defined in (2.1).

(v) Parse $\mathbf{H}$ as a matrix in $\mathbb{F}_2^{13t\times n}$ because a Goppa code is the subfield subcode of the code, i.e.

$$\mathbf{H} = [h_{r,c}] \in \mathbb{F}_{2^{13}}^{t\times n} \quad \Rightarrow \quad \mathbf{H} := [h_0 \mid h_1 \mid \cdots \mid h_{n-1}] \in \mathbb{F}_2^{13t\times n},$$

where $h_c := [h_{0,c}^{(0)} \mid \cdots \mid h_{0,c}^{(12)} \mid h_{1,c}^{(0)} \mid \cdots \mid h_{1,c}^{(12)} \mid \cdots \mid h_{t-1,c}^{(12)}]^T \in \mathbb{F}_2^{13t}$ and $h_{r,c}^{(j)} \in \mathbb{F}_2$ such that $h_{r,c} = \sum_{j=0}^{12} h_{r,c}^{(j)} z^j \in \mathbb{F}_{2^{13}}$ for $r \in [t]$ and $c \in [n]$.

Step 2. Generation of a scrambled code $\widehat{\mathcal{C}}$ of $\mathcal{C}$. (Algorithm 3)

The parity check matrix $\mathbf{H}$ of $\mathcal{C}$ is scrambled below.

(i) Reorder elements of $[n]$ with a random 256-bit string r using the Shuffle. (Algorithm 4)

$$[n] = [0, 1, 2, \ldots, n-1] \quad \xrightarrow{\text{Shuffle with } r} \quad [l_0, \ldots, l_{n-1}].$$

(ii) Compute $\mathbf{HP}$ where $\mathbf{P} \in \mathcal{P}_n$ is the permutation matrix defined by

$$\mathbf{P} := \mathbf{P}_{0,l_0}\mathbf{P}_{1,l_1} \cdots \mathbf{P}_{n-1,l_{n-1}},$$

and $\mathbf{P}_{j,l_j} \in \mathcal{P}_n$ is the permutation matrix for swapping j-th column and l_j-th column. (Algorithm 5) Note that $\mathbf{P}^{-1}$ is $\mathbf{P}_{n-1,l_{n-1}} \cdots \mathbf{P}_{1,l_1}\mathbf{P}_{0,l_0}$.

(iii) Compute the reduced row echelon form $\widehat{\mathbf{H}}$ of $\mathbf{HP}$. If $\widehat{\mathbf{H}}_{[n-k]} \neq \mathbf{I}_{n-k}$, back to (i).

(iv) There exists the invertible matrix $\mathbf{S} \in \mathbb{F}_2^{(n-k)\times(n-k)}$ such that $\widehat{\mathbf{H}} = \mathbf{SHP}$, i.e., $\mathbf{S}^{-1} = (\mathbf{HP})_{[n-k]}$.

Step 3. Since $\widehat{\mathbf{H}}$ is a systematic form matrix, i.e., $\widehat{\mathbf{H}}_{[n-k]} = \mathbf{I}_{n-k}$, return $\widehat{\mathbf{H}}_{[n-k:n]}$ as a public key pk and $(L, g(X), \mathbf{S}^{-1}, r)$ as a secret key sk.

$$pk := \widehat{\mathbf{H}}_{[n-k:n]} \in \mathbb{F}_2^{(n-k)\times k}, \quad sk := (L, g(X), \mathbf{S}^{-1}, r).$$

Shuffle parses a 256-bit random bit string $r = r_0 \| r_1 \| \cdots \| r_{255} \in \{0,1\}^{256}$ as a 16-bit sequence $(r_{[16w:16(w+1)]})_{w=0,\ldots,15}$ and uses each as a random integer required in the Fisher-Yates shuffle. Algorithm 4 shows the process of Shuffle in detail.

Remark 3.3.1. Since $\mathbf{S}^{-1}$ can be computed from $L, g(X)$, and $L, g(X)$ are generated from a 256-bit random string r', the secret key can be defined as a 512-bit string $r' \| r \in \{0,1\}^{512}$.

3.4 Encryption and Decryption

Encryption

PALOMA encryption is as follows. (Algorithm 6)

Step 1. Retrieve the parity check matrix $\widehat{\mathbf{H}} = [\mathbf{I} \mid \widehat{\mathbf{H}}_{[n-k:n]}]$ of the scrambled code $\widehat{\mathcal{C}}$ from the public key $pk = \widehat{\mathbf{H}}_{[n-k:n]} \in \mathbb{F}_2^{(n-k)\times k}$.

Step 2. Return the $(n-k)$-bit syndrome $\widehat{s}(= \widehat{\mathbf{H}}\widehat{e})$ of an n-bit input $\widehat{e} \in \{0,1\}^n$ for $\widehat{\mathbf{H}}$ as a ciphertext of $\widehat{e}$.

Algorithm 1 PALOMA: Generation of Key Pair

Input: Parameter set (t, n)

Output: A public key pk and a secret key sk

1: **procedure** GenKeyPair(t, n)
2: $L, g(X), \mathbf{H} \leftarrow$ GenRandGoppaCode(t, n)
3: $\mathbf{S}^{-1}, r, \widehat{\mathbf{H}} \leftarrow$ GetScrambledCode$(\mathbf{H})$ ▷ $\widehat{\mathbf{H}} = \mathbf{SHP}$
4: $pk \leftarrow \widehat{\mathbf{H}}_{[n-k:n]}$ ▷ $\widehat{\mathbf{H}}_{[n-k:n]}$ is the submatrix of $\widehat{\mathbf{H}}$ consisting of the last k columns
5: $sk \leftarrow (L, g(X), \mathbf{S}^{-1}, r)$
6: **return** pk and sk
7: **end procedure**

Algorithm 2 PALOMA: Generation of a Random Goppa Code

Input: Parameter set (t, n)

Output: A support set L, a Goppa polynomial $g(X)$ and a parity-check matrix $\mathbf{H}$ of $\mathcal{C}$

1: **procedure** GenRandGoppaCode(t, n)
2: $r \xleftarrow{\$} \{0,1\}^{256}$
3: $[\alpha_0, \ldots, \alpha_{2^{13}-1}] \leftarrow$ Shuffle$(\mathbb{F}_{2^{13}}, r)$ ▷ Algorithm 4
4: $L \leftarrow [\alpha_0, \ldots, \alpha_{n-1}]$ ▷ support set of $\mathcal{C}$
5: $g(X) \leftarrow \prod_{j=n}^{n+t-1} (X - \alpha_j)$ ▷ separable Goppa polynomial of $\mathcal{C}$
6: $\mathbf{H} = [h_{r,c}] \leftarrow \mathbf{ABC} \in \mathbb{F}_{2^{13}}^{t \times n}$ ▷ $\mathbf{A}, \mathbf{B}, \mathbf{C}$ are defined in (2.1)
7: $h_c \leftarrow [h_{0,c}^{(0)} \mid \cdots \mid h_{0,c}^{(12)} \mid h_{1,c}^{(0)} \mid \cdots \mid h_{1,c}^{(12)} \mid \cdots \mid h_{t-1,c}^{(12)}]^T \in \mathbb{F}_2^{13t}$ for $c \in [n]$ where $h_{r,c}^{(j)} \in \mathbb{F}_2$ such that $h_{r,c} = \sum_{j=0}^{12} h_{r,c}^{(j)} z^j \in \mathbb{F}_{2^{13}}$
8: $\mathbf{H} \leftarrow [h_0 \mid h_1 \mid \cdots \mid h_{n-1}] \in \mathbb{F}_2^{13t \times n}$ ▷ parity-check matrix of $\mathcal{C}$
9: **return** $L, g(X)$, $\mathbf{H}$
10: **end procedure**

Algorithm 3 PALOMA: Generation of a Scrambled Code

Input: A parity-check matrix $\mathbf{H}$ of $\mathcal{C}$

Output: An invertible matrix $\mathbf{S}^{-1}$, a random bits r and a systematic parity-check matrix $\widehat{\mathbf{H}}$ of $\widehat{\mathcal{C}}$

1: **procedure** GetScrambledCode$(\mathbf{H})$
2: $r \xleftarrow{\$} \{0,1\}^{256}$
3: $\mathbf{P}, \mathbf{P}^{-1} \leftarrow$ GenRandPermMat(r) ▷ Algorithm 5
4: $[\widehat{\mathbf{H}} \mid \mathbf{S}] \leftarrow$ Rref$([\mathbf{HP} \mid \mathbf{I}_{n-k}])$ ▷ $\widehat{\mathbf{H}} \in \mathbb{F}_2^{(n-k) \times n}$, $\mathbf{S} \in \mathbb{F}_2^{(n-k) \times (n-k)}$
5: **if** $\widehat{\mathbf{H}}_{[n-k]} \neq \mathbf{I}_{n-k}$ **then** ▷ $\widehat{\mathbf{H}}_{[n-k]}$ is the submatrix of $\widehat{\mathbf{H}}$ consisting of the first $n-k$ columns
6: Go back to line 2.
7: **end if**
8: $\mathbf{S}^{-1} \leftarrow (\mathbf{HP})_{[n-k]}$
9: **return** $\mathbf{S}^{-1}$, r, $\widehat{\mathbf{H}}$
10: **end procedure**

Algorithm 4 PALOMA: Shuffling with an 256-bit seed

Input: An ordered set $A = [A_0, A_1, \ldots, A_{l-1}]$ and a random 256-bit string r

Output: A shuffled set A

procedure $\textsc{Shuffle}(A, r)$

  $r \leftarrow r \| r \| r \| \cdots$

  $w \leftarrow 0$

  for $i \leftarrow l - 1$ downto 1 **do**

    $j \leftarrow \mathrm{B2I}(r_{[16w:16(w+1)]}) \bmod i + 1$ $j \xleftarrow{\$} [i + 1]$ ▷ $\mathrm{B2I}(r_0 \| \cdots \| r_{15}) = \sum_{j=0}^{15} r_j 2^j$

    $\mathrm{swap}(A_i,\ A_j)$

    $w \leftarrow w + 1$

  end for

  return A

end procedure

Algorithm 5 PALOMA: Generation of a Random Permutation Matrix

Input: A random 256-bit string r

Output: An $n \times n$ permutation matrix $\mathbf{P}, \mathbf{P}^{-1}$

procedure $\textsc{GenRandPermMat}(r)$

  $[l_0, \ldots, l_{n-1}] \leftarrow \textsc{Shuffle}([n], r)$ ▷ Algorithm 4

  $$\mathbf{P} \leftarrow \prod_{j=0}^{n-1} \mathbf{P}_{j,l_j} = \mathbf{P}_{0,l_0}\mathbf{P}_{1,l_1} \cdots \mathbf{P}_{n-1,l_{n-1}} \text{ where } \mathbf{P}_{i,j} := \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 0 & & 1 & \\ & & & \ddots & & \\ & & 1 & & 0 & \\ & & & & & \ddots & \\ & & & & & & 1 \end{pmatrix}.$$

  $\mathbf{P}^{-1} \leftarrow \mathbf{P}_{n-1,l_{n-1}} \cdots \mathbf{P}_{1,l_1}\mathbf{P}_{0,l_0}$

  return $\mathbf{P}, \mathbf{P}^{-1}$

end procedure

Decryption

PALOMA decryption is as follows. (Algorithm 6)

Step 1. Convert the syndrome $\widehat{s} \in \{0,1\}^{n-k}$ of the input $\widehat{\mathcal{C}}$ into the syndrome $s(= \mathbf{S}^{-1}\widehat{s})$ of $\mathcal{C}$ by multiplying the secret key $\mathbf{S}^{-1}$.

Step 2. Recover the error vector e corresponding to s with the secret key $L, g(X)$, which are decoding information of $\mathcal{C}$. At that time, we use the extended Patterson decoding introduced by Section 2.2.2. (Algorithm 7)

Step 3. Return the error vector $\widehat{e}(= \mathbf{P}^{-1}e)$ of $\widehat{\mathcal{C}}$ obtained from e and the permutation matrix $\mathbf{P}^{-1}$ generated by the secret key r.

Figure 3.1 shows these operations.

