IPCC7: Post-Quantum Encryption scheme based on Perfect Dominaitng Set in 3-regular Graph

Jieun Ryu[1], Yongbhin Kim[2], Seungtai Yoon[3], Ju-Sung Kang[1,2] and Yongjin Yeom[1,2]

Abstract— **In response to the rapid development of quantum computers, post-quantum cryptography is being actively studied. NIST, the international standardization organization, is conducting a competition to determine the standard for the next generation of cryptographic algorithms. This effort is intended to address the potential vulnerability of legacy public-key cryptosystems to quantum computers. While previous competitions have selected a single standard, this competition is considering cryptographic algorithms based on a variety of mathematical problems. This approach is taken because the mathematical and technical security of post-quantum cryptosystems (PQC) has not been fully studied as that of legacy cryptosystems. Indeed, SIKE, a recently proposed isogeny-based key-establishment algorithm, was succesfully attacked and subsequently eliminated in the middle of the third round of the competition. Thus, it is crucial to investigate cryptographic algorithms based on various problems because we cannot predict when and how PQC algorithms might be cracked. Hence, in this work we consceptualized and implemented the PCC (Perfect Code Cryptosystem), a PKE (Public Key Encryption) based on combinatorics.**

Our public key cryptosystem is predicated on the fact that when a graph with a PDS (Perfect Dominating Set) is used as the public key, it is challenging to find what PDS is in the graph. The PCC has not received much attention due to the disadvantage of incredibly slow encryption speed. In this work, we propose an enhanced algorithm to overcome the disadvantage. To increase the security of the cryptosystem, we adopted higher degree of the polynomial for the cyphertext, which rapidly slows down the speed of encryption and raises up the size of the cyphertext. We sought to address the problem by multiplying polynomianls of lower degrees. Thus, we show that our algorithm is distinguished from other PQC algorithms of similar public key size and ciphertext size by its large ciphertext and rapid operation. We anticipate that the PCC be more effectively utilized in specific fields such as white-box encoding.

Index Terms— **Perfect Code cryptosystems, PQC, Graph-based KEM, Perfect Dominaitng Set, Perfect Dominating Function.**

I. Introduction

Public key algorithms are utilized in a variety of electronic services today. They are used for online payments, symmetric key sharing, message decryption, and more. However, the rapid development of quantum computers in recent years has threatened the security of existing public key systems. In response, post-quantum cryptography (PQC) has been actively researched for quantum resistance. While PQC has only recently gained attention among nonprofessionals, it has been studied in the field of cryptography since the 70s, when modern cryptography began to be used. As research on PQC, which was not easy to implement due to the limitations of computing power, became more active, various cryptographic algorithms that were only proposed in theory based on different mathematical problems were refined and optimized.

The National Institute of Standards and Technology (NIST) has been standardizing post-quantum cryptography (PQC) for quantum computers through the Post-Quantum Cryptography Standardization since 2016, and many national institutions and companies have been working on PQC research. At this point, NIST has selected a total of four algorithms, including CRYSTALS-KYBER, as standards through the competition, and other four algorithms are competing as candidate algorithms [8].

Notably, the candidate algorithms utilize code-based problems that are different from the standardized algorithms based on the lattice and the hash. Unlike traditional public-key cryptography, which has been proven and attacked about underlying problems for hundreds of years, it is still too early to say whether PQC algorithms are robust. So, it seems that NIST's choice to study multiple PQCs that utilize different underlying problems is intended to prepare for future unexpected attacks on them. In fact, SIKE advanced to the third round of the competition before succumbing to a mathematical attack, leading to its elimination [22]. This likely occurred because SIKE had a shorter research horizon compared to other third-round algorithms. Given that we cannot predict when other PQC algorithms might be attacked like this, it becomes necessary to research and develop cryptographic algorithms based on different foundational problems and to continually evolve them over an extended period of time. The cryptosystem we studied is a public-key cryptosystem based on a novel basis problem, combinatorial theory, that is different from the mainstream algorithms, and can be used as a PQC based on the conjecture that finding a particular subset of the graph is an NP-hard problem [21].

The Perfect Code Cryptosystem, proposed by Koblitz in 1992, is a public-key algorithm with the very intuitive idea of constructing a ciphertext polynomial using the variables of the graph vertices and the constants mapped to the vertices [2], [3]. Algorithms that utilize polynomials to generate ciphertexts are already widely used in cryptosystems, and it is possible to generate a polynomial ciphertext by thinking of each vertex as having one variable on a graph with indexed vertices. The person who encrypts the message can generate a polynomial over the graph opened to the public by the person to be decrypted. In the coefficients of each term of the polynomial, it is scattered that meaningful and meaningless value about the message. Then, when the decryptor receives the ciphertext polynomial, he or she can find terms that consist only of vertices belonging to a certain subset of the graph he or she created. These terms have only meaningful

This work has supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2021M1A2A2043893)

[1]Dept. of Financial Information Security, Kookmin University, Seoul, Republic of Korea

[2]Dept. of information security cryptography mathematics, Kookmin University, Seoul, Republic of Korea

[3]Cold Spring Harbor, NY 11724, United States (e-mail: yoon@cshl.edu)

values, and the original message is obtained by gathering them together.

The meaningless value in the ciphertext makes it more secure against some attacks, but it also makes the ciphertext very large. In addition, this cryptosystem has not received much attention due to its extremely slow encryption speed. Nevertheless, to overcome these difficulties and to utilize the cryptosystem, several researchers have been working on the cryptosystem, including the underlying problem [4], [7]. Previous research has shown that the Perfect Code Cryptosystem becomes more secure as the degree of the ciphertext polynomial increases. However, this improvement comes at the cost of exceedingly slow encryption, and an excessively large polynomial due to the inability to control the number of nonzero terms. The simplest form of polynomial that this cryptosystem can generate is of a size that allows for relatively fast ciphertext generation [5]. Nonetheless, its highly constrained form renders it vulnerable to simple attacks. Consequently, our aim was to find a balance between the most comprehensive polynomial generation method proposed by Koblitz and the simplest form of polynomial generation method. By tuning some parameters, we got the size of the polynomial to be in a range that we could use as a ciphertext and improve the speed by enhancing the encryption process. We have measured the performance of our implemented cryptosystem for the proposed parameters to satisfy 128-bit security against known attacks on the Perfect Code Cryptosystem. The measured encryption time is 2.34s, which can be used in the field.

On the other hand, this cryptosystem has a public key size of 768 bytes and a ciphertext size of 235,818 bytes on average, which is about 300 times the size difference between the two. These characteristics distinguish it from other PQC algorithms submitted to the NIST competition, which have similar public key and ciphertext sizes. When comparing these characteristics, Perfect Code Cryptosystem would be located in the empty area in the upper left corner of the graph, as shown in Fig. 1. These unique properties can be expected to contribute to making it more suitable for one-way functions or white-box encoding than other PQCs.

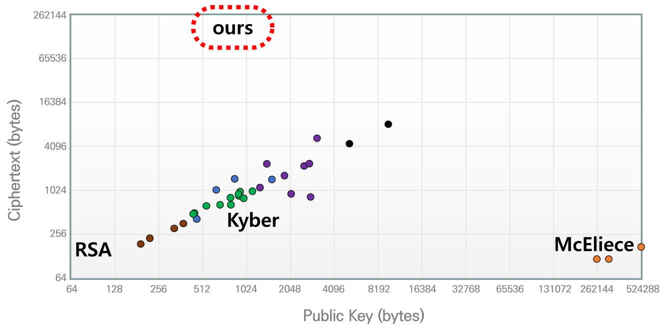


Fig. 1. The property about the size of the public key and ciphertext [11]

II. Preliminary

Let $G = (V, E)$ be a graph G consisting of a set of vertices V and a set of edges $E(\subseteq V \times V)$.

A. Mathematical Background

Definition 1. (Closed neighborhood) For a vertex $v \in V$ of a graph $G = (V, E)$, the *closed neighborhood* of v is the set of vertices consisting of v and its adjacent vertices. The closed neighborhood of vertex v can be denoted as follows:

$$N[v] = \{u \in V \mid uv \in E\}.$$

Example 1. Fig. 2 represents a graph with vertices $V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$. In this graph, all the closed neighborhoods are as follows::

$$\begin{array}{ll} N[v_1] = \{v_1, v_2, v_5\}, & N[v_5] = \{v_1, v_4, v_5, v_6\}, \\ N[v_2] = \{v_1, v_2, v_3, v_6, v_7\}, & N[v_6] = \{v_2, v_5, v_6, v_7\}, \\ N[v_3] = \{v_2, v_3, v_4\}, & N[v_7] = \{v_2, v_6, v_7\}. \\ N[v_4] = \{v_3, v_4, v_5\}, & \end{array}$$

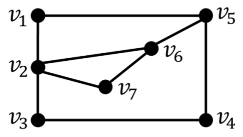


Fig. 2. Example graph

Definition 2. (r-regular graph) For some positive integer k, a simple graph $G = (V, E)$ is called a *r-regular graph* if it satisfies the following:

$$\forall v \in V, \ |N(v)| = r + 1.$$

Example 2. Fig. 3 is a graph with vertices $V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$. For each vertex $v \in V$ in this graph, the closed neighborhood of v is defined as follows:

$$\begin{array}{ll} N[v_1] = \{v_1, v_2, v_4, v_6\}, & N[v_4] = \{v_1, v_3, v_4, v_5\}, \\ N[v_2] = \{v_1, v_2, v_3, v_6\}, & N[v_5] = \{v_3, v_4, v_5, v_6\}, \\ N[v_3] = \{v_2, v_3, v_4, v_5\}, & N[v_6] = \{v_1, v_2, v_5, v_6\}. \end{array}$$

This graph is a 3-regular graph because $|N[v]| = 4$ for all vertices $v \in V$ in the graph.

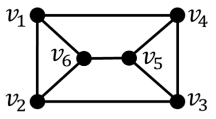


Fig. 3. example of 3-regular graph

Property 1. The number of edges in r-regular graph consisting by n vertices is $nr/2$.

Definition 3. (Perfect Dominating Set, PDS) Given graph $G = (V, E)$, subset $D \subseteq V$ is called a *perfect dominating set (PDS)* of G if $N[v]$ is contains exactly one element of D for every vertex $v \in V$ (i.e., every vertex has exactly one neighbor in D), and we denote $D \in PDS(G)$, where

$$\forall v \in V, \ |N[v] \cup D| = 1$$

Example 3. Fig. 4 represents a graph with vertices $V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$. In this graph, the closed neighborhood of each vertex $v \in V$ is defined as follows:

$$\begin{array}{ll} N[v_1] = \{v_1, v_2, v_4, v_6\}, & N[v_5] = \{v_4, v_5, v_6, v_8\}, \\ N[v_2] = \{v_1, v_2, v_3, v_7\}, & N[v_6] = \{v_1, v_5, v_6, v_7\}, \\ N[v_3] = \{v_2, v_3, v_4, v_8\}, & N[v_7] = \{v_2, v_6, v_7, v_8\}, \\ N[v_4] = \{v_1, v_3, v_4, v_5\}, & N[v_8] = \{v_3, v_5, v_7, v_8\}. \end{array}$$

Let $D = \{v_1, v_8\}$. The intersection of each set $N[v]$ for $v \in V$, with D is given by

$$\begin{aligned} N[v_1]\cap D &= \{v_1\}, & N[v_5]\cap D &= \{v_8\},\\ N[v_2]\cap D &= \{v_1\}, & N[v_6]\cap D &= \{v_1\},\\ N[v_3]\cap D &= \{v_8\}, & N[v_7]\cap D &= \{v_8\},\\ N[v_4]\cap D &= \{v_1\}, & N[v_8]\cap D &= \{v_8\}. \end{aligned}$$

Then, D is the PDS of the graph G because the size of the intersection of each set $N[v]$ with D is always 1.

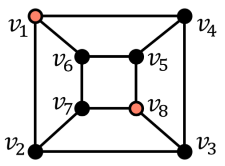


Fig. 4. example of 3-regular graph that has $\{v_1, v_8\} \in V$ as a PDS

A graph may or may not have any PDS, and when it does, there are one or more PDS. Note that the graph in Fig. 3 does not have PDS.

Property 2. The vertex set of r-regular graph G that has some PDSes consists of PDSes, and, if the number of vertices is n, the size of each PDS is $\frac{n}{r+1}$.

Proposition 1. *The distance between distinct vertices belonging to one PDS is always at least 3.*

Example 4. Fig..5 shows two different portions of the 3-regular graph. Assume that the vertices colored in red belong to the same PDS. In the left case, if you check the closed neighborhood set for all vertices, there is always only one red vertex in that set. This means that the vertex set consisting of the red-colored vertices satisfies the definition of a PDS.

However, in the right case, two red vertices appear in the closed neighborhood set of the central black vertex. This implies that these red vertices cannot form a PDS. In other words, if you assume that the distance between two different vertices belonging to the same PDS is less than or equal to 2, then the vertices within that range cannot satisfy the definition of a PDS.

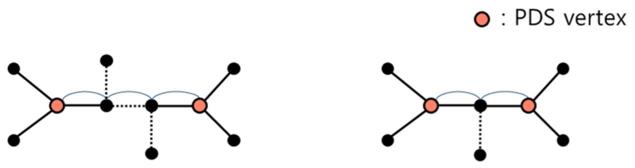


Fig. 5. distances between vertices and the values mapped to them

Definition 4. (Perfect Dominating Function, PDF) Given graph $G = (V, E)$, a function $\chi : V \to \{0, 1\}$ is called a *perfect dominating function* if it satisfies the following.

$$\forall v \in V,\ \textstyle\sum_{u\in N[v]} \chi(u) = 1.$$

In this case, we write $\chi \in \mathrm{PDF}(G)$.

Proposition 2. *Let x_v denote the variable assigned to the vertex $v \in V$. For a 3-regular graph $G = (V, E)$ with PDSes, $\chi_D(x_v)$ is a perfect dominating function if it maps the vertices within D to 1 and all remaining vertices to 0, where D is a randomly selected PDS.*

Example 5. Consider the graph in Fig. 4 that has PDSes. Let x_{v_i} be a variable for each $v_i \in V$, and $u_{i,j} \in N[v_i]$ for each v_i. And, Let the function $\chi(v_i)$ maps vertices within the PDS $\{v_1, v_8\}$ to 1 and all other vertices to 0.

For each vertex v_i, $e_i = \sum_{u\in N[v_i]} x_u$. Then, e_i for all vertices will be as follows:

$$\begin{aligned} e_1 &= x_{v_1}+x_{v_2}+x_{v_4}+x_{v_6}, & e_5 &= x_{v_4}+x_{v_5}+x_{v_6}+x_{v_8},\\ e_2 &= x_{v_1}+x_{v_2}+x_{v_3}+x_{v_7}, & e_6 &= x_{v_1}+x_{v_5}+x_{v_6}+x_{v_7},\\ e_3 &= x_{v_2}+x_{v_3}+x_{v_4}+x_{v_8}, & e_7 &= x_{v_2}+x_{v_6}+x_{v_7}+x_{v_8},\\ e_4 &= x_{v_1}+x_{v_3}+x_{v_4}+x_{v_5}, & e_8 &= x_{v_3}+x_{v_5}+x_{v_7}+x_{v_8}. \end{aligned}$$

All of the e_i values, denoted as $e_i(\chi(v_1), \cdots, \chi(v_8))$, satisfy 1 for $\chi(v_i)$, so $\chi \in \mathrm{PDF}(G)$.

Note that, in this example, regardless of which other PDS - $\{v_2, v_5\}, \{v_3, v_6\}, \{v_4, v_7\}$ - you choose to generate χ, $e_i(\chi(v_1), \cdots, \chi(v_8)) = 1$ is always satisfied.

Definition 5. (Invariant polynomial) Let, for $v_i \in V, i = 1, \cdots, n$ over the graph G with PDS, $\mathbb{Z}_p[x_{v_1}, x_{v_2}, \cdots, x_{v_n}]$ is polynomial ring, $x_{v_i} : \{0,1\}^n \to \{0,1\}$ is a i-th coordinate function such that $x_{v_i}(a_1, a_2, \cdots, a_n) \mapsto a_i$, and $\chi(v_i)$ is a PDF(G). For a constant c, a function $f : \{0,1\}^n \to \mathbb{Z}_p[\{x_{v_i}\}]$ is a *invariant polynomial* if it satisfies the following.

$$f(x_1(\chi(v_1), \cdots, \chi(v_n)), \cdots, x_n(\chi(v_1), \cdots, \chi(v_n))) = c.$$

Here, c is deterministic with respect to χ_D, and we denote this as $(f \circ \boldsymbol{\chi})(v_1, v_2, \cdots, v_n) = c$ or, more simply, as $f \circ \boldsymbol{\chi} = c$.

Our definition of the invariant polynomial follows that of Koblitz, leading to the subsequent proposition [2].

Proposition 3. *Given a graph $G = (V, E)$ with PDSes, for $i \in 1, 2, \cdots, |N[v]|$, if g_i is an arbitrary invariant polynomial of degree $k - 1$, and the different $g_i + c_i$ is evaluated to the same value for calculated constant c_i, then the following is an invariant polynomial of degree k for any vertex v.*

$$\textstyle\sum_{u\in N[v]}(g_u + c_u)x_u$$

Proposition 4. *Given a graph $G = (V, E)$ with a PDS, for $\chi_D \in PDF(G)$ that satisfies Proposition 2, consider $f \in \mathbb{Z}_p[x_1, x_2, \cdots, x_{|V|}]$ is an invariant polynomial over G, and x_v, x_u are variables for vertices $v, u \in V$. If these variables appear in the same term of f, then the following holds: $x_v x_u = x_v$ if $v = u$, and $x_v x_u = 0$ if $v \neq u$ and the distance between the two vertices is ≤ 2.*

This is established by Proposition 1. Throughout this paper, the process of polynomial transformation according to Proposition 4 will be referred to as reduction. The following example shows the process of generating an invariant polynomial through reduction.

Example 6. Consider the graph G shown in Fig. 4. We will construct an invariant polynomial of degree 2 for this graph.

Let $N[v_i] = \{u_{i,1}, u_{i,2}, u_{i,3}, u_{i,4}\}$ for each vertex $v_i \in V$. The simplest invariant polynomial of degree 1, denoted as e_i, is shown in the examples of Proposition 2.

Suppose f_j is an arbitrary invariant polynomial of degree 1, and c_j is a constant such that $(f_j \circ \boldsymbol{\chi}_D) + c_j$ has a constant value. Then we can generate an invariant polynomial of degree 2 for any vertex $v \in V$, denoted as f, as follows:

$$f = \textstyle\sum_{j=1}^{4}(f_j + c_j)x_{u_{i,j}}$$

The subsequent f is an invariant polynomial of degree 2 with a value of 3, generated specifically for v_1.

$$\begin{aligned}
f=&\textstyle\sum_{j=1}^{4}(f_j + c_j)x_{u_{1,j}}\\
=&(4e_2-1)x_{v_1}+(e_1+2e_5)x_{v_2}+(5e_8-2)x_{v_4}+(3e_6)x_{v_6}\\
=&\{4(x_{v_1}+\cancel{x}_{v_2}+\cancel{x}_{v_3}+\cancel{x}_{v_7})-1\}\,x_{v_1}\\
&+\{(\cancel{x}_{v_1}+x_{v_2}+\cancel{x}_{v_4}+\cancel{x}_{v_6})+2(\cancel{x}_{v_4}+x_{v_5}+\cancel{x}_{v_6}+\cancel{x}_{v_8})\}\,x_{v_2}\\
&+\{5(\cancel{x}_{v_3}+\cancel{x}_{v_5}+x_{v_7}+\cancel{x}_{v_8})-2\}\,x_{v_4}\\
&+3(\cancel{x}_{v_1}+\cancel{x}_{v_5}+x_{v_6}+\cancel{x}_{v_7})x_{v_6}\\
=&\ 3x_{v_1}+x_{v_2}-2x_{v_4}+3x_{v_6}+2x_{v_2}x_{v_5}+5x_{v_4}x_{v_7}
\end{aligned}$$

The subsequent f is an invariant polynomial of degree 2 with a value of 3, generated specifically for v_5.

$$\begin{aligned}
f=&\textstyle\sum_{j=1}^{4}(f_j + c_j)x_{u_{5,j}}\\
=&(4e_1-2e_7+1)x_{v_4}+(3e_7)x_{v_5}\\
&+(5e_1-2)x_{v_6}+(6e_2-2e_4-1)x_{v_8}\\
=&\ 5x_{v_4}+3x_{v_6}-x_{v_8}+4x_{v_1}x_{v_8}+3x_{v_2}x_{v_5}-2x_{v_4}x_{v_7}
\end{aligned}$$

B. Perfect code cryptosystem

It is proved that the problem of determining whether an arbitrary graph $G = (V, E)$ has PDSes is an NP-hard problem. It is conjectured that finding any PDS in a graph that has PDSes is also NP-hard [23]. These properties can be used to construct a public-key cryptosystem, where a graph with PDSes serves as the public key, ensuring the one-wayness of the cryptosystem, and a PDF created using one of the PDSes is used as the secret key, establishing the cryptosystem's trapdoor.

The Perfect Code Cryptosystem generates, delivers, and decrypts ciphertext using the property of invariant polynomials generated over a graph that has PDSes. The principle of using an invariant polynomial generated over the graph G as a ciphertext in a cryptosystem and decrypting it to obtain a message operates as follows:

Property 3. When defining $\chi_D \in PDF(G)$ using $D \in PDS(G)$ over a graph G with PDSes, by the definition of PDF, the evaluated value of $\sum_{u\in N[v]} x_u$ by χ_D is always 1. Consequently, even without knowing v for which $v \in \chi_D$, one can calculate c to satisfy $(g \circ \boldsymbol{\chi}_D) + c = m$ for an invariant polynomial g generated by multiplying or adding a number of $\sum_{u\in N[v]} x_u$. Moreover, if $(g_i \circ \boldsymbol{\chi}_D) + c_i = m$ for all $i = 1, 2, \cdots, |N[v]|$, then the evaluated value of $\sum_{u\in N[v]}(g_u + c_u)x_u$ is m. That implies that this expression can be understood as $m\sum_{u\in N[v]}x_u$.