Algorithm 6 PALOMA: Encryption and Decryption

Input: A public key $pk = \widehat{\mathbf{H}}_{[n-k:n]} \in \mathbb{F}_2^{(n-k)\times n}$ and an error vector $\widehat{e} \in \mathbb{F}_2^n$ with $w_H(\widehat{e}) = t$
Output: A syndrome vector $\widehat{s} \in \mathbb{F}_2^{n-k}$

procedure ENCRYPT($pk = \widehat{\mathbf{H}}_{[n-k:n]}; \widehat{e}$)
  $\widehat{\mathbf{H}} \leftarrow [\mathbf{I}_{n-k} \mid \widehat{\mathbf{H}}_{[n-k:n]}] \in \mathbb{F}_2^{(n-k)\times n}$
  $\widehat{s} \leftarrow \widehat{\mathbf{H}}\widehat{e} \in \mathbb{F}_2^{n-k}$
  return $\widehat{s}$
end procedure

Input: A secret key $sk = (L, g(X), \mathbf{S}^{-1}, r)$ and a syndrome vector $\widehat{s} \in \mathbb{F}_2^{n-k}$
Output: An error vector $\widehat{e} \in \mathbb{F}_2^n$ with $w_H(\widehat{e}) = t$

procedure DECRYPT($sk = (L, g(X), \mathbf{S}^{-1}, r); \widehat{s}$)
  $s \leftarrow \mathbf{S}^{-1}\widehat{s}$
  $e \leftarrow$ RECERRVEC($L, g(X); s$) ▷ Algorithm 7
  $\mathbf{P}, \mathbf{P}^{-1} \leftarrow$ GENRANDPERMMAT(r) ▷ Algorithm 5
  $\widehat{e} \leftarrow \mathbf{P}^{-1}e$
  return $\widehat{e}$
end procedure

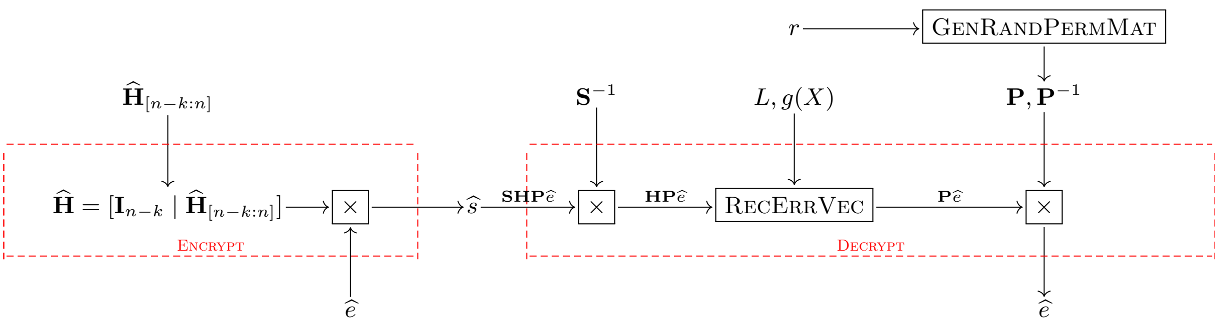


Figure 3.1: PALOMA: Encryption and Decryption

Algorithm 7 PALOMA: Recovering an Error Vector in $\mathcal{C}$ (Extended Patterson Decoding)

Input: A support set L, Goppa polynomial $g(X)$ and a syndrome vector $s \in \mathbb{F}_2^{n-k}$
Output: An error vector $e \in \mathbb{F}_2^n$ with $w_H(e) = t$

procedure RecErrVec($L, g(X); s$)
  $s(X) \leftarrow$ ToPoly(s)
  $\widehat{s}(X), g_1(X), g_2(X), g_{12}(X) \leftarrow$ ConstructKeyEqn($s(X), g(X)$) ▷ Algorithm 8
  $a_2(X), b_1(X) \leftarrow$ SolveKeyEqn($\widehat{s}(X), g_{12}(X), \lfloor \frac{t}{2} \rfloor - \deg(g_2), \lfloor \frac{t-1}{2} \rfloor - \deg(g_1)$) ▷ Algorithm 9
  $a(X), b(X) \leftarrow a_2(X)g_2(X), b_1(X)g_1(X)$
  $\sigma(X) \leftarrow a^2(X) + b^2(X)X$ ▷ σ is the error locator polynomial of e
  $e \leftarrow$ FindErrVec($\sigma(X)$)
  return e
end procedure

Input: A syndrome vector $s = (s_0, s_1, \ldots, s_{13t-1}) \in \mathbb{F}_2^{13t}$
Output: A syndrom polynomial $s(X) \in \mathbb{F}_{2^{13}}[X]$

procedure ToPoly(s)
  for $j = 0$ to $t-1$ **do**
    $w_j \leftarrow \sum_{i=0}^{12} s_{13j+i} z^i \in \mathbb{F}_{2^{13}}$
  end for
  $s(X) \leftarrow \sum_{j=0}^{t-1} w_j X^j \in \mathbb{F}_{2^{13}}[X]$
  return $s(X)$
end procedure

Output: An error locator polynomial $\sigma(X)$ and a support set L
Input: An error vector $e \in \mathbb{F}_2^n$

procedure FindErrVec(σ, L)
  $e = (e_0, \ldots, e_{n-1}) \leftarrow (0, 0, \ldots, 0)$
  for $j = 0$ to $n-1$ **do**
    if $\sigma(\alpha_j) = 0$ **then**
      $e_j \leftarrow 1$
    end if
  end for
  return e
end procedure

Algorithm 8 PALOMA: Key Equation for an Error Locator Polynomial

Output: A syndrome polynomial $s(X)$ and a Goppa polynomial $g(X)$
Input: $\widehat{s}(X), g_1(X), g_2(X), g_{12}(X) \in \mathbb{F}_{2^{13}}[X]$

procedure ConstructKeyEqn($s(X), g(X)$)
  $s^*(X) \leftarrow 1 + Xs(X)$
  $g_1(X), g_2(X) \leftarrow \gcd(g(X), s(X)), \gcd(g(X), s^*(X))$
  $g_{12}(X) \leftarrow \frac{g(X)}{g_1(X)g_2(X)}$
  $s_2^*(X), s_1(X) \leftarrow \frac{s^*(X)}{g_2(X)}, \frac{s(X)}{g_1(X)}$
  $u(X) \leftarrow g_1(X)s_2^*(X)(g_2(X)s_1(X))^{-1} \bmod g_{12}(X)$
  $\widehat{s}(X) \leftarrow \sqrt{u(X)} \bmod g_{12}(X)$
  return $\widehat{s}(X), g_1(X), g_2(X), g_{12}(X)$
end procedure

Algorithm 9 PALOMA: Solving a Key Equation for an Error Locator Polynomial

Output: $\widehat{s}(X)$, $g_{12}(X)$, $dega$, $degb$

Input: $a_1(X), b_2(X)$ such that $b_2(X)\widehat{s}(X) \equiv a_1(X) \pmod{g_{12}(X)}$ and $\deg(a_1) \leq dega, \deg(b_2) \leq degb$

procedure SolveKeyEqn($\widehat{s}(X), g_{12}(X), dega, degb$)
  $a_0(X), a_1(X) \leftarrow \widehat{s}(X), g_{12}(X)$
  $b_0(X), b_1(X) \leftarrow 1, 0$
  while $a_1(X) = 0$ **do**
    $q(X), r(X) \leftarrow div(a_0(X), a_1(X))$
    $a_0(X), a_1(X) \leftarrow a_1(X), r(X)$
    $b_2(X) \leftarrow b_0(X) - q(X)b_1(X)$
    $b_0(X), b_1(X) \leftarrow b_1(X), b_2(X)$
    if $\deg(a_0) \leq dega$ and $\deg(b_0) \leq degb$ **then**
      `break`
    end if
  end while
  return $a_0(X)$, $b_0(X)$
end procedure

3.5 Encapsulation and Decapsulation

Random Oracles

PALOMA is a KEM designed by random oracle model. PALOMA uses two random oracles, RO_G and RO_H, defined as the Korean KS standard hash function LSH-512. Algorithm 10 shows the definition.

Algorithm 10 PALOMA: Random Oracles

Input: An l-bit string $x \in \{0,1\}^l$

Output: An 256-bit string $r \in \{0,1\}^{256}$

procedure $\mathrm{RO}_G(x)$
  return LSH(`"PALOMAGG"`$\|x)_{[:256]}$ ▷ ASCII(`"PALOMAGG"`) = `0x50414c4f4d414747`
end procedure

procedure $\mathrm{RO}_H(x)$
  return LSH(`"PALOMAHH"`$\|x)_{[:256]}$ ▷ ASCII(`"PALOMAHH"`) = `0x50414c4f4d414848`
end procedure

Encapsulation

PALOMA Encap has a public key pk as an input and returns a key k and the ciphertext $c = (\widehat{r}, \widehat{s})$ of the k. The procedure is as follows. (Algorithm 11)

Step 1. Reorder elements of $[n]$ with a random 256-bit string r^* using the Shuffle. (Algorithm 4)

$$[n] = [0, 1, 2, \ldots, n-1] \quad \xrightarrow{\text{Shuffle with } r^*} \quad [l_0, \ldots, l_{n-1}].$$

Step 2. Define n-bit error vector $e^* \in \{0,1\}^n$ such that $\mathrm{supp}\,(e^*) = \{l_0, \ldots, l_{t-1}\}$.

Step 3. Query e^* to the random oracle RO_G and obtain a 256-bit string $\widehat{r} \in \{0,1\}^{256}$.

Step 4. Compute the permutation matrix $\mathbf{P}, \mathbf{P}^{-1} \in \mathcal{P}_n$ corresponding to $\widehat{r}$ using Algorithm 5.

Step 5. Compute $\widehat{e} = \mathbf{P}e^*$.

Step 6. Obtain the syndrome $\widehat{s} \in \{0,1\}^{n-k}$ of $\widehat{e}$ using Encrypt equipped with the public key pk.

Step 7. Query $(e^* \| \widehat{r} \| \widehat{s})$ to the random oracle RO_H and obtain a 256-bit key $k \in \{0,1\}^{256}$.

Step 8. Return the key k and its ciphertext $c = (\widehat{r}, \widehat{s})$.

Figure 3.2a outlines Encap.

Algorithm 11 PALOMA: Encapsulation

Input: A public key $pk \in \{0,1\}^{(n-k)\times n}$

Output: A key $k \in \{0,1\}^{256}$ and a ciphertext $c = (\widehat{r}, \widehat{s}) \in \{0,1\}^{256} \times \{0,1\}^{n-k}$

procedure Encap(pk)
- $r^* \xleftarrow{\$} \{0,1\}^{256}$
- $e^* \leftarrow$ GenRandErrVec(r^*) ▷ Algorithm 13
- $\widehat{r} \leftarrow \mathrm{RO}_G(e^*)$ ▷ $\widehat{r} \in \{0,1\}^{256}$
- $\mathbf{P}, \mathbf{P}^{-1} \leftarrow$ GenRandPermMat($\widehat{r}$) ▷
- $\widehat{e} \leftarrow \mathbf{P}e^*$
- $\widehat{s} \leftarrow$ Encrypt($pk; \widehat{e}$) ▷ $\widehat{s} \in \{0,1\}^{n-k}$
- $k \leftarrow \mathrm{RO}_H(e^* \| \widehat{r} \| \widehat{s})$ ▷ $k \in \{0,1\}^{256}$
- **return** k and $c = (\widehat{r}, \widehat{s})$

end procedure

Decapsulation

Decap of PALOMA returns the key k when passing the secret key sk and the ciphertext $c = (\widehat{r}, \widehat{s})$ as inputs. The process is as follows. (Algorithm 12)

Step 1. Obtain the error vector $\widehat{e}$ by entering $\widehat{s}$ into the Decrypt function set to the secret key sk.

Step 2. Generate the permutation matrix $\mathbf{P}, \mathbf{P}^{-1} \in \mathcal{P}_n$ from $\widehat{r}$ which is part of the ciphertext c.

Step 3. Compute $e^* = \mathbf{P}^{-1}\widehat{e}$.

Step 4. Query e^* to the RO_G and obtain a 256-bit string $\widehat{r}' \in \{0,1\}^{256}$.

Step 5. Generate the error vector $\widetilde{e}$ using GenRandErrVec with the secret key r.

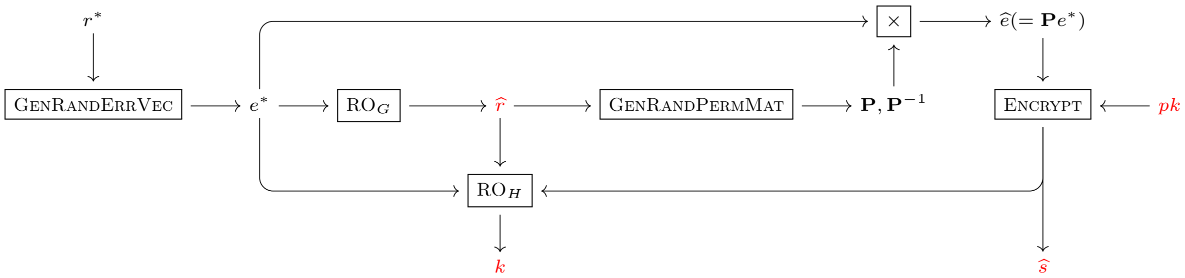
Step 6. If $\widehat{r}' = \widehat{r}$, then query $(e^* \| \widehat{r} \| \widehat{s})$ to the random oracle RO_H, and if not, query $(\widetilde{e} \| \widehat{r} \| \widehat{s})$ to RO_H. Return the received bit string from RO_H as a key k.

Figure 3.2b outlines Decap.

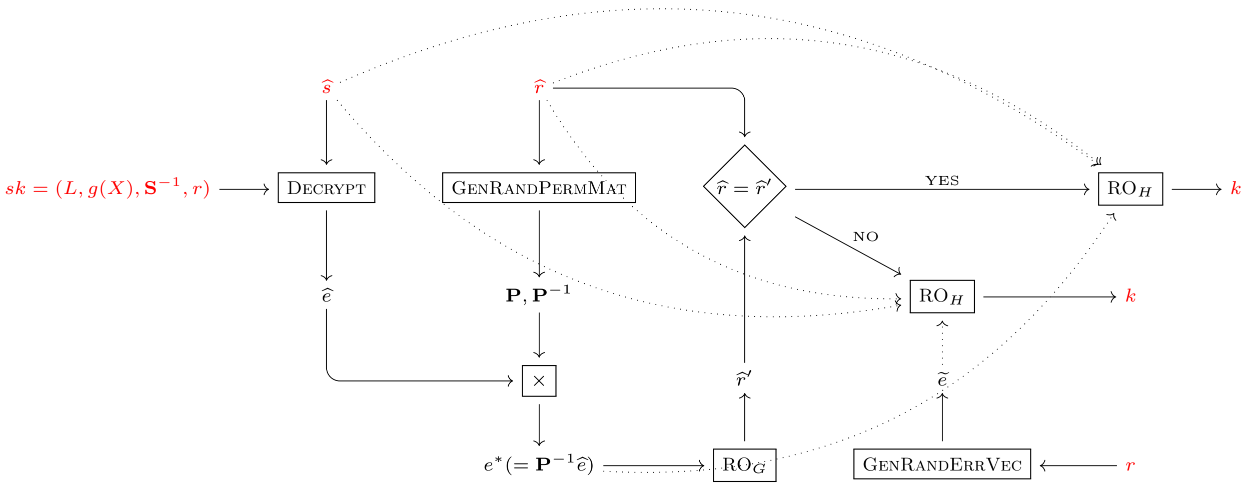
Algorithm 12 PALOMA: Decapsulation

Input: A secret key $sk = (L, g(X), \mathbf{S}^{-1}, r)$ and a ciphertext $c = (\widehat{r}, \widehat{s}) \in \{0,1\}^{256} \times \{0,1\}^{n-k}$
Output: A key $k \in \{0,1\}^{256}$

```
procedure DECAP(sk = (L, g(X), S^{-1}, r); c = (r̂, ŝ))
    ê ← DECRYPT(sk; ŝ)                                  ▷ Algorithm 6
    P, P^{-1} ← GENRANDPERMMAT(r̂)                       ▷ Algorithm 5
    e* ← P^{-1} ê                                        ▷ ê ∈ {0,1}^n
    r̂′ ← RO_G(e*)
    ẽ ← GENRANDERRVEC(r)
    if r̂′ ≠ r̂ then                                      ▷ k ∈ {0,1}^256
        return k ← RO_H(ẽ‖r̂‖ŝ)
    end if
    k ← RO_H(e*‖r̂‖ŝ)                                    ▷ k ∈ {0,1}^256
    return k
end procedure
```



(a) $k, (\widehat{r}, \widehat{s}) \leftarrow$ ENCAP(pk)



(b) $k \leftarrow$ DECAP$(sk, (\widehat{r}, \widehat{s}))$

Figure 3.2: PALOMA: Encapsulation and Decapsulation

Algorithm 13 PALOMA: Generating a Random Error Vector

Input: A random 256-bit string $r \in \{0,1\}^{256}$

Output: An error vector $e = (e_0, e_1, \ldots, e_{n-1}) \in \mathbb{F}_2^n$

```
procedure GenRandErrVec(r)
    e = (e_0, e_1, ..., e_{n-1}) ← (0, 0, ..., 0)
    (l_0, l_1, ..., l_{n-1}) ← Shuffle([n], r)
    for j = 0 to t − 1 do
        e_{l_j} ← 1
    end for
    return e
end procedure
```

Chapter 4

Performance Analysis

In this chapter, we provide the performance analysis result of PALOMA.

4.1 Description of Benchmark

4.1.1 Platforms

PALOMA is implemented in ANSI C. Speed benchmark is performed in the following two platforms.

Platform 1. macOS Monterey ver.12.5, Apple M1, 8GB RAM

Platform 2. macOS Monterey ver.12.4, Intel core i5, 8GB RAM

We use the GCC compiler (ver.13.1.6.) with speed option `-O2`.

4.1.2 Data Structure for a Polynomial Ring $\mathbb{F}_{2^{13}}[X]$

The elements of $\mathbb{F}_{2^{13}} = \mathbb{F}_2[z]/\langle f(z)\rangle$ are stored in the 2-byte data type `unsigned short`. The data structure for a field element is defined as follows.

$$a(z) = \sum_{i=0}^{12} a_i z^i \in \mathbb{F}_{2^{13}} \quad \Leftrightarrow \quad 0\|0\|0\|a_{12}\|a_{11}\|\cdots\|a_0 \in \{0,1\}^{16}.$$

A polynomial $a(X) \in \mathbb{F}_{2^{13}}[X]$ with degree l is stored in $2(l+1)$-byte as follows.

$$a(X) = \sum_{i=0} a_i X^i \in \mathbb{F}_{2^{13}} \quad \Leftrightarrow \quad a_0\|a_1\|\cdots\|a_l \in \{0,1\}^{2(l+1)}.$$

4.1.3 Arithmetics in $\mathbb{F}_{2^{13}}$ using Pre-computated Tables

PALOMA uses the pre-computed tables for multiplication, square, square root, and inverse in $\mathbb{F}_{2^{13}}$.

(i) Multiplication in $\mathbb{F}_{2^{13}}$: To store the multiplication of all pairs in $\mathbb{F}_{2^{13}}$, the table of 128 MB(=2×2^{26}-byte) is required. To decrease the size of a table, PALOMA deals with the multiplication of three small sizes of tables. $a(z), b(z) \in \mathbb{F}_{2^{13}}$ can be written as follows.

$$a = a_1(z)z^7 + a_0(z), \quad b = b_1(z)z^7 + b_0(z), \quad (deg(a_0), deg(b_0) \leq 6,\ deg(a_1), deg(b_1) \leq 5).$$

So, the multiplication of $a(z)$ and $b(z) \in \mathbb{F}_{2^{13}}$ can be computed as follows.

$$
\begin{aligned}
&a(z)b(z) \bmod f(z) \\
&= (a_1(z)z^7 + a_0(z))(b_1(z)z^7 + b_0(z)) \bmod f(z) \\
&= \left(a_1(z)b_1(z)z^{14} \bmod f(z)\right) + \left(a_1(z)b_0(z)z^7 \bmod f(z)\right) + \left(a_0(z)b_1(z)z^7 \bmod f(z)\right) + (a_0b_0(z)) .
\end{aligned}
$$

Thus, the multiplication in $\mathbb{F}_{2^{13}}$ can be calculated by the following three tables for all possible pairs.

Table 1. $\mathsf{MUL}_{00} : \{0,1\}^7 \times \{0,1\}^7 \to \{0,1\}^{16}$ defined by $\mathsf{MUL}_{00}[a_0, b_0] = a_0(z)b(z) \bmod f(z)$

Table 2. $\mathsf{MUL}_{10} : \{0,1\}^6 \times \{0,1\}^7 \to \{0,1\}^{16}$ defined by $\mathsf{MUL}_{10}[a_1, b_0] = a_1(z)b_0(z)z^7 \bmod f(z)$

Table 3. $\mathsf{MUL}_{11} : \{0,1\}^6 \times \{0,1\}^6 \to \{0,1\}^{16}$ defined by $\mathsf{MUL}_{11}[a_1, b_1] = a_1(z)b_1(z)z^{14} \bmod f(z)$

Note that $(a_1(z)b_0(z))z^7 \bmod f(z)$ is computed using the table MUL_{10}.

(ii) Squaring, square root, inversion in $\mathbb{F}_{2^{13}}$: Tables SQU, SQRT and INV store the results of a square, square root, and inverse for all elements in $\mathbb{F}_{2^{13}}$, respectively. Note that we define the inverse of 0 as 0.

Table 4.1 shows the size of pre-computed tables for arithmetics in $\mathbb{F}_{2^{13}}$ used in PALOMA.

Table 4.1: Precomputed Tables for Arithmetics in $\mathbb{F}_{2^{13}}$ used in PALOMA

Table	Size (in bytes)	Description
MUL_{00}	32,768	$a_0(z)b_0(z)$
MUL_{10}	16,384	$a_1(z)b_0(z)z^7 \bmod f(z)$
MUL_{11}	8,192	$a_1(z)b_1(z)z^{14} \bmod f(z)$
SQU	16,384	$a(z)^2 \bmod f(z)$
SQRT	16,384	$\sqrt{a(z)}$ where $a(z) = \left(\sqrt{a(z)}\right)^2 \bmod f(z)$
INV	16,384	$a(z)^{-1}$ where $1 = a(z)^{-1}a(z) \bmod f(z)$
Total	106,496	

4.2 Performance of Reference Implementation

4.2.1 Data Size

We determine the size of a public key, a secret key, and a ciphertext in terms of byte strings. Each size in bytes is computed by the following formula.

$$\mathsf{bytelen}(pk) = \mathsf{bytelen}(\widehat{\mathbf{H}}_{[n-k:n]}) = \left\lceil \frac{(n-k)k}{8} \right\rceil ,$$

$$\begin{aligned}\textsf{bytelen}(sk) &= \textsf{bytelen}(L) + \textsf{bytelen}(g) + \textsf{bytelen}(\mathbf{S}^{-1}) + \textsf{bytelen}(r) \\ &= n\left\lceil\frac{13}{8}\right\rceil + t\left\lceil\frac{13}{8}\right\rceil + \left\lceil\frac{(n-k)^2}{8}\right\rceil + 32.\end{aligned}$$

The size of data of PALOMA-128, PALOMA-192, and PALOMA-256 is shown in Table 4.2.

Table 4.2: Data Size Performance of PALOMA (in bytes)

		PALOMA-128	PALOMA-192	PALOMA-256
Public key $pk = \widehat{\mathbf{H}}_{[n-k:n]}$	$\widehat{\mathbf{H}}_{[n-k:n]} \in \mathbb{F}_2^{(n-k)\times k}$	319,488	812,032	1,025,024
Secret key	$L \in \mathbb{F}_{2^{13}}^n$	7,808	11,136	13,184
$sk = (L, g, \mathbf{S}^{-1}, r)$	$g(X) \in \mathbb{F}_{2^{13}}[X]$	128	256	256
	$\mathbf{S}^{-1} \in \mathbb{F}_2^{(n-k)\times(n-k)}$	86,528	346,112	346,112
	$r \in \{0,1\}^{256}$	32	32	32
	Total	94,496	357,536	359,584
Ciphertext	$\widehat{r} \in \{0,1\}^{256}$	32	32	32
$c = (\widehat{r}, \widehat{s})$	$\widehat{s} \in \mathbb{F}_2^{(n-k)}$	104	208	208
	Total	136	240	240
Key k	$k \in \{0,1\}^{256}$	32	32	32

As mentioned in Remark 3.3.1, the size of a secret key can be 512-bit. However, this degrades the speed performance of DECRYPT.