According to Property 3, in the Perfect Code Cryptosystem, a sender who encrypts the message can generate a ciphertext in the same way, without any knowledge of the PDS. The result of evaluating the generated ciphertext polynomial will always have the same value as the original message. Moreover, even if a reduction is applied to the ciphertext invariant polynomial, the result of the polynomial still satisfies the properties of the invariant polynomial and retains the same value as the original message.

The construction of the cryptosystem involves key generation, encryption, and decryption processes, using a graph that has PDSes as the public key and the PDF, created from a PDS, as the secret key.

The overall flow of the cryptosystem is illustrated in Fig. 6. First, the receiver Bob generates a graph G with n vertices that has PDSes and selects D, one of the PDSes, to create a PDF χ_D. Bob stores χ_D as his secret key and shares the generated graph G with the sender Alice.

Alice, after receiving the public key, prepares a message m over the message space $\mathbb{Z}_p$ for encryption. The encryption algorithm generates a ciphertext invariant polynomial ct. The ciphertext invariant polynomial of degree k has the form $\sum_{u\in N[v]}(g_u + c_u)x_u$ for a randomly selected vertex $v \in V$. Here g_u is randomly selected from the set $\mathcal{F}_{k-1}$, which contains all invariant polynomials of degree $\leq k - 1$. The constant c_u can be computed such that $g_u + c_u = m$. The encryption process is completed by applying the reduction to this generated invariant polynomial.

Example 7. Using Example 6, suppose Alice has the message 3 and wants to encrypt it into an invariant polynomial of degree 2. Alice gets the public key, which is the graph depicted in Fig. 3. Although she doesn't know $\chi(v)$, Alice knows that $e_i \circ \boldsymbol{\chi} = 1$ is always satisfied for $e_i (i = 1, \cdots, 8)$, according to Example 5.

To generate the ciphertext, Alice initially selects a vertex v_5. Then, for each vertex in $N[v_5]$, specifically v_4, v_5, v_6, and v_8, she randomly selects a degree 1 invariant polynomial.

$$\begin{aligned}
g_{v_4} &= 4e_1 - 2e_7, & g_{v_6} &= 5e_1,\\
g_{v_5} &= 3e_7, & g_{v_8} &= 6e_2 - 2e_4.
\end{aligned}$$

Next, calculates a constant c_u, such that $(g_u \circ \boldsymbol{\chi}) + c_u = 3$.

$$\begin{aligned}
c_{v_4} &= 3 - (4 - 2) = 1, & c_{v_6} &= 3 - (5) = -2,\\
c_{v_5} &= 3 - (3) = 0, & c_{v_8} &= 3 - (6 - 2) = -1.
\end{aligned}$$

With these elements, she constructs the invariant polynomial ct of degree 2 as follows.

$$\begin{aligned}
ct=&(g_{v_4}+c_{v_4})x_{v_4}+(g_{v_5}+c_{v_5})x_{v_5}\\
&+(g_{v_6}+c_{v_6})x_{v_6}+(g_{v_8}+c_{v_8})x_{v_8}\\
=&(4e_1-2e_7+1)x_{v_4}+(3e_7)x_{v_5}\\
&+(5e_1-2)x_{v_6}+(6e_2-2e_4-1)x_{v_8}\\
=&\ 5x_{v_4}+3x_{v_6}-x_{v_8}+4x_{v_1}x_{v_8}+3x_{v_2}x_{v_5}-2x_{v_4}x_{v_7}
\end{aligned}$$

This is the result after the reduction has been applied.

Upon completion of the encryption process, Alice sends the ciphertext polynomial ct to Bob. Bob then executes the decryption process, substituting each variable in the polynomial with the corresponding value from the secret key χ_D to recover the original message. The result of evaluating the invariant polynomial is the original message m.

Example 8. Suppose Bob, who possesses the secret key corresponding to the public key graph in Fig. 3, receives ct in Example 7. Bob's secret key χ maps v_1, v_8 to 1 and all other vertices to 0. By substituting the value of χ into ct,

$$\begin{aligned}
&(ct(x_{v_1}, x_{v_2}, \cdots, x_{v_8}) \circ \boldsymbol{\chi})\,(v_1, v_2, \cdots, v_8)\\
&= ct(x_{v_1}(\chi(v_1), \cdots, \chi(v_8)), \cdots, x_{v_8}(\chi(v_1), \cdots, \chi(v_8)))\\
&= ct(x_{v_1}(1,0,0,0,0,0,0,1), \cdots, x_{v_8}(1,0,0,0,0,0,0,1))\\
&= 5\cdot 0 + 3\cdot 0 - 1 + 4\cdot 1\cdot 1 + 3\cdot 0\cdot 0 - 2\cdot 0\cdot 0\\
&= -1 + 4 = 3.
\end{aligned}$$

Thus, Bob can decrypt the message without knowing which vertices or invariant polynomials Alice selected.

The Perfect Code Cryptosystem is especially valuable in environments where decryption is used more frequently than encryption because the decryption process is very compact and fast. However, the generation of the invariant polynomial presents a challenge due to its highly memory and time requirements.

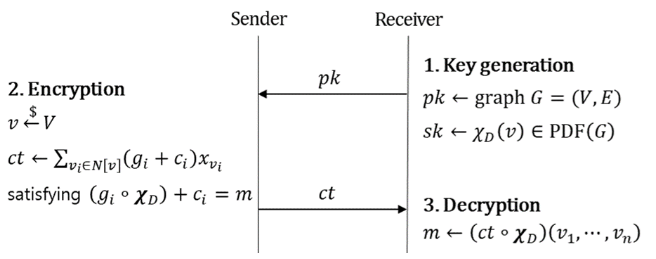

Sender
Receiver
1. Key generation
$pk \leftarrow$ graph $G = (V, E)$
$sk \leftarrow \chi_D(v) \in \mathrm{PDF}(G)$
pk
2. Encryption
$v \overset{\$}{\leftarrow} V$
$ct \leftarrow \sum_{v_i \in N[v]} (g_i + c_i) x_{v_i}$
satisfying $(g_i \circ \chi_D) + c_i = m$
ct
3. Decryption
$m \leftarrow (ct \circ \chi_D)(v_1, \cdots, v_n)$

Fig. 6. Perfect Code Cryptosystem flow chart

Fortunately, the memory issue can be practically mitigated through the application of the reduction. During the encryption process, the reduction eliminates about half of the terms in the polynomial, and terms with the same monomial combine into a single term. This greatly reduces the number of terms in the polynomial. From a cryptosystem perspective, this not only enhances the system more manageable from an implementation standpoint, but also makes it more difficult to attack a ciphertext polynomial by factoring it, enhancing the security of the cryptosystem.

Apart from the issue of storing already generated ciphertext, there exists another memory-related issue. The use of the cryptosystem with $\mathcal{F}_{k-1}$, which requires an overwhelming amount of memory for ciphertext generation, is very inefficient. To overcome this problem, Koblitz proposed a method to generate a invariant polynomial recursively.

III. The Proposed cryptosystem

In this section, we propose a post-quantum encryption scheme IPCC7; Improved Perfect Code Cryptosystem with degree 7 polynomials. and propose an improved encryption algorithm to enhance the system's performance.

Before handling these topics, we first clarify the notations commonly used throughout this paper. Here, m denotes the message to be encrypted. ct represents the ciphertext, which takes the form of a polynomial. f_k is used to indicate an arbitrary invariant polynomial of degree k or less.

A. Parameters

The IPCC7 uses a 3-regular graph with PDSes as the public key. The secret key is generated in the form of the PDF χ_{D_i} by selecting a set D_i $(i \in \{1, 2, 3, 4\})$ from one of the four distinct PDS D_1, D_2, D_3, D_4 of this graph.

$$\chi_{D_i}(v) = \begin{cases} 1, & \text{if } v \in D_i \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

Besides the key, several other parameters are shared for security.

- p is a parameter that determines the message space $\mathbb{Z}_p$, and it can be a composite number.
- n is the number of vertices in the graph (i.e., the size of the vertex set, $n = |V|$).
- k is the maximum degree of the ciphertext polynomial to be sent by the sender to the receiver.
- k' is the maximum degree of the subpolynomial generated in the encryption algorithm procedure.
- n_e is the parameter used for generating invariant polynomials of degree 1.

TABLE I
SECURITY PARAMETER FOR PROPOSED CRYPTOSYSTEM

parameter	p	n	k	k'	n_e
value	2^{32}	256	7	$2, 3, 4$	3

The following parameter set, which is provided for implementation, is chosen to ensure 128-bit security.

The algorithm receives the information about each parameter through the parameter set $D = \{p, n, k, k', n_e\}$. The rationale behind these specific parameter choices will be discussed in the Chapter IV.

B. Algorithms

Values reflecting the properties of regular graphs are used in various parts of the algorithm. Specifically, we frequently utilize the fact that the number of closed neighborhoods of a vertex is 4 for all vertices in a 3-regular graph.

We use the notation $\overset{\$}{\longleftarrow}$ for the random selection of an element from a set. For example, $v \overset{\$}{\longleftarrow} V$ where v is an element of V. Similarly, $U \underset{i}{\overset{\$}{\longleftarrow}} V$ means that the set U is a randomly selected subset of the set V with size, i.e., $U \subseteq V$.

The cryptosystem is composed of key generation, encryption, and decryption algorithms.

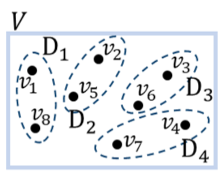
1) KEY GENERATION

The key generation algorithm is a process that the receiver in a cryptographic communication uses to generate a 3-regular graph with PDSes. Since the 3-regular graph used as the public key is a simple-regular graph, the receiver generates the public key by assigning six random one-to-one correspondences to the four pairs of PDSes.

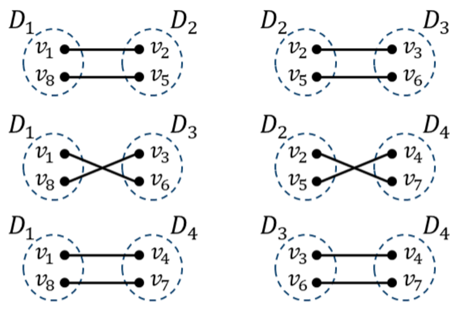
Algorithm 1 KeyGeneration

```
Input: The security parameters $(n, k)$ and modulus $p$
Output: A pair of keys $(pk, sk)$
1: Divide a set $V$ into 4 subsets which are called
   $D_1, D_2, D_3$,and $D_4$ such that
   $|D_1| = |D_2| = |D_3| = |D_4| = \frac{n}{4}$
2: Randomly create six one-to-one correspondences
   between the sets and connect related vertices each.
   Then the result is generated as the graph $G = (V, E)$
   with 4 PDSes.
3: Check whether the generated graph is connected. If
   not, repeat step 2.
4: WLOG set $D_1$ is used as $PDS$, and assign a value 0
   or 1 to each vertex $v \in V$ using the map
   $\chi_{D_1} \in \mathrm{PDF}(G)$, mapping $\chi_{D_1} : V \to \{0, 1\}$ by
   $\chi_{D_1}(x_v) = \begin{cases} 1, & \text{if } v \in D_1 \\ 0, & \text{otherwise} \end{cases}$
5: $pk \leftarrow G = (V, E)$ and $sk \leftarrow \chi_{D_1}$
6: return $pk, sk$
```

Example 9. The procedure for generating the graph in Fig. 3 using Algorithm 1 is as follows: First, Divide a set $V = \{x_1, x_2, \cdots, x_8\}$ into 4 subsets.