Table 4.3 shows the data size comparison among the NIST competition round 4 code-based ciphers and PALOMA.

The data size of PALOMA is similar to Classic McEliecebecause of the usage of SDP-based trapdoor. Compared to HQC and BIKE, the size of a public key and a secret key is relatively large. However, the size of the ciphertext which is the actual transmitted value is smaller than HQC and BIKE. Therefore, PALOMA is suitable for the situation of long-term key or reused key.

4.2.2 Speed

We measure the operation time for each function of PALOMA in two platforms. The results are shown in Table 4.4.

Compare the time of PALOMA with Classic McEliece, which is the same SDP-based KEM. Time is measured in the Apple M1 platform.

Compared to Classic McEliece, an SDP-based trapdoor, PALOMA operates faster except for the parameter providing a 192-bit security. It is the reason that the number of correctable errors($= t$) among 192-bit security parameters is 128 in PALOMA compared to 96 Classic McEliece.

Table 4.3: Data Size Comparison of Code-based KEMs (in bytes)

Algorithm	Security	Public key	Secret key	Ciphertext	Key
hqc-128	128	2,249	40	4,481	64
BIKE	128	1,541	281	1,573	32
mceliece348864	128	261,120	6,452	128	32
PALOMA-128	128	319,488	94,496	136	32
hqc-192	192	4,522	40	9,026	64
BIKE	192	3,083	419	3,115	32
mceliece460896	192	524,160	13,568	188	32
PALOMA-192	192	812,032	355,400	240	32
hqc-256	256	7,245	40	14,469	64
BIKE	256	5,122	580	5,154	32
mceliece6688128	256	1,044,992	13,892	240	32
mceliece6960119	256	1,047,319	13,908	226	32
mceliece8192128	256	1,357,824	14,080	240	32
PALOMA-256	256	1,025,024	357,064	240	32

Table 4.4: Speed Performance of PALOMA (in milliseconds)

		PALOMA-128		PALOMA-192		PALOMA-256	
		M1	Intel	M1	Intel	M1	Intel
GenKeyPair	GenRandGoppaCode	15	26	74	144	93	168
	GetScrambledCode	42	61	179	263	211	281
	total	64	89	261	423	323	469
Encrypt		0.002	0.003	0.003	0.004	0.003	0.005
Decrypt	ConstructKeyEqn	8	12	53	92	53	92
	SolveKeyEqn	0.2	0.4	2	3	2	3
	FindErrVec	1	2	3	4	4	5
	total	10	14	59	100	59	101
Encap		0.03	0.05	0.04	0.07	0.04	0.08
Decap		9	15	59	101	60	101

Table 4.5: Speed Performance Comparison between PALOMA and Classic McEliece (in milliseconds)

		GenKeyPair	Encap	Decap
128-bit	PALOMA-128	64	0.03	9
	mceliece348864	74	0.04	18
192-bit	PALOMA-192	258	0.04	58
	mceliece460896	211	0.06	42
256-bit	PALOMA-256	323	0.04	58
	mceliece6688128	517	0.10	82

Chapter 5

Security

5.1 OW-CPA-secure PKE

When evaluating the security of PALOMA, even though there have been no known critical attacks on binary separable Goppa codes, we need to assume that the scrambling code of a Goppa code is indistinguishable from a random matrix. Therefore, the security of PALOMA is evaluated by the number of bit operations of ISD, which is the most powerful generic attack of an NP-hard SDP.

From now on, let $\mathsf{SDP}(\mathbf{H}, s, w)$ be the root set of SDP defined with a parity check matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$, a syndrome $s \in \mathbb{F}_2^{n-k}$, and a Hamming weight w, and let $\mathcal{E}_w^n$ be the set of all n-bit vectors with Hamming weight w. The zero matrix is denoted by $\mathbf{0}$. The parameters n, t, and k of PALOMA assure that the base SDP has a unique root and are all even.

5.1.1 Exhaustive Search

The naive algorithm finding roots of SDP is an exhaustive search. It checks all candidate vectors with a Hamming weight of w, that is, it checks if the sum of all possible w columns in a matrix $\mathbf{H}$ equals the syndrome. Algorithm 14 shows the exhaustive search algorithm in detail.

To generate $t_l (l = 1, 2, \ldots, w)$ in Algorithm 14, one column vector addition is required. Since t_l is defined from $j_1, \ldots, j_l$, $\binom{n}{l}$ column vector additions are required to generation t_l. Therefore, the total number of column vector additions T is as follows.

$$T = \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{w}.$$

If $w < \frac{n}{2}$, T approximates $\binom{n}{w}$. Therefore, the amount of the exhaustive search is $O\left(\binom{n}{w}(n-k)\right)$ in terms of bit operations.

5.1.2 Birthday-type Decoding

For a random permutation matrix $\mathbf{P} \in \mathcal{P}_n$, $\mathsf{SDP}(\mathbf{H}, s, w)$ and $\mathsf{SDP}(\mathbf{HP}, s, w)$ have the following necessary and sufficient conditions.

$$e \in \mathsf{SDP}(\mathbf{H}, s, w) \quad \Leftrightarrow \quad \mathbf{P}^{-1}e \in \mathsf{SDP}(\mathbf{HP}, s, w).$$

Algorithm 14 Exhaustive Search of SDP

Input: $\mathbf{H} = [h_1 \mid h_2 \mid \cdots \mid h_n] \in \mathbb{F}_2^{(n-k)\times n}$, $s \in \mathbb{F}_2^{n-k}$, and w
Output: $e \in \mathbb{F}_2^n$ such that $\mathbf{H}e = s$ and $w_H(e) = w$

```
for j_1 = 1 to n − (w − 1) do
    t_1 ← s + h_{j_1}
    for j_2 = j_1 + 1 to n − (w − 2) do
        t_2 ← t_1 + h_{j_2}
        ···
        for j_w = j_{w−1} + 1 to n do
            t_w ← t_{w−1} + h_{j_w}
            if t_w = 0^{n−k} then
                set e with supp(e) = {j_1, ..., j_w}
                return e
            end if
        end for
    end for
end for
```

Birthday-type decoding transforms SDP until finding the solution $\widehat{e} = (\widehat{e}_I \| \widehat{e}_J) \in \mathsf{SDP}(\widehat{\mathbf{H}}(= \mathbf{HP}), s, w)$ that satisfies $w_H(\widehat{e}_I) = w_H(\widehat{e}_J) = \frac{w}{2}$ for $I = [\frac{n}{2}]$ and $J = [n] \setminus I$, a random permutation matrix $\mathbf{P} \in \mathcal{P}_n$. To find $\widehat{e}_I$ and $\widehat{e}_J$, check the intersection of the following two sets.

$$T_I := \left\{ s + \widehat{\mathbf{H}}_I \widehat{e}_I \in \mathbb{F}_2^{n-k} : \widehat{e}_I \in \mathcal{E}_{w/2}^{n/2} \right\}, \quad T_J := \left\{ \widehat{\mathbf{H}}_J \widehat{e}_J \in \mathbb{F}_2^{n-k} : \widehat{e}_J \in \mathcal{E}_{w/2}^{n/2} \right\}.$$

Two sets must satisfy $|T_I| = |T_J| \geq 2^{\frac{n-k}{2}}$ to have a intersection with $1/2$ probability. However, since the parameter of PALOMA is $\binom{n/2}{w/2} \ll 2^{\frac{n-k}{2}}$, the probability that an intersection exists is very low. For the root $e \in \mathsf{SDP}(\mathbf{H}, s, w)$, since the probability that $\widehat{e}$ satisfies the hamming weight condition is $p = \binom{n/2}{w/2}^2 / \binom{n}{w}$, the process of transforming SDP to a new permutation matrix $\mathbf{P}$ must be repeated at least $1/p$ times. Algorithm 15 shows this attack in detail.

Since the number of bit computations for $\widehat{\mathbf{H}}_I \widehat{e}_I$ and $\widehat{\mathbf{H}}_J \widehat{e}_J$ are $O(\binom{n/2}{w/2}(n-k))$, the total amount of computations is as follows.

$$2\binom{n}{w}(n-k) \Big/ \binom{n/2}{w/2}. \tag{5.1}$$

To bring the probability p close to 1 in birthday-type decoding, define the following two subsets I and J of $[n]$

$$I = [n/2 + \varepsilon], \quad J = [n/2 - \varepsilon : n] \text{ for some } \varepsilon > 0.$$

When we find $e_1, e_2 \in \mathcal{E}_{w/2}^{n/2+\varepsilon}$ which satisfy $s + \widehat{\mathbf{H}}_I e_1 = \widehat{\mathbf{H}}_J e_2$, it does not assume that $(e_1 \| 0^{\frac{n}{2}-\varepsilon}) + (0^{\frac{n}{2}-\varepsilon} \| e_2)$ is a root. If $w_H((e_1 \| 0^{\frac{n}{2}-\varepsilon}) + (0^{\frac{n}{2}-\varepsilon} \| e_2) = w$, then $(e_1 \| 0^{\frac{n}{2}-\varepsilon}) + (0^{\frac{n}{2}-\varepsilon} \| e_2)$ is the root. So this discriminant must be added. In this attack, ε is set to a value that makes the probability $p = \binom{n/2+\varepsilon}{w/2}^2 / \binom{n}{w}$ close to 1. The calculated amount of birthday-type decoding is counted as

Algorithm 15 Birthday-type Decoding of SDP

Input: $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$ and $s \in \mathbb{F}_2^{n-k}$, w and $I = [\frac{n}{2}]$, $J = [n] \setminus I$
Output: $e \in \mathbb{F}_2^n$ such that $\mathbf{H}e = s$ and $w_H(e) = w$

while `true` **do**
  $\mathbf{P} \xleftarrow{\$} \mathcal{P}_n$
  $\widehat{\mathbf{H}} \leftarrow \mathbf{HP}$
  $T[j] \leftarrow$ `null` for all $j \in \{0,1\}^{n-k}$
  for $\widehat{e}_I$ in $\mathcal{E}_{w/2}^{n/2}$ **do**
    $u \leftarrow s + \widehat{\mathbf{H}}_I \widehat{e}_I$ // #operations: $\binom{n/2}{w/2}(n-k)$ (exhaustive search)
    $T[u] \leftarrow \widehat{e}_I$
  end for
  for $\widehat{e}_J$ in $\mathcal{E}_{w/2}^{n/2}$ **do**
    $u \leftarrow \widehat{\mathbf{H}}_J \widehat{e}_J$ // #operations: $\binom{n/2}{w/2}(n-k)$ (exhaustive search)
    if $T[u] \neq$ `null` **then**
      $\widehat{e} \leftarrow (T[u] \| \widehat{e}_J)$ ▷ $L[u] = \widehat{e}_I$
      return $\mathbf{P}\widehat{e}$
    end if
  end for
end while

follows.

$$2(n-k)\binom{n/2+\varepsilon}{w/2} \approx 2(n-k)\sqrt{\binom{n}{w}}. \tag{5.2}$$

5.1.3 Improved Birthday-type Decoding

We can find the root of SDP from the roots of two small sizes of SDPs. Consider $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$ as a concatenation of two submatrices $\mathbf{H}_1$ and $\mathbf{H}_2$ for some $r \leq n-k$ as follows.

$$\mathbf{H} = \begin{pmatrix} \mathbf{H}_1 \\ \mathbf{H}_2 \end{pmatrix}, \text{ where } \mathbf{H}_1 \in \mathbb{F}_2^{r\times n}, \mathbf{H}_2 \in \mathbb{F}_2^{(n-k-r)\times n}.$$

For the roots $x, y \in \mathbb{F}_2^n$ of two SDPs for $\mathbf{H}_1$ below,

$$x \in \mathsf{SDP}\left(\mathbf{H}_1, s_{[r]}, w/2+\varepsilon\right), \quad y \in \mathsf{SDP}\left(\mathbf{H}_1, 0^r, w/2+\varepsilon\right),$$

if x and y satisfy the following, then $x+y$ is the root of $\mathsf{SDP}(\mathbf{H}, s, w)$.

$$\mathbf{H}_2(x+y) = s_{[r:n-k]}, \quad w_H(x+y) = w,$$

Algorithm 16 shows this attack in detail.

The amount of bit operation in this algorithm is as follows.

$$4r\sqrt{\binom{n}{w/2+\varepsilon}} + \frac{\binom{n}{w/2+\varepsilon}}{2^r}\left((w+2\varepsilon)(n-k-r) + \frac{n\binom{n}{w/2+\varepsilon}}{2^{n-k}}\right).$$

Algorithm 16 Improved Birthday-type Decoding of SDP

Input: $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$, $s \in \mathbb{F}_2^{n-k}$, w and r
Output: $e \in \mathbb{F}_2^n$ such that $\mathbf{H}e = s$ and $w_H(e) = w$

$T[j] \leftarrow \emptyset$ for all $j \in \{0,1\}^{n-k-r}$
for x in $\mathsf{SDP}\left(\mathbf{H}_1, s_{[r]}, w/2+\varepsilon\right)$ **do** // birthday-type decoding, $\left|\mathsf{SDP}\left(\mathbf{H}_1, s_{[r]}, w/2+\varepsilon\right)\right| \approx \frac{\binom{n}{w/2+\varepsilon}}{2^r}$
    $idx \leftarrow s_{[r:n-k]} + \mathbf{H}_2 x$ // num. of bit operations $= \frac{\binom{n}{w/2+\varepsilon}}{2^r}(w/2+\varepsilon)(n-k-r)$
    $T[idx] \leftarrow T[idx] \cup \{x\}$
end for
for y in $\mathsf{SDP}\left(\mathbf{H}_1, 0^r, w/2+\varepsilon\right)$ **do** // birthday-type decoding, $\left|\mathsf{SDP}\left(\mathbf{H}_1, 0^r, w/2+\varepsilon\right)\right| \approx \frac{\binom{n}{w/2+\varepsilon}}{2^r}$
    $idx \leftarrow \mathbf{H}_2 y$ // num. of bit operations $= \frac{\binom{n}{w/2+\varepsilon}}{2^r}(w/2+\varepsilon)(n-k-r)$
    for x in $T[idx]$ **do** // $|T[idx]| \approx \frac{\binom{n}{w/2+\varepsilon}}{2^r} \times \frac{1}{2^{n-k-r}}$
        $e \leftarrow x + y$ // num. of bit operations $= \frac{\binom{n}{w/2+\varepsilon}}{2^r} \times \frac{n\binom{n}{w/2+\varepsilon}}{2^{n-k}}$
        if $w_H(e) = w$ **then**
            return e
        end if
    end for
end for

Choice of e. When two subsets A and B with the number of elements $w/2+\varepsilon$ are randomly selected from the set $[n] = \{0, \ldots, n-1\}$, the expected value $E[|A \cap B|]$ is $\frac{(w/2+\varepsilon)^2}{n}$. Therefore, for the roots x and y of each SDP, $E[w_H(x+y)]$ is as follows.

$$\begin{aligned} E[w_H(x+y)] &= E[2(|\mathrm{supp}\,(x)| - |\mathrm{supp}\,(x) \cap \mathrm{supp}\,(y)|)] \\ &= 2E[|\mathrm{supp}\,(x)|] - 2E[|\mathrm{supp}\,(x) \cap \mathrm{supp}\,(y)|)] \\ &= 2\,(w/2+\varepsilon) - \frac{2(w/2+\varepsilon)^2}{n}. \end{aligned}$$

Set ε to satisfy $\varepsilon = \frac{(w/2+\varepsilon)^2}{n}$. (i.e. $\varepsilon = \frac{\sqrt{n^2-2nw}+(n-w)}{2}$.) Then $E[w_H(x+y)] = w$.

Choice of r. For $e \in \mathsf{SDP}(\mathbf{H}, s, w)$, the number of (x, y) pairs satisfying $e = x + y$ as follows.

$$|\{(x,y) \in (\mathcal{E}^n_{w/2+\varepsilon})^2 : e = x + y\}| = \binom{w}{w/2}\binom{n-w}{\varepsilon}.$$

Therefore, set r to satisfy $2^r \approx \binom{w}{w/2}\binom{n-w}{\varepsilon}$ to count the number of roots of small SDP accurately.

5.1.4 Information Set Decoding

ISD(Information Set Decoding) is a generic decoding algorithm for random linear codes. The first phase of ISD is to transform the parity check matrix $\mathbf{H}$ into a systematic type for finding an error-free information set. Then, in the second phase, we find error vectors satisfying certain conditions, partly using birthday attack type search and partial brute force attacks. First proposed by E. Prange in 1962, ISD has improved computational complexity by changing the conditions of error vectors and applying search techniques in terms of birthday attacks.

5.1.4.1 Procedure.

ISD uses Proposition 5.1.1, the relationship between the code $\mathcal{C}$ and the scrambled code $\widehat{\mathcal{C}}$ of $\mathcal{C}$ in terms of the root of SDP.

Proposition 5.1.1. *Let $e \in \mathsf{SDP}(\boldsymbol{H}, s, w)$. For an invertible matrix $\boldsymbol{S} \in \mathbb{F}_2^{(n-k)\times(n-k)}$ and a permutation matrix $\boldsymbol{P} \in \mathcal{P}_n$, $\boldsymbol{P}^{-1}e \in \mathsf{SDP}(\boldsymbol{SHP}, \boldsymbol{S}s, w)$.*

Proof. Since $(\mathbf{SHP})(\mathbf{P}^{-1}e) = \mathbf{S}(\mathbf{H}e) = \mathbf{S}s$ and $w = w_H(e) = w_H(\mathbf{P}^{-1}e)$, $\mathbf{P}^{-1}e \in \mathsf{SDP}(\mathbf{SHP}, \mathbf{S}s, w)$. □

ISD is a probabilistic algorithm that modifies SDP until finding a root satisfying certain conditions. ISD proceeds to the following two-phase.

(Phase 1) Redefining a problem: Find $\mathsf{SDP}(\mathbf{H}, s, w) \Rightarrow$ Find $\mathsf{SDP}(\widehat{\mathbf{H}} = \mathbf{SHP}, \widehat{s} = \mathbf{S}s, w)$

SHP is a partial systematic matrix obtained by applying elementary row operations. i.e.

$$\mathbf{H} \xrightarrow{\text{random permutation } \mathbf{P}} \mathbf{HP} \xrightarrow{\text{Gaussian elimination}} \mathbf{SHP} = \begin{pmatrix} \mathbf{I}_l & \mathbf{M}_1 \\ \mathbf{0} & \mathbf{M}_2 \end{pmatrix}.$$

(Phase 2) Find $\widehat{e}(= \mathbf{P}^{-1}e) \in \mathsf{SDP}(\widehat{\mathbf{H}}, \widehat{s}, w)$ which satisfies the specific Hamming weight condition and return $e(= \mathbf{P}\widehat{e})$. If no root satisfies the condition, go back to (Phase 1).

5.1.4.2 Computational Complexity.

Let p be the probability that the root $\widehat{e}$ satisfies a specific Hamming weight condition in the modified problem. The computational complexity of ISD is as follows.

$$\frac{1}{p} \times \Big(\text{(Phase 1)'s computational amount} + \text{(Phase 2)'s computational amount} \Big).$$

(Phase 1) is to modify the problem using the Gaussian elimination, so most ISD algorithms result in $O((n-k)^2 n)$ bit operations in this phase. ISD has developed while improving the computational amount of (Phase 2) and the probability p. Table 5.1 shows the matrix form and hamming weight conditions used in the significant ISD algorithms.

We thought that the BJMM-ISD was the most effective ISD because the proposed ISDs after the BJMM-ISD in 2012 are minor improvements in specific situations. Therefore, the parameters of PALOMA were selected based on the precise calculation of the number of bit operations that happened in BJMM-ISD. BJMM-ISD transforms the SDP into a small SDP and finds a root of the SDP applying to birthday-type attacks.

5.1.4.3 Becker-Joux-May-Meurer (2012).

BJMM-ISD is an ISD that applies improvided birth-type decoding to the partial RREF[1]. Transform $\mathbf{H}$ into the following form $\widehat{\mathbf{H}}$ by applying a partial RREF for some $l(\leq n-k)$.

$$\widehat{\mathbf{H}} = \mathbf{SHP} = \left(\begin{array}{c|c} \mathbf{I}_{n-k-l} & \mathbf{H}_1 \\ \hline \mathbf{0} & \mathbf{H}_2 \end{array} \right) \text{ where } \mathbf{H}_1 \in \mathbb{F}_2^{(n-k-l)\times(k+l)},\ \mathbf{H}_2 \in \mathbb{F}_2^{l\times(k+l)}.$$

Table 5.1: Hamming weight condition of ISD algorithms

ISD	$\widehat{\mathbf{H}}(= \mathbf{SHP})$	Hamming weight condition of $\widehat{e}$	Prob.
Prange(1962)	$[\mathbf{I}_{n-k} \mid \mathbf{M}]$	$\underbrace{\boxed{w}}_{n-k}\underbrace{\boxed{0}}_{k}$	$\frac{\binom{n-k}{w}}{\binom{n}{w}}$
LB(1988)	$[\mathbf{I}_{n-k} \mid \mathbf{M}]$	$\underbrace{\boxed{w-p}}_{n-k}\underbrace{\boxed{p}}_{k}$	$\frac{\binom{n-k}{w-p}\binom{k}{p}}{\binom{n}{w}}$
Leon(1988)	$\left(\begin{array}{c\|c\|c} \mathbf{I}_{n-k-l} & 0 & \mathbf{M}_L \\ \hline 0 & \mathbf{I}_l & \mathbf{M}_R \end{array}\right)$	$\underbrace{\boxed{w-p}}_{n-k-l}\underbrace{\boxed{0}}_{l}\underbrace{\boxed{p}}_{k}$	$\frac{\binom{n-k-l}{w-p}\binom{k}{p}}{\binom{n}{w}}$
Stern(1989)	$\left(\begin{array}{c\|c\|c} \mathbf{I}_{n-k-l} & 0 & \mathbf{M}_L \\ \hline 0 & \mathbf{I}_l & \mathbf{M}_R \end{array}\right)$	$\underbrace{\boxed{w-2p}}_{n-k-l}\boxed{0}_{l}\underbrace{\boxed{p}}_{k/2}\underbrace{\boxed{p}}_{k/2}$	$\frac{\binom{n-k-l}{w-2p}\binom{k/2}{p}^2}{\binom{n}{w}}$
FS(2009)	$\left(\begin{array}{c\|c} \mathbf{I}_{n-k-l} & \\ \hline 0 & \mathbf{H} \end{array}\right)$	$\underbrace{\boxed{w-2p}}_{n-k-l}\underbrace{\boxed{2p}}_{k+l}$	$\frac{\binom{n-k-l}{w-2p}\binom{k+l}{2p}}{\binom{n}{w}}$
BLP(2011)	$\left(\begin{array}{c\|c\|c} \mathbf{I}_{n-k-l} & 0 & \mathbf{M} \\ \hline 0 & \mathbf{I}_l & \mathbf{N} \end{array}\right)$	$\underbrace{\boxed{w-2p-2q}}_{n-k-l}\underbrace{\boxed{q}}_{l/2}\underbrace{\boxed{q}}_{l/2}\underbrace{\boxed{p}}_{k/2}\underbrace{\boxed{p}}_{k/2}$	$\frac{\binom{n-k-l}{w-2p-2q}\binom{l/2}{q}^2\binom{k/2}{p}^2}{\binom{n}{w}}$
MMT(2011)	$\left(\begin{array}{c\|c} \mathbf{I}_{n-k-l} & \mathbf{H}_1 \\ \hline 0 & \mathbf{H}_2 \end{array}\right)$	$\underbrace{\boxed{w-p}}_{n-k-l}\underbrace{\boxed{p}}_{k+l}$	$\frac{\binom{n-k-l}{w-p}\binom{k+l}{p}}{\binom{n}{w}}$
BJMM(2012)	$\left(\begin{array}{c\|c} \mathbf{I}_{n-k-l} & \mathbf{H}_1 \\ \hline 0 & \mathbf{H}_2 \end{array}\right)$	$\underbrace{\boxed{w-p}}_{n-k-l}\underbrace{\boxed{p}}_{k+l}$	$\frac{\binom{n-k-l}{w-p}\binom{k+l}{p}}{\binom{n}{w}}$

Define the index sets I, J, and L as follows.

$$I := [n-k-l], \quad J := [n] \setminus I, \quad L := [n-k] \setminus I.$$

BJMM-ISD finds the root $\widehat{e} = (\widehat{e}_I \| \widehat{e}_J)$ of $\mathsf{SDP}(\widehat{\mathbf{H}} = \mathbf{SHP}, \widehat{s} = \mathbf{S}s, w)$ that satisfies the following conditions.

$$w_H(\widehat{e}_I) = w - p, \quad w_H(\widehat{e}_J) = p, \quad \widehat{e}_J \in \mathsf{SDP}(\mathbf{H}_2, \widehat{s}_L, p), \quad \widehat{e}_I + \widehat{e}_J \mathbf{H}_1 = \widehat{s}_I.$$

The process of BJMM-ISD is as follows.