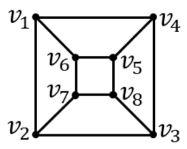


Then, create six random one-to-one correspondences between the subsets.

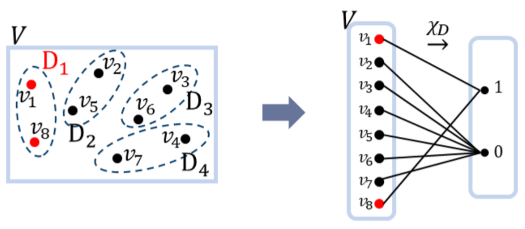


D_1 D_2 v_1 v_8 v_2 v_5
D_2 D_3 v_2 v_5 v_3 v_6
D_1 D_3 v_1 v_8 v_3 v_6
D_2 D_4 v_2 v_5 v_4 v_7
D_1 D_4 v_1 v_8 v_4 v_7
D_3 D_4 v_3 v_6 v_4 v_7

Subsequently, check whehter the graph generated by these correspondences is connected. In this example, the generated graph is connected. Hence, the sender uses this graph as a public key.



Finally, generate a secret key χ_{D_1} with D_1, one of the divided subsets among D_1, D_2, D_3, D_4.



2) ENCRYPTION

The encryption algorithm is a process that generates an invariant polynomial, which results in the value of the message m when evaluated using χ_{D_1}, even if the encryptor does not have knowledge of χ_{D_1}.

```
Algorithm 2 One of the encryption methods
Input: graph $G = (V, E)$, parameter set $D$, message $m$
Output: invariant polynomial $f$ of the degree $k$
1: $m' \xleftarrow{\$} \mathbb{Z}_p$
2: $m'' \leftarrow m/m' \pmod p$
3: $k' \leftarrow \lfloor k/2 \rfloor$
4: $k'' \leftarrow k - k'$
5: $f'_{k'} \leftarrow$ GenPolyDk($G$, GRT, , $k'$, $m'$ )
6: $f''_{k''} \leftarrow$ GenPolyDk($G$, GRT, , $k''$, $m''$ )
7: $f_k \leftarrow f'_{k'} \times f''_{k''}$
8: Reduction($f_k$)
9: return $f_k$
```

Algorithm 6 is illustrated in Fig. 7, and the detailed structure of the hiding procedure is shown in Fig. 8.

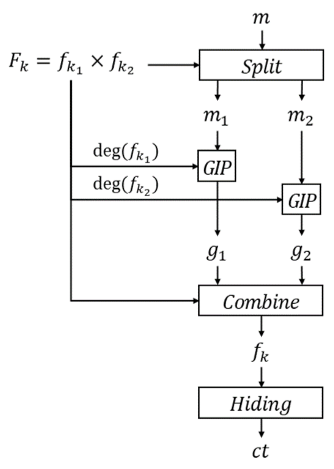


Fig. 7. Encryption flow

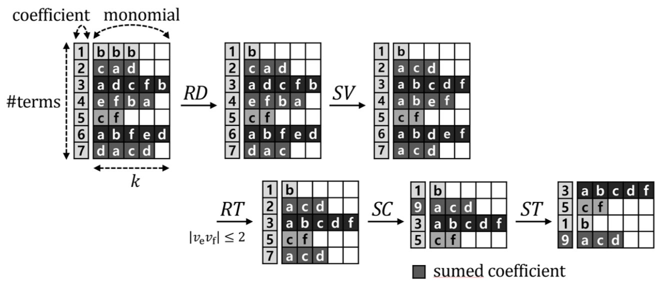


Fig. 8. Hiding procedure

In Fig. 8., a, b, c, d, e, f denote different vertex variables x_v.

GIP is a function that generates an Invariant Polynomial, and *RD* (Reduce Degree), *RT* (Reduce Terms), and *SC* (Sum Coefficients) are functions that organize the reduction process. The *SV* (Sort Variables) and *ST* (Shuffle Terms) processes prevent tracing of the polynomial generation process, making it impossible to perform attacks that factorize the ciphertext. The specific algorithm for hiding is described in APPENDIX C.

```
Algorithm 3 EncDeg1
Input: graph $G = (V, E)$, degree 1, message $m$
Output: invariant polynomial $f$ of the degree 1
1: Set the ciphertext polynomial $f_1 = 0$ and temporary
   value $sum = 0$
2: Select the number of vertices $n_v$ from the set
   $\{1, \cdots, n_e\}$ and select distinct $n_v$ vertices from $V$
        $n_v \xleftarrow{\$} \{1, \cdots, n_e\},$
        $\{v_1, \cdots, v_{n_v}\} \xleftarrow[n_v]{\$} V$
3: For each $v_j (j = 1, \cdots n_v)$ that is selected in Step 2
   except the last one, choose a random constant value
   $const$ on $\mathbb{Z}_p$ and add it into $sum$ then generate the
   invariant polynomial $const \sum_{u \in N[v]} x_u$ and add it into
   $f_1$, i.e.,
        $sum \leftarrow sum + const$ on $\mathbb{Z}_p$
```

```
            $f_1 \leftarrow f_1 + const \sum_{u \in N[v]} x_u$
4: For the last one vertex, calculate the value $m - sum$
   on $\mathbb{Z}_p$ as $const$ then generate the invariant polynomial
   $const \sum_{u \in N[v]} x_u$ and add it into $f_1$, i.e.,
            $f_1 \leftarrow f_1 + (m - sum) \sum_{u \in N[v]} x_u$
5: return $f_1$
```

Algorithm 4 EncDegK

```
Input: graph $G = (V, E)$, degree $k'$, message $m$
Output: invariant polynomial $f_{k'}$ of the degree $k'$
1: If $k' = 1$, return $f_{k'} \leftarrow$ EncDeg1($G$, $k'$, $m$)
2: Set the ciphertext polynomial $f_{k'} = 0$ and select a
   random vertex $v$ form $V$
3: For each $u \in N[v]$, choose the random fragment
   $m' \xleftarrow{\$} \mathbb{Z}_p$ then calculate $const = m - m' \pmod p$ and
   generate $f_{k'-1}$ by re-executing this algorithm
   recursively with input parameter $G$, $k' - 1$, $m'$ then
   add the term $(f_{k'-1} + const)x_u$ into $f_k$, i.e.,
            $f_{k'-1} \leftarrow \text{EncDegK}(G, k' - 1, m')$
            $f_{k'} \quad \leftarrow f_{k'} + (f_{k'-1} + const)x_u$
4: return $f_{k'}$
```

From the perspective of anyone other than the sender, an invariant polynomial generated by the encryption algorithm can represent a ciphertext polynomial as follows. Where $|ct|$ is the number of terms in the ciphertext, the i-th term is denoted $term_i$, and the coefficient of the i-th term is denoted c_i, the ciphertext takes the form

$$ct = \sum_{i=1}^{|ct|} term_i = \sum_{i=1}^{|ct|} c_i \prod x_v. \qquad (2)$$

3) DECRYPTION

The decryption algorithm is remarkably simple. The receiver can obtain the message by substituting the variable in Equation (2) with the corresponding value of χ_{D_1}. In essence, decryption is a process of performing $(f_k \circ \chi_{D_1})(v_1, \cdots, v_n)$, where $\chi_{D_1}(v_i)$ replaces v_i with 0 or 1 according to the Equation (1).

Algorithm 5 Decryption

```
Input: Private key $sk(\chi_{D_1})$, ciphertext $ct$
Output: message $m$
1: Evaluate the polynomial $ct$ using the private key $sk$
            $m \leftarrow (ct \circ \chi_{D_1})(v_1, \cdots, v_n)$
2: return $m$
```

C. Design rationale

Since the Perfect Code Cryptosystem's characteristic of having a large ciphertext can be utilized in certain roles such as a one-way function, we aimed to enhance encryption speed when designing the new cryptographic algorithm. Unlike encryption speed, key generation and decryption speed are significantly fast, so improving encryption speed would be particularly useful in environments where the decryption needs to be performed much more frequently than encryption.

The decrease in encryption speed in cryptosystems is a fundamental problem that emerges when the degree of the ciphertext polynomial is increased. To mitigate this, we sought to enhance the encryption speed by leveraging low-degree polynomials to generate a high-degree polynomial. As a result, while the ciphertext is generated at a similar speed to a low-degree polynomial, an attacker would need to exert effort equivalent to breaking a high-degree polynomial. Since the encryption speed decreases almost as a square root when the maximum degree is halved, it is feasible to reduce the encryption speed while preserving the security of the cryptosystem by utilizing low-degree polynomials to generate high-degree polynomial.

1) IMPLEMENTATION IDEA

A k-degree invariant polynomial on a 3-regular graph with PDSes can be generated by utilizing a $k-1$-degree invariant polynomial, as described in Property 3. In theory, to cover all invariant polynomials, we could pre-generate and store all invariant polynomials of degree $\leq k-1$ in the set $\mathfrak{F}_{k-1}$.

However, randomly selecting a polynomial from this set each time encryption is performed is inefficient in terms of memory utilization and hence, is not suitable for practical implementation. Therefore, the encryption algorithm uses a recursive approach to generate an invariant polynomial of degree k the form $\sum_{i=1}^{|N[u]|}(g_i + c_i)y_i,\ u \in V$. Here, g_i is an invariant polynomial of degree $k-1$ generated recursively such that x_v does not contain any variables of $v \in N[u]$, and c_i is a constant belonging to $\mathbb{Z}_p$.

However, when generating ciphertext polynomials in this manner, it is not possible to cover all conceivable invariant polynomials over the graph G. Nonetheless, since there are $O\left(n^{3k}/k^{2k}\right)$ (where $n \gg k$) invariant polynomials of degree k that can be generated recursively, we can expect this to be a sufficient burden for an attacker.

On the other hand, even with a recursive approach, the computational burden of encryption escalates drastically as the maximum degree of the polynomial increases. For the algorithm's usability, we do not simply generate a high-degree polynomial recursively, but instead perform the encryption by recursively generating several low-degree subpolynomials and combining them to form a high-degree polynomial.

2) MAIN IDEA

The primary concept of the proposed encryption algorithm is to generate a low-degree polynomial by fragmenting the message, as demonstrated in Algorithm 9. Our goal is to bring a randomized combinatorial method into the process of splitting and reassembling messages. The algorithm that summarizes this idea is presented below. $\mathcal{F}_k$ is a family of methods that combines subpolynomials of degree $\leq k-1$ so that the resulting polynomial has a degree of at most k. We denote by F a polynomial combination method randomly selected from $\mathcal{F}_k$, represented as $F(f_1, f_2, \cdots)$.

Algorithm 6 Encryption

```
Input: graph $G = (V, E)$, parameter set $D$, message $m$
Output: ciphertext $ct$
1: Select the combination method $F$ from $\mathcal{F}_k$
2: Split the message into random message pieces
```

```
    $m_1, m_2, \cdots$ according to the form of $F(k_1.k_2, \cdots)$
    such that $F(m_1, m_2, \cdots) = m$
3: For each message $m_i$, execute the algorithm 4 and
    store the result as $g_i$, i.e.,
        $g_i \leftarrow \text{EncDegK}(G, k_i, m_i)$
4: Combine the $g_i$ into $f_k$ according to the form of $F$
5: Apply the Hiding procedure, which includes the
    reduction, to $f_k$
6: $ct \leftarrow f_k$
7: return $ct$
```

Algorithm 6 is illustrated in Fig. 9.