(Phase 1) Randomly select a permutation matrix $\mathbf{P} \in \mathcal{P}_n$. Apply partial RREF to $\mathbf{HP}$ to obtain a partial canonical matrix as follows.

$$\widehat{\mathbf{H}} = \left(\begin{array}{c|c} \mathbf{I}_{n-k-l} & \mathbf{H}_1 \\ \hline \mathbf{0} & \mathbf{H}_2 \end{array} \right).$$

In this process, the invertible matrix $\mathbf{S}$ satisfying $\widehat{\mathbf{H}} = \mathbf{SHP}$ is obtained together. If there is no invertible matrix $\mathbf{S}$ that makes it a partial systematic form, (Phase 1) is performed again.

(Phase 2) Obtain $\mathsf{SDP}(\mathbf{H}_2, \widehat{s}_L, p)$ using the improved birthday-type decoding. If the root does not exist, go back to (Phase 1). If the Hamming weight of the vector $x := \widehat{s}_I + \mathbf{H}_1 y$ for $y \in \mathsf{SDP}(\mathbf{H}_2, \widehat{s}_L, p)$ is $w - p$, return $\mathbf{P}\widehat{e}$ because it is $\widehat{e} = (x \| y) \in \mathsf{SDP}(\widehat{\mathbf{H}}, \widehat{s}, w)$. If not, go back to (Phase 1).

Algorithm 17 shows BJMM-ISD process in detail.

The probability that $\widehat{e} = \mathbf{P}^{-1} e$ satisfies the Hamming Weight condition for $e \in \mathsf{SDP}(\mathbf{H}, s, w)$ in BJMM-ISD is as follows.

$$\Pr[(w_H(\widehat{e}_I) = w - p) \wedge (w_H(\widehat{e}_J) = p) \mid \mathbf{P} \xleftarrow{\$} \mathcal{P}_n] = \frac{\binom{n-k-l}{w-p}\binom{k+l}{p}}{\binom{n}{w}}. \tag{5.3}$$

Therefore, the bit operation calculation amount of the BJMM-ISD is as follows.

$$\frac{\binom{n}{w}}{\binom{n-k-l}{w-p}\binom{k+l}{p}} \left((n-k-l)(n-k)n + \frac{p(n-k-l)\binom{k+l}{p}}{2^l} + \text{num. of operations for } \mathsf{SDP}(\mathbf{H}_2, \widehat{s}_L, p) \right). \tag{5.4}$$

In this process, ε and r are set as follows when computing $\mathsf{SDP}(\mathbf{H}_2, \widehat{s}_L, p)$.

$$\varepsilon = \frac{\sqrt{(k+l)^2 - 2(k+l)p} + (k+l-p)}{2}, \quad r = \log_2 \left(\binom{p}{p/2} \binom{k+l-p}{\varepsilon} \right).$$

Table 5.2 shows the number of bit operations that occur in the several attacks in PALOMA.

Algorithm 17 BJMM-ISD(2012)

Input: $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$, $s \in \mathbb{F}_2^{n-k}$ and w

Output: $e \in \mathbb{F}_2^{n}$ such that $\mathbf{H}e = s$ and $w_H(e) = w$

while true **do**
    $\mathbf{P} \xleftarrow{\$} \mathcal{P}_n$
    $\widehat{\mathbf{H}} = \mathbf{SHP} \leftarrow$ partial RREF($\mathbf{HP}$)   // #operations $= (n-k-l)(n-k)n$
    if $\widehat{\mathbf{H}}_{I\times I} = \mathbf{I}_{n-k-l}$ **then**
        $\mathbf{H}_1, \mathbf{H}_2 \leftarrow \mathbf{H}_{J\times I}, \mathbf{H}_{J\times L}$ $\triangleright\ \widehat{\mathbf{H}} = \left(\begin{array}{c|c} \mathbf{I}_{n-k-l} & \mathbf{H}_1 \\ \hline \mathbf{0} & \mathbf{H}_2 \end{array}\right)$
        $\widehat{s} \leftarrow \mathbf{S}s$
        for y in SDP($\mathbf{H}_2, \widehat{s}_L, p$) **do**   // improved birthday-type decoding, $|\mathsf{SDP}(\mathbf{H}_2, \widehat{s}_L, p)| \approx \frac{\binom{k+l}{p}}{2^l}$
            $x \leftarrow \widehat{s}_I + \mathbf{H}_1 y$ // #operations $= p(n-k-l)\frac{\binom{k+l}{p}}{2^l}$
            if $w_H(x) = w - p$ **then**
                $\widehat{e} \leftarrow (x \| y)$
                return $\mathbf{P}\widehat{e}$
            end if
        end for
    end if
end while

Table 5.2: Computational Complexity of Several Attacks of PALOMA and Classic McEliece

	BJMM-ISD	Improved Birthday-type Decoding	Birthday-type Decoding	Exhaustive Search
PALOMA-128	$2^{166.21}$ $(l = 67, p = 14)$	$2^{225.78}$	$2^{244.11}$	$2^{476.52}$
PALOMA-192	$2^{267.77}$ $(l = 105, p = 22)$	$2^{399.67}$	$2^{448.91}$	$2^{885.11}$
PALOMA-256	$2^{289.66}$ $(l = 126, p = 26)$	$2^{415.59}$	$2^{464.66}$	$2^{916.62}$
mceliece348864	$2^{161.97}$ $(l = 66, p = 14)$	$2^{220.26}$	$2^{238.75}$	$2^{465.91}$
mceliece460896	$2^{215.59}$ $(l = 86, p = 18)$	$2^{311.80}$	$2^{345.58}$	$2^{678.88}$
mceliece6688128	$2^{291.56}$ $(l = 126, p = 26)$	$2^{416.95}$	$2^{466.01}$	$2^{919.32}$
mceliece6960119	$2^{289.92}$ $(l = 136, p = 28)$	$2^{402.41}$	$2^{443.58}$	$2^{874.57}$
mceliece8192128	$2^{318.34}$ $(l = 157, p = 32)$	$2^{436.05}$	$2^{484.90}$	$2^{957.10}$

5.1.5 Guessing Attacks

Since both the Goppa code and the error vector of PALOMA are generated from the 256-bit string, the estimated attack amount is 2^{256}, ensuring 256-bit security.

5.2 IND-CCA2-secure KEM

PALOMA is an IND-CCA2-secure KEM. According to the analysis results in Section 5.1, it is assumed that the underlying PKE = (GenKeyPair, Encrypt, Decrypt) of PALOMA is OW-CPA-secure. PKE has the following properties.

(i) (Injectivity) For all key pairs (pk, sk), if $\text{Encrypt}(pk; \widehat{e}_1) = \text{Encrypt}(pk; \widehat{e}_2)$, then $\widehat{e}_1 = \widehat{e}_2$.

(ii) (Correctness) $\Pr[\text{Decrypt}(sk; \widehat{s}) \neq \widehat{e} \mid \widehat{s} \leftarrow \text{Encrypt}(pk; \widehat{e})] = 0$.

There are several variants of the Fujisaki-Okamoto transformation. PALOMA is designed as IND-CCA2-secure using the above properties under the following assumptions.

Assumption 1. $\text{GenRandErrVec} : \{0,1\}^{256} \to \mathcal{E}_t^n$ *is injective. It means if* $r_1 \neq r_2$, *then* $\text{GenRandErrVec}(r_1) \neq \text{GenRandErrVec}(r_1)$.

According to this assumption, the size of the message space for Encrypt used inside PALOMA-Encap is 2^{256}, not $\binom{n}{t}$.

PALOMA is designed based on the implicit rejection $\mathsf{KEM}^{\not\perp} = \mathsf{U}^{\not\perp}[\mathsf{PKE}_1 = \mathsf{T}[\mathsf{PKE}_0, G], H]$ among FO-like transformations proposed by Hofheinz et al. [8]. This is combined with two modules: T: converting OW-CPA-secure PKE_0 to OW-PCA-secured PKE_1 and $\mathsf{U}^{\not\perp}$: converting it to IND-CCA2-secure KEM as follows.

$$\begin{aligned}
&\text{OW-CPA-secure } \mathsf{PKE}_0 = (\text{GenKeyPair}, \text{Encrypt}_0, \text{Decrypt}_0)\\
&\xrightarrow{\mathsf{T}\ \text{with a random oracle } G}\ \text{OW-PCA-secure } \mathsf{PKE}_1 = (\text{GenKeyPair}, \text{Encrypt}_1, \text{Decrypt}_1)\\
&\xrightarrow{\mathsf{U}^{\not\perp}\ \text{with a random oracle } H}\ \text{IND-CCA2-secure } \mathsf{KEM}^{\not\perp} = (\text{GenKeyPair}, \text{Encap}, \text{Decap}).
\end{aligned}$$

5.2.1 PKE_0 = (GenKeyPair, Encrypt_0, Decrypt_0)

PKE_0 is defined with the PKE and GenRandPermMat of PALOMA, as shown in Algorithm 18. Since PKE is assumed to be OW-CPA-secure, PKE_0 is OW-CPA-secure as well.

Algorithm 18 PALOMA: PKE_0

procedure $\text{Encrypt}_0(pk; \widehat{r}; e^*)$
  $\mathbf{P}, \mathbf{P}^{-1} \leftarrow \text{GenRandPermMat}(\widehat{r})$
  $\widehat{e} \leftarrow \mathbf{P}e^*$
  $\widehat{s} \leftarrow \text{Encrypt}(pk; \widehat{e})$
  return $c = (\widehat{r}, \widehat{s})$
end procedure

procedure $\text{Decrypt}_0(sk; c = (\widehat{r}, \widehat{s}))$
  $\widehat{e} \leftarrow \text{Decrypt}(sk; \widehat{s})$
  $\mathbf{P}, \mathbf{P}^{-1} \leftarrow \text{GenRandPermMat}(\widehat{r})$
  $e^* \leftarrow \mathbf{P}^{-1}\widehat{e}$
  return e^*
end procedure

5.2.2 $\mathsf{PKE}_1 = (\mathbf{GenKeyPair}, \mathbf{Encrypt}_1, \mathbf{Decrypt}_1)$

The transform T for converting $\mathsf{OW\text{-}PCA}$-secure PKE_0 to $\mathsf{OW\text{-}PCA}$-secure PKE_1 is defined by

$$\text{ENCRYPT}_1(pk; e^*) := \text{ENCRYPT}_0(pk; G(e^*); e^*).$$

Algorithm 19 and Figure 5.1 show PKE_1 of PALOMA constructed by this transformation T and a random oracle RO_G.