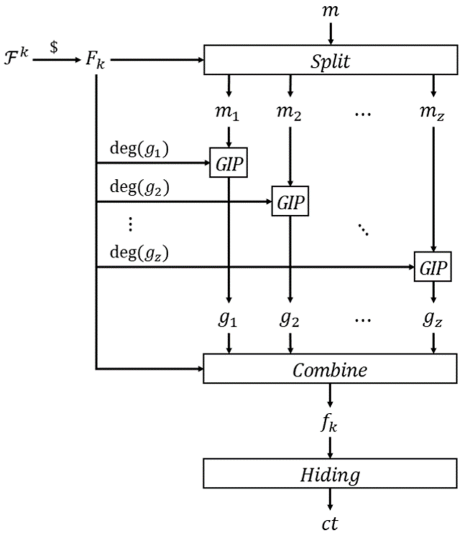


Fig. 9. Aim for Encryption Flow

Divide the message m into $m_1, m_2, \cdots$ according to the combinatorial scheme F selected from $\mathcal{F}_k$. Each of these is used recursively to generate k'-degree subpolynomials $g_1, g_2, \cdots$. Then, combine $g_1, g_2, \cdots$ into a k-degree polynomial f to form the ciphertext.

When constructing a high-degree ciphertext polynomial from low-degree polynomials, the computation required by the sender to generate the ciphertext is $O(4^k)$. This is extremely efficient compared to the computation required by an attacker to recover the plaintext, as discussed in Section IV.

In this scenario, the attacker could attempt an attack based on low-degree polynomials. However, due to the difficulty of polynomial factorization, the attacker cannot divide the ciphertext polynomial into low-degree polynomials [24], [25]. Additionally, the reduction procedure performed in the encryption algorithm alters the composition of polynomial terms, further complicating the factorization.

The receiver can decrypt the ciphertext without knowing how the sender generated it. So, the sender doesn't need to inform anyone about the chosen F. Eventually, the attacker will not be able to factorize the ciphertext without knowing F, meaning the attack will be based on the degree of the high-degree polynomial.

Example 10. Suppose the sender generates a k-degree ciphertext polynomial f_k by choosing the following combination scheme.

$$F = f_{k_1} \times f_{k_2} + f_{k_3} \times f_{k_4} \xleftarrow{\$} \mathcal{F}_k$$

When an attacker eavesdrops the ciphertext and attempts to attack it, they know that the sender generated the k-degree ciphertext using polynomials of degree $\leq k$, so they might attempt to factorize F into $f_{k_1} \times f_{k_2} + f_{k_3} \times f_{k_4}$ and conduct an attack on each of these. However, by only intercepting the ciphertext, the attacker cannot determine whether the ciphertext was generated using any of the various F, including the following methods.

$$\begin{aligned} &f_{k_1} \times f_{k_2} \\ &f_{k_1} + f_{k_3} \times f_{k_4} \\ &f_{k_1} \times f_{k_2} + f_{k_3} \times f_{k_4} \\ &\quad\vdots \end{aligned}$$

Therefore, the attacker cannot perform an attack on each of them by factoring into low-degree polynomials, and must attacks based on the degree k of the ciphertext F_k.

IV. SECURITY ANALYSIS

The primary attack strategies against the Perfect Code Cryptosystem are exhaustive key search and plaintext recovery attack. The main security parameters directly affecting both attacks are n and k. In addition, the ciphertext search attack in the exhaustive key search is influenced by the security parameter n_e. The key recovery attack of the exhaustive key search is only affected by n, and security can be balanced by increasing k to ensure a similar level of security against the plaintext recovery attack as the one provided against this attack.

A. Main attack

Before delving the various attack techniques, we note that our cryptosystem implements the reduction procedure when the distance between vertices is 2 or less. Therefore, when executing attacks, it is more efficient to choose vertices that are at a distance of 3 or more apart. This approach ensures that the security analysis doesn't account for the reduction, rather than constructing a ciphertext over all possible vertices and subsequently inferring the result of the reduction. This strategy of vertex selection is often used in proofs of attack complexity computation, as we will describe later. Proposition 5 presents the number of combinations to select j vertices in a 3-regular graph such that all selected vertices are not reductions. This proposition corresponds to a specific case where $r_1 = 4, r_2 = 10$ in Proposition 6.3(a) of Koblitz's paper [2].

Proposition 5. *In a 3-regular graph with n vertices, the minimum number of ways to select distinct j vertices, which generate the same monomial both before and after applying the reduction, is*

$$10^j \binom{(n-4)/10}{j}.$$

1) KEY RECOVERY ATTACK

A key recovery attack is a technique that aims to find $n/4$ PDS vertices within the n vertices of a public key graph, with the goal of uncovering the receiver's secret key. The problem of determining whether an arbitrary graph has PDSes is NP-complete, and it is conjectured that the problem of finding these vertices in a graph with PDSes presents a comparable level of security [23]. Even if the attacker does not identify exactly D - the arbitrary PDS selected by the receiver during key generation - the attack is successful if at least one PDS is discovered.

The Perfect Code Cryptosystem responds to key recovery attacks by leveraging NP-hardness, where the computation of a key recovery attack is proportional to the size of the graph n.

Theorem 1. *The complexity of the key recovery attack by exhaustive key search is* $O((3e)^{n/4}) \leq O(\binom{n}{n/4}) \leq O((4e)^{n/4})$.

Proof. Since a 3-regular graph can be constructed with 4 PDSes, an attacker can recover the secret key by finding $n/4$ vertices that belong to one of the PDSes among n vertices.

$$\binom{n}{n/4} = \frac{n(n-1)\cdots(n-(n/4)+1)}{(n/4)!}.$$

We can apply the stalling approximation to get an upper and lower bound for the following expression,

$$\frac{(3n/4)^{n/4}}{(\pi n/2)^{1/2}(n/4e)^{n/4}} \leq \binom{n}{n/4} \leq \frac{n^{n/4}}{(\pi n/2)^{1/2}(n/4e)^{n/4}}.$$

This inequality can then be simplified as follows.

$$(3e)^{n/4} \leq \binom{n}{n/4} \leq (4e)^{n/4}.$$

□

The security of our algorithm against key recovery attacks increases exponentially with n, the size of the graph.

2) CIPHERTEXT SEARCH ATTACK

Another exhaustive key search strategy is the ciphertext search attack, which aims to identify a message that corresponds to a given ciphertext.

Over the given graph, every invariant polynomial has a unique value. Therefore, the set $\mathfrak{F}$ of all possible ciphertext polynomials that an attacker can generate by following the cryptosystem's algorithm can be considered as the disjoint union of the set $\mathfrak{F}(m)$ of ciphertext polynomials for every message m.

$$\mathfrak{F} = \bigsqcup_{m \in \mathbb{Z}_p} \mathfrak{F}(m)$$

By eavesdropping the sender's ciphertext, an attacker can locate $\mathfrak{F}(m)$ within $\mathfrak{F}$ that has the same polynomials with the ciphertext and thus determine the message that corresponds to the ciphertext. If the number of available invariant polynomials is small, the number of searches required by the attacker to find a polynomial that matches the ciphertext is significantly reduced. Therefore, the sender must deal with a sufficiently large number of invariant polynomials to ensure that the attacker's operations cannot identify the set to which the ciphertext belongs.

Property 4. The computational cost of the attacker's ciphertext search attack, depends on the number t_k of all k-degree ciphertext polynomials that the sender can generate.

This attack is highly dependent on the security parameters n and k, as well as n_e.

Lemma 1. *The quantity t_k, representing the count of ciphertext polynomials of degree k or less that the sender can generate, satisfies the following expression for k.*

$$\begin{aligned} t_1 &= p^{n_e}\binom{n}{n_e} \\ t_k &\geq \frac{\{p^{n_e}\binom{n}{n_e}\}^{4^{k-1}}}{[p^4\{n-4-10(n_e+k-3)\}]^{1/3}} \quad (k>1) \end{aligned}$$

Proof. The form of polynomial t_k generated by the sender is influenced by the number of vertices and the range of coefficients that can be selected.

When $k = 1$, the polynomial f_1 is constructed by selecting n_e different vertices, multiplying the invariant polynomial e_i from Example 5 generated by each vertex with an arbitrary coefficient, and subsequently adding them together. Thus,

$$t_1 = p^{n_e}\binom{n}{n_e}.$$

For $k > 1$, the polynomial f_k is generated by of selecting one vertex from $n-4-10(n_e+k-3)$ previously unselected vertices that is irreducible, multiplying f_{k-1} by some coefficient for each of the four vertex variables, and then adding them together. Then,

$$t_k \geq (t_{k-1}p)^4\{n-4-10(n_e+k-3)\}.$$

Let, $r(k) = p^4\{n-4-10(n_e+k-3)\}, r(1) = 1$.
The generalization of t_k for k is

$$\begin{aligned} t_k &= t_{k-1}^4 r(k) \\ &= t_1^{4^{k-1}}/r(k)^{\frac{1}{3}}. \end{aligned}$$

and Therefore

$$t_k \geq \frac{\{p^{n_e}\binom{n}{n_e}\}^{4^{k-1}}}{[p^4\{n-4-10(n_e+k-3)\}]^{1/3}}.$$

□

Theorem 2. *In a environment where the attacker has already prepared all t_k polynomials of degree k, the computational cost of a ciphertext search attack is* $O(t_k) \geq O(\{(n-n_e)/n_e\}^{n_e 4^{k-1}})$.

Proof. The computational cost of a ciphertext search attack is reliant on the number of ciphertext terms, denoted as t_k.

$$t_1 = p^{n_e}\binom{n}{n_e}.$$

And here,

$$p^{n_e}\binom{n}{n_e} = p^{n_e}\frac{n(n-1)\cdots(n-n_e+1)}{n_e!}$$

We can utilize the Stirling approximation to get a lower bound for this expression,

$$p^{n_e}\binom{n}{n_e} \geq \frac{\{p(n-n_e)\}^{n_e}}{(2n_e\pi)^{1/2}(n_e/e)^{n_e}}.$$

A simplified representation of this inequality is as follows.

$$p^{n_e}\binom{n}{n_e} \geq \left(\frac{pe(n-n_e)}{n_e}\right)^{n_e}.$$

This yields the following expression for t_k.

$$\begin{aligned} t_k &\geq \frac{\{p^{n_e}\binom{n}{n_e}\}^{4^{k-1}}}{[p^4\{n-4-10(n_e+k-3)\}]^{1/3}} \quad (k>1) \\ &\approx \frac{1}{p^4}\left(\frac{pe(n-n_e)}{n_e}\right)^{n_e 4^{k-1}} \end{aligned}$$

In this approximation, p is a fixed constant parameter. □

According to this theorem, given the parameters $n = 256, n_e = 3$ proposed by our algorithm, the security against ciphertext search attacks increases exponentially with the degree k of the ciphertext, thereby achieving a security of 4.8×10^5-bits for the proposed parameter $k = 7$.

3) PLAINTEXT RECOVERY ATTACK

A plaintext recovery attack is a technique that compares the coefficients of a arbitrary invariant polynomial with the coefficients of a ciphertext to find an equivalent polynomial that evaluates to the same value. This method does not directly identify the key (i.e., find some PDS) or ciphertext polynomial, but derives the message information dispersed among the coefficients useing Gaussian cancellation.

In a plaintext recovery attack, the attacker generates an arbitrary polynomial to recover the message, then compares the coefficients of the terms in the intercepted ciphertext with the coefficients of the terms in the arbitrary polynomial they generated. They find a set of coefficients that are equivalent (i.e., have the same value) to the random coefficients set chosen by the sender. To identify this set of coefficients, the attacker generates a linear system for the their polynomial and the ciphertext and represents it as a matrix to apply Gauss-Jordan elimination. The last column of the matrix represents the values of the coefficients of the ciphertext, and after applying the elimination, the last column of the RREF matrix is summed at $\mathbb{Z}_p$ to retrieve the message.

The algorithm for this attack procedure is described in APPENDIX A and an example of its execution can be found in APPENDIX B.

The computational complexity of this attack depends on the size of the matrix to which it is applied, as it follows the computational complexity of Gaussian elimination. The computational complexity of Gaussian-Jordan elimination on the $a \times b$ matrix is $O(a^2b)$.

Property 5. Let $O\left(a^2b\right)$ denote the attacker's computational cost for a plaintext recovery attack, assuming that the attacker can prepare the necessary arbitrary polynomials in advance. Here, a is the minimal number of monomials appearing in an arbitrary ciphertext and b is the minimal number of unknowns appearing in an arbitrary ciphertext.

Both a and b are proportional to n and k, and have values independent of n_e.

Lemma 2. *When generating an arbitrary polynomial for an attack, the polynomials generated when $n_e = 1$ and when $n_e > 1$ are equivalent. Moreover, the polynomials generated by the attacker, either considering or ignoring the terms added by neighboring vertices in the process of increasing the polynomial's degree, are equivalent.*

Proof. To deal with all possible monomials generated by the sender, the attacker collects all possible combinations of vertices without repetition, encrypts each combination, and adds them to generate an arbitrary polynomial.

Assume that the public key graph is given by Fig. 4. The vertex pairs that can be selected to generate f_1 are

$$\{\{v_1\}, \{v_2\}, \cdots, \{v_8\}, \{v_1, v_2\}, \cdots, \{v_1, v_2, \cdots, v_8\}\}.$$

When $n_e = 1$, g_1 is the invariant polynomial of degree 1, as shown below, with $S_1 = \{\{v_1\}, \{v_2\}, \cdots, \{v_8\}\}$ covers all cases where the size of the vertex pair set is 1.

$$g_1 = \textstyle\sum_{i \in S_1} c_i \sum_{j \in N[i]} x_j$$

When $n_e = 2$, g_2 is the invariant polynomial of degree 1, as shown below, with $S_2 = \{\{v_1, v_2\}, \{v_1, v_3\}, \cdots, \{v_7, v_8\}\}$ covers all cases where the size of the pair set equals 2.

$$g_1 = \textstyle\sum_{i \in S_2} \sum_{j \in i} c_j \sum_{l \in N[j]} x_l =$$

The coefficients of $\sum_{v \in N[v_1]} x_v$ and $\sum_{v \in N[v_1]} x_v$, generated by g_1 and g_2 respectively, are simply sums of unknowns. We cannot distinguish between g_1 and g_2 when we substitute c_1' and c_2' respectively.

Similarly, when $n_e \geq 3$, g_{n_e} is indistinguishable from g_1, This implies that the polynomials generated for the plaintext recovery attack are equivalent, regardless of whether $n_e = 1$ or $n_e > 1$.

On the other hand, consider the process of increasing the degree of a polynomial over the graph represented in Fig. 4. Let $g_i, i \in \{1, 2, \cdots, 7\}$ be an invariant polynomial of degree $k-1$ and $c_i, i \in \{1, 2, \cdots, 7\}$ be a constant. Then, let the k-degree polynomial generated by selecting vertex v_1 is $\widetilde{f_k}$ and the k-degree polynomial generated by selecting vertex v_2 is $\widetilde{f_k'}$.

$$\widetilde{f_k} = (g_1+c_1)x_{v_1} + (g_2+c_2)x_{v_2} + (g_3+c_3)x_{v_4} + (g_4+c_4)x_{v_6}$$
$$\widetilde{f_k'} = (g_1+c_1)x_{v_1} + (g_5+c_5)x_{v_2} + (g_6+c_6)x_{v_3} + (g_7+c_7)x_{v_7}$$

The polynomial generated for the attack takes the form $\widehat{f_k} = \widetilde{f_k} + \widetilde{f_k'} + \cdots$. We cannot distinguish between the term $(g_1 + c_1)x_{v_1}$ generated by $\widetilde{f_k}$ and the term $(g_1 + c_1)x_{v_1}$ generated by $\widetilde{f_k}$.

Moreover, the vertex selected to generate the k-degree polynomial and the polynomial of degree $k-1$ generated by its neighboring vertices are determined independently, and hence, they do not share coefficient information. Consequently, the coefficients, which are the sum of the unknowns from terms by each neighboring vertex, can be expressed by substituting distinct other unknowns.

If $g_i', i \in \{1, 2\}$ is an invariant polynomial of degree $k-1$ and $c_i', i \in \{1, 2\}$ is a constant, the expression can be represented as

$$\begin{aligned}\widehat{f_k} &= (2g_1 + 2c_1)x_{v_1} + (g_2 + c_2 + g_5 + c_5)x_{v_2} \\ &\quad + (g_6 + c_6)x_{v_3} + (g_3 + c_3)x_{v_4} + \cdots \\ &= (g_1' + c_1')x_{v_1} + (g_2' + c_2')x_{v_2} \\ &\quad + (g_6 + c_6)x_{v_3} + (g_3 + c_3)x_{v_4} + \cdots.\end{aligned}$$

In conclusion, this is equivalent to not taking into account the terms from neighboring vertices during the process of increasing the degree of the polynomial. □

Lemma 3. *The minimal number of monomials that appear in an arbitrary ciphertext polynomial generated by an attacker is $\sum_{i=1}^{k} \frac{10^{i-1}n}{i}\binom{\frac{n-4}{10}}{i-1}$ and the minimal number of unknowns is $n^k + \frac{n(n^{k-1}-1)}{n-1}$.*

Proof. The set of monomials that should be included in any ciphertext generated by the attacker must enclude all monomials that could appear in the intercepted sender's ciphertext.

The sender's encryption scheme has the ability to represent all terms that could appear after the reduction over the graph G. According to the Proposition 5, the number of combinations for selecting j vertices from the vertex set V to generate j polynomials is

$$\geq 10^j \binom{(n-4)/10}{j}$$

where $\binom{x}{j}$ is interpreted to be equal to zero if $x \leq j - 1$.

As the arbitrary polynomials we generate for our attack cover the cases of $i = 1, 2, \cdots, k$, the minimum number of monomials that need to be present in an arbitrary ciphertext polynomial is $\sum_{i=1}^{k} \frac{10^{\,i-1}n}{i}\binom{\frac{n-4}{10}}{i-1}$.

On the other hand, the number of unknowns required to generate a monomial according to the cryptographic algorithm is determined by the vertices chosen to generate the monomial and the random variables that map to them.

Given that the unknowns represent each vertex chosen to generate f_1 and a constant term that is multiplied when increasing the degree of the polynomial, the number of unknowns π_k used in an arbitrary polynomial of degree f_k can be given by a specific expression.

$$\pi_1 = n_e\binom{n}{n_e}$$
$$\pi_k = (\pi_{k-1}+1)\times|N[v]|\times|V|$$

By Lemma 2, the attacker does not need to consider n_e and $|N[v]|$. Therefore, a generalization of π_k is

$$\begin{aligned}\pi_k &= (\pi_{k-1}+1)\,n\\ &= n^k + \frac{n(n^{k-1}-1)}{n-1}.\end{aligned}$$

□

Theorem 3. *In an environment where the attacker can prepare the polynomials of degree k needed for the attack in advance, the computational cost of the plaintext recovery attack is $O\left(n^{3k}/k^{2k}\right)$.*

Proof. The computational complexity of the plaintext recovery attack depends on the size of the matrix applied to Gaussian elimination. The size of the row space of the matrix required for the attack is $\sum_{i=1}^{k}\frac{10^{\,i-1}n}{i}\binom{\frac{n-4}{10}}{i-1}$, and the size of the column space is $n^k+\frac{n(n^{k-1}-1)}{n-1}$.

Let the size of the row space be a and the size of the column space be b. Then,

$$\begin{aligned}a^2\times b &= \left\{\sum_{i=1}^{k}\frac{10^{\,i-1}n}{i}\binom{\frac{n-4}{10}}{i-1}\right\}^2\left(n^k+\frac{n(n^{k-1}-1)}{n-1}\right)\\ &\geq \left\{\sum_{i=1}^{k}10^i\binom{n/10}{i}\right\}^2(2n^k+n^{k-1}+\cdots+n).\end{aligned}$$

Using the Stalling approximation, the lower bound is

$$a^2\times b \geq \left\{\sum_{i=1}^{k}\frac{(n-10i)^i}{(2i\pi)^{1/2}(i/e)^i}\right\}^2(2n^k+n^{k-1}+\cdots+n)$$

As a result, we can simplify this inequality as follows:

$$\begin{aligned}a^2\times b &\geq \left\{\sum_{i=1}^{k}\left\{\frac{e(n-10i)}{i}\right\}^i\right\}^2(2n^k)\\ &\geq \left\{\frac{e(n-10k)}{k}\right\}^{2k}(2n^k)\\ &\geq \left\{\frac{n(n-10k)^2}{k^2}\right\}^k\\ &\geq \left\{\frac{n^3}{k^2}-\frac{20n}{k}+100\right\}^k.\end{aligned}$$

□

According to Theorem 3, given that the parameters of the IPCC7 are such that $n \gg k$, the security of our algorithm against plaintext recovery attacks increases exponentially with the ciphertext degree k.

Remark. The computational cost that an attacker needs to perform for each attack can be summarized as follows.

- Key recovery attack : $O(\binom{n}{n/4}) \geq O((3e)^{n/4})$
- Ciphertext search attack : $O\left(t_k\right) \geq O\left(\binom{(n-n_e)}{n_e}^{n_e4^{k-1}}\right)$
- Plaintext recovery attack : $O\left(n^{3k}/k^{2k}\right)$

Note that this computational cost for each attack grows exponentially with the security parameters n, k, n_e.

Moreover, for the plaintext recovery attack, which assumes an arbitrary ciphertext polynomial with unknown coefficients, the computation required by the attacker to generate a single polynomial differs from that required by the sender to generate the ciphertext by at most $O(n_e^k)$. Therefore, security can be adjusted by increasing or decreasing the value of the relevant security parameter.

B. *Rationale for parameter selection*

The security corresponding to the value of each parameter for each attack is illustrated in the following.