Algorithm 19 PALOMA: PKE_1

```
procedure ENCRYPT_1(pk; e*)
    r̂ ← RO_G(e*)
    c = (r̂, ŝ) ← ENCRYPT_0(pk; r̂; e*)
    return c = (r̂, ŝ)
end procedure
```

```
procedure DECRYPT_1(sk; c = (r̂, ŝ))
    e* ← DECRYPT_0(sk; ŝ)
    r̂' ← RO_G(e*)
    if r̂' ≠ r̂ then
        return ⊥
    end if
    return e*
end procedure
```

For any $\mathsf{OW\text{-}PCVA}$-attackers $\mathcal{B}$ on PKE_1, there exists an $\mathsf{OW\text{-}CPA}$-attacker $\mathcal{A}$ on PKE_0 satisfying the inequality below[8, Theorem 3.1].

$$\mathsf{Adv}^{\mathsf{OW\text{-}PCVA}}_{\mathsf{PKE}_1}(\mathcal{B}) \leq (q_G + q_P + 1)\mathsf{Adv}^{\mathsf{OW\text{-}CPA}}_{\mathsf{PKE}_0}(\mathcal{A}),$$

where q_G and q_P are the number of queries to the random oracle RO_G and plaintext-checking oracle PCO. Therefore, if PKE_0 is $\mathsf{OW\text{-}CPA}$-secure, $\mathsf{Adv}^{\mathsf{OW-PCVA}}_{\mathsf{PKE}_1}(\mathcal{B})$ is negligible, so PKE_1 is $\mathsf{OW\text{-}PCVA}$-secure.

5.2.3 $\mathsf{KEM}^{\not\perp} = (\mathbf{GenKeyPair}, \mathbf{Encap}, \mathbf{Decap})$

The transform $\mathsf{U}^{\not\perp}$ for converting $\mathsf{OW\text{-}PCA}$-secure PKE_1 to $\mathsf{IND\text{-}CCA2}$-securePKE_1 is as follows.

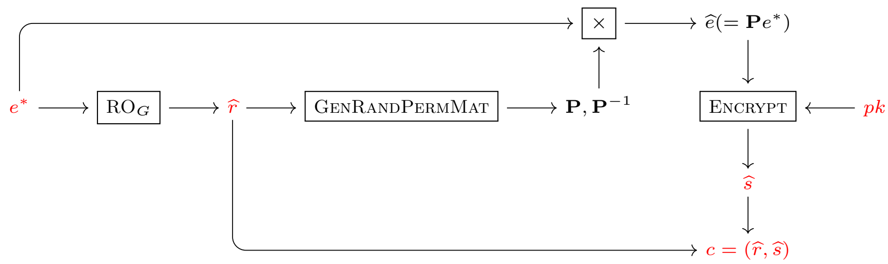
$$\text{ENCAP}(pk) := (c = \text{ENCRYPT}_1(pk; e^*), \text{RO}_H(e^*, c)).$$

Algorithm 20 shows $\mathsf{KEM}^{\not\perp}$ of PALOMA constructed by this transformation $\mathsf{U}^{\not\perp}$ and a random oracle RO_H.

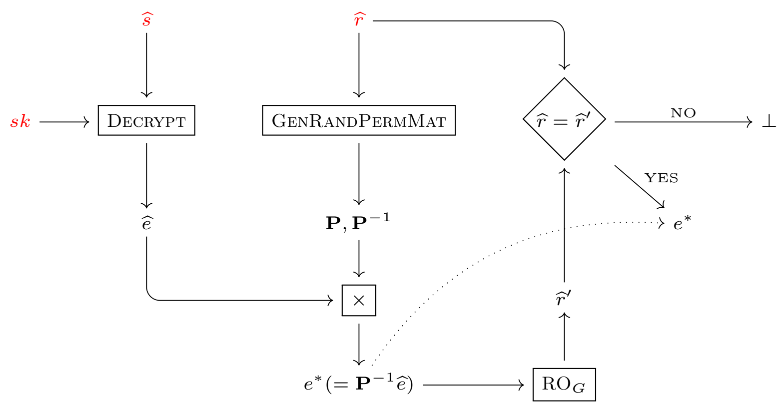
For any $\mathsf{IND\text{-}CCA2}$-attackers $\mathcal{B}$ on $\mathsf{KEM}^{\not\perp}$, there exists an $\mathsf{OW\text{-}PCA}$-attacker $\mathcal{A}$ on PKE_1 satisfying the inequality below[8, Theorem 3.4].

$$\mathsf{Adv}^{\mathsf{IND\text{-}CCA2}}_{\mathsf{KEM}^{\not\perp}}(\mathcal{B}) \leq \frac{q_H}{2^{256}}\mathsf{Adv}^{\mathsf{OW\text{-}PCA}}_{\mathsf{PKE}_1}(\mathcal{A}),$$

where q_H is the number of queries to the plaintext-checking oracle PCO. Therefore, if PKE_1 is $\mathsf{OW\text{-}PCA}$-secure, $\mathsf{Adv}^{\mathsf{OW-PCA}}_{\mathsf{PKE}_1}(A)$ is negligible, so $\mathsf{KEM}^{\not\perp}$ is $\mathsf{OW\text{-}PCVA}$-secure.



(a) $c = (\widehat{r}, \widehat{s}) \leftarrow \textsc{Encrypt}_1(pk; e^*)$



(b) $\widehat{e}$ or $\perp \leftarrow \textsc{Decrypt}_1(sk; c = (\widehat{r}, \widehat{s}))$

Figure 5.1: PALOMA: $\textsc{Encrypt}_1$ and $\textsc{Decrypt}_1$ of PKE_1

Algorithm 20 PALOMA: $\mathsf{KEM}^{\not\perp}$

procedure $\textsc{Encap}(pk)$
    $r^* \xleftarrow{\$} \{0,1\}^{256}$
    $e^* \leftarrow \textsc{GenRandErrVec}(r^*)$
    $c = (\widehat{r}, \widehat{s}) \leftarrow \textsc{Encrypt}_1(pk; e^*)$
    $k \leftarrow \mathrm{RO}_H(e^* \| \widehat{r} \| \widehat{s})$
    return k and $c = (\widehat{r}, \widehat{s})$
end procedure

procedure $\textsc{Decap}(sk = (L, g(X), \mathbf{S}^{-1}, r); c = (\widehat{r}, \widehat{s}))$
    $e^* \leftarrow \textsc{Decrypt}_1(sk; c = (\widehat{r}, \widehat{s}))$
    $\widehat{r}' \leftarrow \mathrm{RO}_G(e^*)$
    $\widetilde{e} \leftarrow \textsc{GenRandErrVec}(r)$
    if $\widehat{r}' \neq \widehat{r}$ or $e^* = \perp$ **then**
        return $k \leftarrow \mathrm{RO}_H(\widetilde{e} \| \widehat{r} \| \widehat{s})$
    end if
    $k \leftarrow \mathrm{RO}_H(e^* \| \widehat{r} \| \widehat{s})$
    return k
end procedure

Chapter 6

Summary

In this paper, we introduce PALOMA, which is IND-CCA2-secure KEM based on a SDP with a binary separable Goppa code. Even though the components and mechanisms used in PALOMA have been studied for a long time, no critical attacks are found. Many cryptographic communities believe the scheme constructed by these would be secure. Therefore, we believe PALOMA can be a reliable alternative to current cryptosystems in quantum computers. Classic McEliece is the round 4 cipher in NIST PQC competition, which use a binary Goppa code[4]. Finally, we give the feature comparison between PALOMA and Classic McEliece in Table 6.1.

Table 6.1: Comparison between PALOMA and Classic McEliece

	PALOMA	Classic McEliece
Scheme	Fujisaki-Okamoto-structure KEM (implicit rejection)	SXY-structure KEM (implicit rejection)
Problem	SDP	SDP
Trapdoor type	Niederreiter	Niederreiter
Linear code $\mathcal{C}$	Binary separable Goppa code	Binary irreducible Goppa code
Goppa polynomial $g(X)$	Separable (not irreducible)	Irreducible
Time for generating $g(X)$	Constant	Non-constant
Field $\mathbb{F}_{q^m}$	$\mathbb{F}_{2^{13}}$	$\mathbb{F}_{2^{12}}$, $\mathbb{F}_{2^{13}}$
Parity-check matrix $\mathbf{H}$ of $\mathcal{C}$	**ABC**	**BC**
Form of a parity-check matrix $\widehat{\mathbf{H}}$ of $\widehat{\mathcal{C}}$	Systematic	Systematic
Decoding algorithm	Extended Patterson	Berlekamp-Massey
Probability of decryption failure (correctness)	0	0

Bibliography

[1] Becker, A., Joux, A., May, A., Meurer, A.: Decoding random binary linear codes in 2n/20: How $1 + 1 = 0$ improves information set decoding. In: Pointcheval, D., Johansson, T. (eds.) Advances in Cryptology – EUROCRYPT 2012. pp. 520–536. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)

[2] Berlekamp, E.: Nonbinary bch decoding (abstr.). IEEE Transactions on Information Theory **14**(2), 242–242 (1968). https://doi.org/10.1109/TIT.1968.1054109

[3] Berlekamp, E., McEliece, R., van Tilborg, H.: On the inherent intractability of certain coding problems (corresp.). IEEE Transactions on Information Theory **24**(3), 384–386 (1978)

[4] Bernstein, D., Chou, T., Lange, T., von Maurich, I., Misoczki, R., Niederhagen, R., Persichetti, E., Peters, C., Schwabe, P., Sendrier, N., Szefer, J., Wang, W.: Classic mceliece (2017)

[5] Bezzateev, S.V., Noskov, I.K.: Patterson algorithm for decoding separable binary goppa codes. In: 2019 Wave Electronics and its Application in Information and Telecommunication Systems (WECONF). pp. 1–5 (2019). https://doi.org/10.1109/WECONF.2019.8840650

[6] Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology. p. 537–554. CRYPTO '99, Springer-Verlag, Berlin, Heidelberg (1999)

[7] Goppa, V.D.: A new class of linear error-correcting codes. Probl. Inf. Transm. **6**, 300–304 (1970)

[8] Hofheinz, D., Hövelmanns, K., Kiltz, E.: A modular analysis of the fujisaki-okamoto transformation. In: Kalai, Y., Reyzin, L. (eds.) Theory of Cryptography. pp. 341–371. Springer International Publishing, Cham (2017)

[9] Karp, R.M.: Reducibility among Combinatorial Problems, pp. 85–103. Springer US, Boston, MA (1972), `https://doi.org/10.1007/978-1-4684-2001-2_9`

[10] Lee, P.J., Brickell, E.F.: An observation on the security of mceliece's public-key cryptosystem. In: Barstow, D., Brauer, W., Brinch Hansen, P., Gries, D., Luckham, D., Moler, C., Pnueli, A., Seegmüller, G., Stoer, J., Wirth, N., Günther, C.G. (eds.) Advances in Cryptology — EUROCRYPT '88. pp. 275–280. Springer Berlin Heidelberg, Berlin, Heidelberg (1988)

[11] Leon, J.S.: A probabilistic algorithm for computing minimum weights of large error-correcting codes. IEEE Transactions on Information Theory **34**(5), 1354–1359 (1988)

[12] Massey, J.: Shift-register synthesis and bch decoding. IEEE Transactions on Information Theory **15**(1), 122–127 (1969). https://doi.org/10.1109/TIT.1969.1054260

[13] May, A., Meurer, A., Thomae, E.: Decoding random linear codes in $o(2^{0.054n})$. In: Lee, D.H., Wang, X. (eds.) Advances in Cryptology – ASIACRYPT 2011. pp. 107–124. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)

[14] May, A., Ozerov, I.: On computing nearest neighbors with applications to decoding of binary linear codes. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology – EUROCRYPT 2015. pp. 203–228. Springer Berlin Heidelberg, Berlin, Heidelberg (2015)

[15] McEliece, R.J.: A Public-Key Cryptosystem Based On Algebraic Coding Theory. Deep Space Network Progress Report **44**, 114–116 (Jan 1978)

[16] Minder, L., Shokrollahi, A.: Cryptanalysis of the sidelnikov cryptosystem. In: Naor, M. (ed.) Advances in Cryptology - EUROCRYPT 2007. pp. 347–360. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)

[17] Niederreiter, H.: Knapsack-type cryptosystems and algebraic coding theory. In: Problems of Control and Information Theory 15. pp. 159–166 (1986)

[18] Patterson, N.: The algebraic decoding of goppa codes. IEEE Trans. Inf. Theor. **21**(2), 203–207 (Sep 2006). https://doi.org/10.1109/TIT.1975.1055350, `http://dx.doi.org/10.1109/TIT.1975.1055350`

[19] Prange, E.: The use of information sets in decoding cyclic codes. IRE Transactions on Information Theory **8**(5), 5–9 (1962)