TABLE II

COMPLEXITY OF KEY RECOVERY ATTACK FOR n (BIT)

$n = \|V\|$	80	120	160	200	240
complexity	61	93	126	158	191

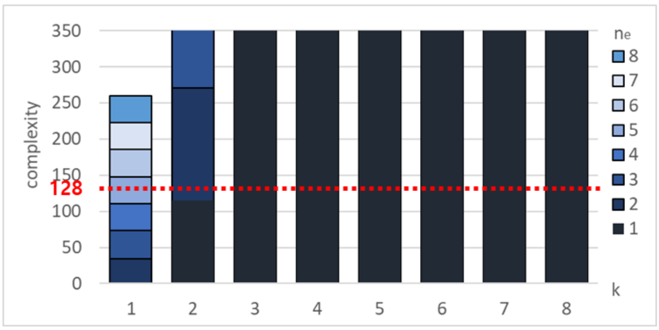


Fig. 10. Complexity of ciphertext search attack for k and n_e (unit:bit, $n = 256$)

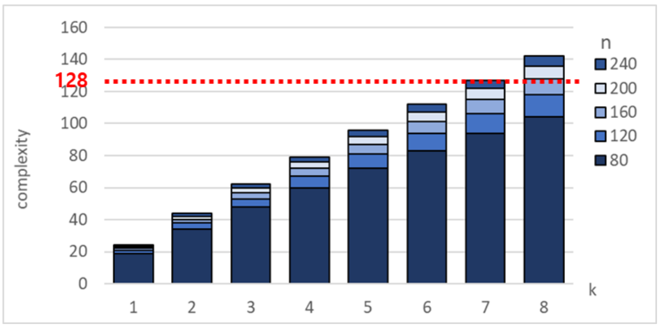


Fig. 11. Complexity of plaintext recovery attack for n and k (unit: bit)

TABLE III

COMPLEXITY OF PLAINTEXT RECOVERY ATTACK FOR k (UNIT: BIT, $n = 256$)

k	2	3	4	5	6	7	8
complexity	45	63	81	97	113	129	145

According to TABLE II, to ensure 128-bit security, the size of the graph should be $n \geq 164$. When $n = 164$, it provides 130-bit security against key exhaustive search attacks. The parameter $n = 256$, proposed in TABLE I, satisfies the 128-bit security requirement.

Before selecting the parameter n, we considered the data type to represent the graph vertices from an implementation perspective. This is related to the memory size required for intermediate operations during the algorithm execution and for the ciphertext. We found that increasing the size of the graph within the same data type does not significantly affect the algorithm speed or memory usage. Therefore, we chose the graph size of $n = 256$, which can represent more than 164 vertices, to satisfy the 128-bit security level using the data type *char* (1 byte). This allows us to generate a 3-regular graph. The number 256 is a multiple of 4, so we can generate a 3-regular graph with that number of vertices, representing 256 vertices (from 0 to 255) in one byte.

According to Fig. 10, to ensure 128-bit security against the ciphertext recovery attack when $n_e = 1$, the maximum degree of the polynomial should be $k \geq 2$. The parameters k and k', proposed in TABLE I, satisfy the 128-bit security level. Since the 128-bit security level is satisfied starting from $n_e = 1$, if there is a desire to use higher values to increase security, these can be adjusted considering memory and speed. We determined the proposed parameters by taking into account the implementation speed and ciphertext size.

The plaintext recovery attack generates an arbitrary polynomial based on the maximum degree of the eavesdroped ciphertext. Therefore, we can counter this attack by increasing k. According to TABLE III, when the graph size is 256, it requires $k \geq 7$ to ensure 128-bit security, and our proposed parameter $k = 7$ satisfies 144-bit security.

C. Other attacks for specific case

In addition to the main attack techniques, there are other possible attacks.

For ciphertexts of a certain form, it is possible to perform an attack on the message by computing the sum of the coefficients. During the process of generating ciphertext, if U is a subset of V, then an invariant polynomial of the following form is vulnerable to a straightforward sum over coefficients attack.

$$ct = \sum_{U_i \subset U} c_i \prod_{u \in U_i} \sum_{v \in N[u]} x_v$$

This issue arises when the recursive process used to generate a k-degree invariant polynomial only takes an invariant polynomial g of degree $k-1$ into consideration, causing the invariant polynomial to take the form $(g + a)\,\Sigma_{u \in N[v]} x_u$. Naturally, this polynomial can be used as a ciphertext, but the problem is that the message information is naively exposed in the coefficients.

In this case, the same coefficient c_i appears approximately 4^k times for degree k, and there are no meaningless coefficients selected to confuse the attacker. For a ciphertext with these properties, it is possible to recover the message in a very simple way by merely adding the different coefficients that appear one by one.

To avoid this simplistic coefficient summation attack, f_{k-1} should be chosen more diversely when generating the ciphertext.

Additionally, the frequency of a vertex variable's appearance can be used to analyze which vertices are selected when the degree is increased. This characteristic is inherent to the recursive method: vertex variables selected to generate high-degree polynomials appear more often than those selected to generate low-degree polynomials.

To counteract this issue, we can use a method that generates multiple k-degree polynomials and sums them together. This method can solve the problem at the cost of a constant factor increase in the computational effort compared to the computational effort of the existing encryption method for the sender. Our algorithm incorporates this technique in the process of selecting $\mathcal{F}$.

V. Implementation

The encryption speed of a cryptosystem is heavily influenced by k. We propose a method that constructs high-degree polynomials from low-degree polynomials, which allows for rapid encryption while enhancing security against known attack strategies.

In this chapter, we give the performance of the implemented program that utilizes the proposed parameters and algorithms. The pseudocode of the cryptosystem is provided in APPENDIX C, and an example of program execution is described in APPENDIX D.

Moreover, it has not been mentioned in this paper that the Perfect Code Cryptosystem relies more heavily on random numbers from a random number generator than other cryptographic algorithms do. However, our cryptosystem operates under the assumption that the random numbers used for vertex selection and coefficients are obtained from a cryptographically secure random number generator. For the implementation of our algorithm, we employed AES-CTR-DRBG.

A. Implementation Considerations

1) POLYNOMIAL STRUCTURE

In speed-sensitive cryptosystems, it is necessary to devise a design for static memory allocation rather than dynamic allocation. However, static memory allocation can lead to wasted memory due to various reasons. Fortunately, when designing the implementation of the invariant polynomial in the IPCC7, unlike general polynomials, it is possible to measure the suitable range of the upper and lower limits based on the degree of polynomial. Therefore, we use static memory allocation in consideration of the upper limit.

And in the design of a polynomial, a coefficient and vertices together form a term. This uses 4 bytes for the coefficient and 1 byte for each vertex. The coefficient is designed as an unsigned int data type to calculate coefficients without big number operations, while the vertex is represented as a byte data type to represent vertices in a cryptosystem with 128-bit security. By using different data types for each kind of data, we were able to minimize the memory required for the entire encryption process to approximately a quarter of what it would be if we only used the coefficient's data type.

2) LOOKUP TABLE

Encryption speed is significantly affected by the hiding procedure, which accounts for 90% of the total encryption time. However, hiding is essential as it increases the complexity of factoring the ciphertext. To enhance the speed of this operation, we utilize a precomputed lookup table. The public key, the output of the key generation algorithm, stores the information of the edges. By reorganizing it into a 256×4 table of closed neighborhoods for each vertex, we managed to improve the operation of checking whether the distance between two vertices of the same term is 1 or 2 in the RD of the hiding process - this reduced the complexity from $O(n)$ to $O(1)$. Such enhancement leads to a considerable improvement in encryption performance, especially as the number of graph vertices increases and the degree of the invariant polynomial rises.

Decryption is the process of obtaining a message using the PDS. The secret key generated by the key generation algorithm is a byte array containing $256/4$ PDS vertices. For decryption, this array is reconstructed into a 256×1 table. This is achieved by storing a value of 1 for the vertex belonging to the PDS and 0 for the vertex not belonging to the PDS in a vertex indexing box. Using this, the vertices that make up the term of the ciphertext can be indexed to obtain a value, and an AND operation can be performed to determine whether all vertices in the term belong to the PDS. In other words, if the result of the AND operation is 1, then all vertices in the term belong to the PDS. Using this method, the speed of the search operation is reduced from $O(n)$ to $O(1)$ compared to the existing PDS storage method. In actual measurements, this has led to a performance improvement of about 150 times or more.

3) *FUTURE WORK*

Currently, in our cryptosystem, the encryption algorithm spends most of the encryption time in the SC process during the hiding phase, which has a time complexity of $O(n^2)$. However, by defining terms based on lexical ordering according to vertices, it is possible to achieve a speed improvement of about 5 times. In other ways, it is predicted that by appropriately applying hashing to the lexical ordering - similar to the dictionary of Python or hashmap of Java - the time complexity will decrease from $O(n^2)$ to $O(n)$.

B. *Performance analysis*

The Perfect Code Cryptosystem, known for outputting very large ciphertexts, assumes an environment with an abundant memory capacity. Thus, when analyzing time and memory complexity, we first examine the time complexity before considering the memory complexity.

All benchmarks were obtained using an Intel(R) Core(TM) i5-7360U CPU @2.3GHz processor. The benchmarking PC is equipped with 8GB RAM and compiled using Apple clang version 13.0.0.

All measured results are the averages of 1,000 iterations.

As we observed in the previous analysis, the security strength of the Perfect Code Cryptosystem is determined by two main security parameters: n, the number of graph vertices, and k, the maximum degree of the ciphertext polynomial. For a cryptosystem to be adopted in the real world, a balance must be struck between these parameters in terms of security and performance. From an implementation perspective, the speed of key generation is influenced by n. For a graph with $n = 256$, the key generation speed is 2.32*ms*.

The program we implemented to measure speed does not include the part where we verify whether the generated graph is connected, as seen in line 3 of Algorithm 1. Checking whether a graph is connected requires slightly more than $O(n)$ computation, which is not insignificant when considering the total computation required for the key generation algorithm. Furthermore, if the size of the graph is increased for security reasons, this step could make the key generation process extremely time-consuming. However, we expect the probability of generating a disconnected graph is low, and thus, we excluded this check from our speed measurements.

The encryption speed is impacted by both n and k, with the effect of k being particularly significant. While the decryption speed is also influenced by n and k, its dependency on these variables is much less pronounced than that of the encryption speed.

When n is fixed at 256, A comparison of the operation speed of the algorithm, between the existing method and our approach, is presented in Fig. 12.

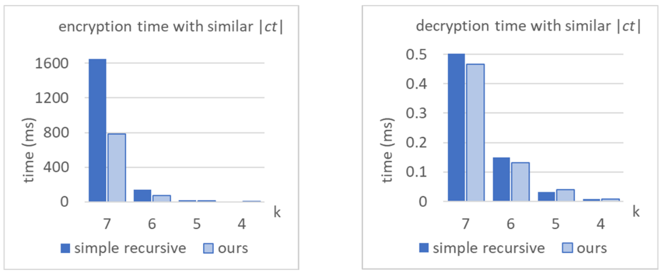


Fig. 12. Algorithm performance about each case for k

The marginally slower decryption speed when running the IPCC7 at $k = 4, 5$ is due to the slightly greater number of terms in the ciphertext generated by our algorithm. We adjusted n_e so that the average number of generated ciphertext terms is comparable, but it is not possible to fine-tune the parameters such that both methods produce exactly the same number of terms. However, the difference in decryption speeds between the two methods is insignificant. Furthermore, even when $k = 6, 7$, the IPCC7 generates a slightly higher number of terms in the ciphertext, yet the difference in decryption speeds between the two methods remains insignificant.