[20] SIDELNIKOV, V.M., SHESTAKOV, S.O.: On insecurity of cryptosystems based on generalized reed-solomon codes. Discrete Mathematics and Applications **2**(4), 439 – 444 (1992). https://doi.org/https://doi.org/10.1515/dma.1992.2.4.439, `https://www.degruyter.com/view/journals/dma/2/4/article-p439.xml`

[21] Stern, J.: A method for finding codewords of small weight. In: Cohen, G., Wolfmann, J. (eds.) Coding Theory and Applications. pp. 106–113. Springer Berlin Heidelberg, Berlin, Heidelberg (1989)

[22] Targhi, E.E., Unruh, D.: Post-quantum security of the fujisaki-okamoto and oaep transforms. In: Hirt, M., Smith, A. (eds.) Theory of Cryptography. pp. 192–216. Springer Berlin Heidelberg, Berlin, Heidelberg (2016)

Appendix A

SAGE code for a Binary Separable Goppa code used in **PALOMA**

```
'''

Copyright 2022 FDL(Future cryptograph Design Laboratory, Kookmin University

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:
The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

'''
###################################################
'''
    Binary Separable Goppa Code used in PALOMA
    developed by FDL/KMU
'''
###################################################

'''
    F2m = GF(2^13) (i.e., m = 13)
    Separable Goppa Polymoial g(X) with degree t in F2m[X] (t-error collectable code)

    n + t <= q^m = 2^13 = 8192
    k >= n - mt = n - 13t

    parameters:
    PALOMA128: n = 3904(61), k = 3072, n-k = 832(13), m = 13, t = 64
    PALOMA192: n = 5568(87), k = 3904, n-k = 1664(26), m = 13, t = 128
    PALOMA256: n = 6592(103), k = 4928, n-k = 1664(26), m = 13, t = 128

    Toy parameters:
    n = 37, k = 19, n-k = 18, t = 3, m = 6, f = z^6 + z^4 + z^3 + z + 1
    n = 100, k = 72, n-k = 28, t = 4, m = 7, f = z^7 + z + 1
    n = 120, k = 64, n-k = 56, t = 8, m = 7, f = z^7 + z + 1
    n = 241, k = 121, n-k = 120, t = 15, m = 8, f = z^8 + z^4 + z^3 + z^2 + 1
```

```

    n = 53, k = 27, n-k = 26, t = 2, m = 13, f = z^13 + z^7 + z^6 + z^5 + z^0
    n = 79, k = 40, n-k = 39, t = 3, m = 13, f = z^13 + z^7 + z^6 + z^5 + z^0

'''
##################################################

reset()
var('z')

##################################################

def line():
    print ("\n====================================================")

def newline():
    print (" ")

line()

##################################################
# parameters: n, t, m, irr_poly
##################################################
paloma_param = [
    [37, 3, 6, z^6 + z^4 + z^3 + z + 1],
    [100, 4, 7, z^7 + z + 1],
    [120, 8, 7, z^7 + z + 1],
    [241, 15, 8, z^8 + z^4 + z^3 + z^2 + 1],

    [53, 2, 13, z^13 + z^7 + z^6 + z^5 + z^0],
    [79, 3, 13, z^13 + z^7 + z^6 + z^5 + z^0],

    [216, 8, 13, z^13 + z^7 + z^6 + z^5 + 1],
    [424, 16, 13, z^13 + z^7 + z^6 + z^5 + 1],

    [3904, 64, 13, z^13 + z^7 + z^6 + z^5 + 1],
    [5568, 128, 13, z^13 + z^7 + z^6 + z^5 + 1],
    [6592, 128, 13, z^13 + z^7 + z^6 + z^5 + 1],
]

n, t, m, f = paloma_param[8]
k = n - m*t

##################################################

R2.<z> = GF(2)[]
F2m.<z> = GF(2^m, modulus = R2(f))
R2m.<X> = PolynomialRing(F2m)

##################################################
# function for hex representation
##################################################

def str_f2m_hex(x):
    return "0x{:04x}".format(ZZ(list(F2m(x).polynomial()), base = 2))
#    return hex(ZZ(list(F2m(x).polynomial()), base = 2))

def show_mat_hex(m):
    nrows, ncols = m.nrows(), m.ncols()
    for r in range(0, nrows):
        str = "[ "
        for c in range(0, ncols):
            str += str_f2m_hex(m[r][c]) + " "
        print (str, "]")

def show_poly_hex(f):
    show_mat_hex(matrix(list(f)))
```

```

##################################################
# Generate Random Binary Separable Goppa Code
##################################################

print ("Random Binary Separable Goppa Code")
print ("n = {}({}), n-k = {}({}), t = {}, m = {}".format(n, n/64, n-k, (n-k)/64, t, m))
newline()

##################################################

listF2m = list(F2m)
mbitset = list(range(0,2^m,1))

##################################################
# Generate Support Set L and Separable Goppa polynomial g(X)
##################################################

# shuffle(mbitset)

'''
    Support set L
'''
L = [listF2m[j] for j in mbitset[:n]]
print ("Support Set L")
print (L)
show_mat_hex(matrix(L))
line()

'''
    Separable Goppa polynomial g(X)
'''
g = prod([(X+listF2m[j]) for j in mbitset[n:n+t]])
print ("Goppa Poly. g(X)")
print (g)
show_poly_hex(g)
print("roots = ", [listF2m[j] for j in mbitset[n:n+t]])
line()

##################################################
# Compute Parity-check Matrix H = A*B*C
##################################################

'''
    Matrix A
'''
coeffg = list(g) + [0]*(t-1)
A = matrix([coeffg[i:i+t] for i in [1..t]])

print ("\nA")
#print (A)
show_mat_hex(A)
newline()


'''
    Matrix B*C
'''
time B = matrix(F2m, t, n, lambda r, c: (L[c]^r))
print ("Parity-check Matrix H = B")
#print (B)
show_mat_hex(B)
newline()

#T1 = [g(L[c]) for c in range(0,n)]
#T2 = [g(L[c])^-1 for c in range(0,n)]
#print("T1: ", T1)
```

```
#print("T2: ", T2)

time BC = matrix(F2m, t, n, lambda r, c: (L[c]^r) * (g(L[c])^-1) )

print ("\nParity-check Matrix H = BC")
show_mat_hex(BC)
#print (BC)
newline()

time H = A*BC
print ("\nParity-check Matrix H = ABC")
#print (H)
show_mat_hex(H)
newline()

'''
    Parity-check matrix derived from (X-aj)^-1
'''
'''
H1 = []
for i in [0..n-1]:
    inv = R2m((g - g(L[i]))/(X-L[i]))*g(L[i])^-1
    H1 += [list(inv)]

H1 = Matrix(F2m, H1).transpose()
print("H1 == H?", H1 == H)
'''

###################################################
# Modified Patterson Decoding for Binary Separable Goppa Code
###################################################

'''
    Given f s.t gcd(f,g),
    find f^-1 such that f^-1*f = 1 (mod g)
'''
def getInv(f, g):
    t = g.degree()
    d0, d1 = R2m(f), R2m(g)
    a0, a1 = R2m(1), R2m(0)

    while d1 != 0:
        r = d0%d1
        q = R2m((d0 - r)/d1)
        d0, d1 = d1, r
        a2 = a0 - q*a1
        a0, a1 = a1, a2

    return a0*d0.leading_coefficient()^-1

###################################################

'''
    Find a2, b1 such that b1*s_hat = a2 (mod g12) with deg condition
'''
def EEA_for_keyeqn(s_hat, g12, dega, degb):
    a0, a1 = R2m(s_hat), R2m(g12)
    b0, b1 = R2m(1), R2m(0)

    while a1 != 0:
        q, r = a0.quo_rem(a1)
        a0, a1 = a1, r
        b2 = b0 - q*b1
        b0, b1 = b1, b2
        if a0.degree() <= dega and b0.degree() <= degb:
            break
    return a0, b0
```

```
##################################################

'''
    Compute Square Root of f(X) mod g12(X)
'''

def get_sqrt(f, g):
    sqrtx = power_mod(R2m(X), 2^(m*t-1), g)
    print("sqrtx^2%g == X?", sqrtx^2%g == X)
    print("sqrt(X) mod g12 =", sqrtx)
    degf = R2m(f).degree()
    listf = list(f)
    fe = [sqrt(listf[2*j]) for j in [0..floor(degf/2)]]
    fo = [sqrt(listf[2*j+1]) for j in [0..floor((degf-1)/2)]]

    sqrtf = (R2m(fe) + R2m(fo)*sqrtx)%g
    return sqrtf

##################################################

'''
    Given f, find a(X), b(X) such that f = a^2(X) + b^2(X)*X
'''
def get_a2b2x(f):
    degf = R2m(f).degree()
    listf = list(f)
    fe = [sqrt(listf[2*j]) for j in [0..floor(degf/2)]]
    fo = [sqrt(listf[2*j+1]) for j in [0..floor((degf-1)/2)]]
    a = R2m(fe)
    b = R2m(fo)
    return a, b

##################################################

line()

##################################################
# Step 0. Generate Random Error Vector with Hamming Weight t
##################################################

nset = list(range(0,n))
shuffle(nset)

e = [0]*n
for i in nset[0:t]:
    e[i] = 1
#print ("Error vector e\n", e)
print ("Error Polynomial e(X) =", R2m(e))
line ()

'''
    error locator polynomial sigma_t = a_t^2 + b_t^2*X for checking correctness
'''
sigma_t = R2m(1)
for i in range(0,n):
    if e[i] == 1:
        sigma_t = sigma_t * (X + L[i])

a_t, b_t = get_a2b2x(sigma_t)

print("sigma_t(X) =", sigma_t)
print("a_t(X) =", a_t)
print("b_t(X) =", b_t)
print("R2m(a_t^2 + b_t^2*X) == sigma_t?", R2m(a_t^2 + b_t^2*X) == sigma_t)
newline()
```

```
##################################################
# Step 1. Compute Syndrome s(X) of e(X)
##################################################

He = H * vector(e)
s = R2m(list(He))

'''
H1e = H1 * vector(e)
print("He == H1e?", He == H1e)
'''

print("s(X)     =", s)

'''
    Checking Correctness
'''
syndrome = R2m(0)
for i in [0..n-1]:
    syndrome += e[i]*R2m((g - g(L[i]))/(X-L[i]))*g(L[i])^-1
print("syndrome =", syndrome)
print("s(X) == syndrome?", s == syndrome)

##################################################
# Step 2. Find Error Locator Polynomial sigma(X)
##################################################

'''
    Checking Correctness
'''
print("sigma_t*s%g == sigma_t.derivative()?", sigma_t*s%g == sigma_t.derivative())
newline()

##################################################

'''
    Derive Key Equation
'''
s_ast = R2m(1) + X*s(X)
g1 = gcd(g, s)
g2 = gcd(g, s_ast)
g12 = R2m(g/g1/g2)
s2_ast = R2m(s_ast/g2)
s1 = R2m(s/g1)

u = (g1 * s2_ast * getInv(g2*s1, g12))%g12
print("g2*s1*getInv(g2*s1, g12)%g12 == 1?", g2*s1*getInv(g2*s1, g12)%g12 == 1)

s_hat = get_sqrt(R2m(u), R2m(g12))

print("s_hat^2%g12 == u?", s_hat^2%g12 == u)

'''
    Solve Key Equation
'''

a2, b1 = EEA_for_keyeqn(s_hat, g12, floor(t/2)-g2.degree(), floor((t-1)/2)-g1.degree())
print ("b1*s_hat%g12 == a2?", b1*s_hat%g12 == a2)

'''
    Compute a, b
'''

a = a2*g2
b = b1*g1
print("a(X) =", a)
print("b(X) =", b)
```

```
'''
    Checking Correctness
'''
print ("b^2*s_ast%g == a^2*s%g?", b^2*s_ast%g == a^2*s%g)
print ("b^2*(1+X*s)%g == a^2*s%g?", b^2*(1+X*s)%g == a^2*s%g)


sigma = (a^2 + b^2*X).monic()
print ("sigma == sigma_t?", sigma == sigma_t)

##################################################
# Step 3. Find Roots of sigma(X)
##################################################


err_support_set = []
for i in [0..n-1]:
    if sigma(L[i]) == 0:
        err_support_set += [i]
print("recovered supp(e) =", err_support_set)
line()

##################################################
'''
    Result
'''

print ("\nDo we find the correct error?", err_support_set == R2m(e).exponents())
line()
```