For the encryption speed and ciphertext size, the performance differences between using a simple recursive method and our algorithm are as follows:

TABLE IV
PERFORMANCE OF ENCRYPTION ALGORITHM FOR k WHEN $n = 256$

k	speed (ms)		$\|ct\|$	
	simple recursive	**our algorithm**	**simple recursive**	**our algorithm**
4	3.40	2.28	380	325
5	18.33	16.68	1322	1474
6	141.13	184.83	4438	6263
7	1498.83	2335.44	14402	21438

Compared to Koblitz's recursive scheme, our simple recursive algorithm produces ciphertexts with fewer terms but achieves a faster encryption speed. And, when comparing our algorithm IPCC7 with the simple recursive method, you can observe that the IPCC7 generates ciphertexts with more terms and achieves an even faster encryption speed. This implies that our algorithm is significantly faster compared to Koblitz's algorithm, while still maintaining the number of terms that is meaningful from a security analysis perspective.

For all the provided parameters, after averaging the results of 1,000 runs of each scheme, the operational time is 2.32ms for key generation, 2335.44ms for encryption, and 0.947ms for decryption. The size of the public key is $(3n/2)2 = 768$ bytes, the size of the secret key is $n/4 = 64$ bytes, and the average size of the ciphertext is 235,818 bytes.

TABLE V
IMPLEMENTATION SPEED WITH PROPOSED PARAMETERS

Key Generation	Encryption	Decryption
3.33 Mbps	0.81 Mbps	1,886.54 Mbps

C. Comparing to other PQCs

In our implementation, the Perfect Code Cryptosystem produces a ciphertext of 235,818 bytes for a 768-byte public key with 128-bit security. The data size transmitted over the network is 236,586 bytes, and the algorithm execution time of 2,338.71ms (this includes key generation, encryption, and decryption).

Fig. 13 and Fig. 14 compare the performance of our proposed algorithm with several other PQC algorithms, all running with parameters that ensure 128-bit security [11].

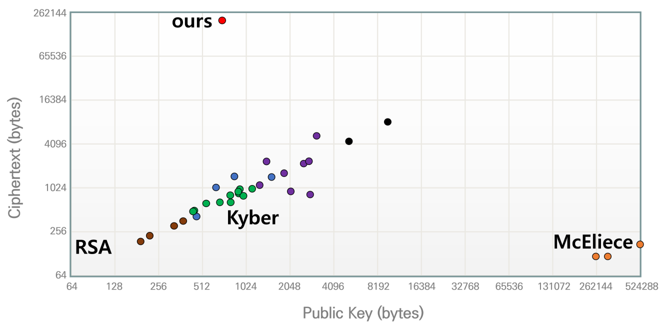


Fig. 13. Public Key and Ciphertext size

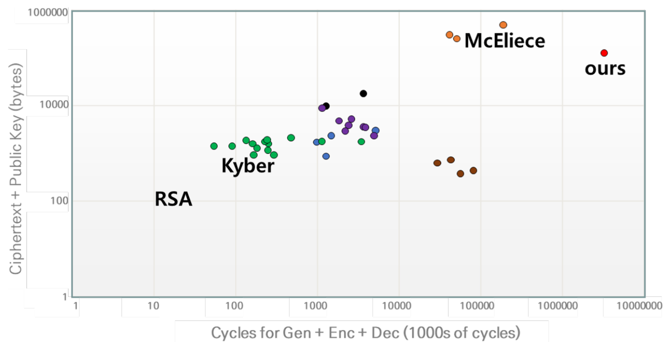


Fig. 14. Operation speed and Data size

The Perfect Code Cryptosystem has the advantage of very fast key generation and decryption speeds, and its encryption speed, though relatively slower, is still realistically usable. Unique features of this cryptosystem include its reliance on a novel problem that is distinct from the ones proposed during the NIST PQC algorithm standardization process: small public key and large ciphertext sizes. Given these characteristics, it is expected to outperform other PQC algorithms in applications such as one-way functions and white-box encryption, especially in environments where memory limitations are not a major concern.

VI. Conclusion

In the research field of public-key cryptosystems, the Perfect Code Cryptosystem has long been overlooked. The extreme inefficiency of this cryptosystem, in terms of speed and memory efficiency, has kept it out of the spotlight. However, by integrating the concept of polynomial combinatorics, we have achieved a dramatic increase in encryption speed.

In this paper, we conducted a study to determine the appropriate parameters related to the security of the cryptosystems, including the size of the graph, the maximum degree of the ciphertext, the number of terms in the ciphertext polynomial, and the number of polynomial types that can be generated. We then demonstrated that the algorithm could be realistically implemented and used with these parameters.

Cryptosystem design based on graph theory and combinatorics is not yet a fully matured technique. The proposed algorithm IPCC7 has several aspects that are not yet implemented and require further security assessment, considering the completeness of the cryptosystem. Nonetheless, we have presented and implemented a realistic algorithm for a Perfect Code Cryptosystem and identified directions for future research.

Numerous ideas could be studied and integrated for the effective usage of cryptosystems. For instance, it is possible to generate subpolynomials using multiple different graphs sharing a PDS, or to extend PDF(Perfect Dominating Function) to PMDF(Perfect Minus Dominating Function) to enhance security against key recovery attacks.

In our future work, we will analyze more general security, improve the algorithm, and optimize the implementation. We anticipate that these studies will bolster the field of combinatorics-based cryptosystems and contribute to diversifying the problems underlying post-quantum cryptography.

Appendix

A. algorithm of the plaintext recovery attack

Algorithm 7 Plaintext Recovery Attack

```
Input: ciphertext $ct = f_k$
Output: plaintext $pt$
1: To generate an arbitrary polynomial $\widehat{f_k}$ that encompasses all terms that can appear in the ciphertext the sender might generate, collect the vertices required to generate generating the ciphertext along with their corresponding pairs of unknowns.
2: Use these organized pairs to generate an arbitrary polynomial $\widehat{f_k}$. Note that $\widehat{f_k}$ is an invariant polynomial, generated in a simple recursive method without utilizing $\mathcal{F}_k$.
3: Then, set $f_k = \widehat{f_k}$ to obtain a linear system for the sum of the unknowns, which are the coefficients of the monomial of $\widehat{f_k}$.
4: Transform the resultant linear system and apply Gauss-Jordan elimination to the matrix.
5: Add the values in the last column of the RREF matrix over $\mathbb{Z}_p$ to derive the plaintext $pt$.
6: return $pt$
```

B. Example of Executing the Plaintext Recovery Attack

This example illustrates how Gauss-Jordan elimination can be utilized in the plaintext recovery attack. Suppose an attacker, Eve, wants to recover a message by intercepting Alice's ciphertext. The security parameter

$$p = 11, n = 8, k = 1, k' = 1, n_e = 2$$

shared by the principals of the cryptographic communication, Alice (the sender) and Bob (the receiver), is known, and Bob's public key graph is illustrated in Fig. 4.

Eve is also aware of this public security parameter and the public key. The eavesdropped Alice's ciphertext is displayed below.

$$ct = 4x_{v_1} + 9x_{v_3} + 2x_{v_5} + 4x_{v_6} + 2x_{v_7} + 9x_{v_8}$$

To carry out Gauss-Jordan elimination by generating a linear equation, Eve constructs an arbitrary polynomial that contains all possible combinations that Alice could have chosen to generate an invariant polynomial of degree 1. Given that $k = 1$, there are ${}_8C_1 = 8$ sets of vertices that can be chosen to generate a polynomial, and the (vertex, coefficient) pairs that map a random coefficient $\widehat{c}_i$ to each of these sets are as follows.

$$(v_1, \widehat{c}_1), (v_2, \widehat{c}_2), (v_3, \widehat{c}_3), (v_4, \widehat{c}_4), \\ (v_5, \widehat{c}_5), (v_6, \widehat{c}_6), (v_7, \widehat{c}_7), (v_8, \widehat{c}_8)$$

Using these pairs to generate a ciphertext results in the following arbitrary invariant polynomial of degree 1.

$$\begin{aligned} \widehat{ct} = {} & \widehat{c}_1(x_{v_1} + x_{v_2} + x_{v_4} + x_{v_6}) \\ & + \widehat{c}_2(x_{v_1} + x_{v_2} + x_{v_4} + x_{v_6}) \\ & + \cdots \\ & + \widehat{c}_7(x_{v_2} + x_{v_6} + x_{v_7} + x_{v_8}) \\ & + \widehat{c}_7(x_{v_3} + x_{v_5} + x_{v_7} + x_{v_8}) \end{aligned}$$

At present, this polynomial is arranged for the unknown coefficients $\widehat{c}_i$, but it can be rearranged for the vertex variables x_v. The solved polynomial is then compared to the stolen ciphertext to generate the following system of equations.

$$\begin{cases} \widehat{c}_1 + \widehat{c}_2 + \widehat{c}_4 + \widehat{c}_6 = 4 & \text{(coefficient of } x_{v_1}) \\ \widehat{c}_1 + \widehat{c}_2 + \widehat{c}_3 + \widehat{c}_7 = 0 & \text{(coefficient of } x_{v_2}) \\ \widehat{c}_2 + \widehat{c}_3 + \widehat{c}_4 + \widehat{c}_8 = 9 & \text{(coefficient of } x_{v_3}) \\ \widehat{c}_1 + \widehat{c}_3 + \widehat{c}_4 + \widehat{c}_5 = 0 & \text{(coefficient of } x_{v_4}) \\ \widehat{c}_4 + \widehat{c}_5 + \widehat{c}_6 + \widehat{c}_8 = 2 & \text{(coefficient of } x_{v_5}) \\ \widehat{c}_1 + \widehat{c}_5 + \widehat{c}_6 + \widehat{c}_7 = 4 & \text{(coefficient of } x_{v_6}) \\ \widehat{c}_2 + \widehat{c}_6 + \widehat{c}_7 + \widehat{c}_8 = 2 & \text{(coefficient of } x_{v_7}) \\ \widehat{c}_3 + \widehat{c}_5 + \widehat{c}_7 + \widehat{c}_8 = 9 & \text{(coefficient of } x_{v_8}) \end{cases}$$

This system of equations can be represented as a matrix, as shown in Fig. 15. Applying Gauss-Jordan elimination to the matrix in Fig. 15 yields a Reduced Row Echelon Form (RREF) matrix, as illustrated in Fig. 16. The message can be recovered from this by summing all the values in the last column over $\mathbb{Z}_p$.

The values in the last column of Fig. 16 are {2, 2, 7, 0, 2, 0, 0, 0}, and their sum gives $13 = 2 \pmod{p}$. This is consistent with $4+9 = 2 \pmod{p}$, which is obtained by substituting the value of the secret key PDF corresponding to PDS $\{v_1, v_8\}$ in the public key graph Fig. 4 into the intercepted ciphertext ct.

$$\begin{array}{cccccccc|c} \hat{c}_1 & \hat{c}_2 & \hat{c}_3 & \hat{c}_4 & \hat{c}_5 & \hat{c}_6 & \hat{c}_7 & \hat{c}_8 & \\ 1 & 1 & & 1 & & 1 & & & 4 \\ 1 & 1 & 1 & & & & 1 & & 0 \\ & 1 & 1 & 1 & & & & 1 & 9 \\ 1 & & 1 & 1 & 1 & & & & 0 \\ & & & 1 & 1 & 1 & & 1 & 2 \\ 1 & & & & 1 & 1 & 1 & & 4 \\ & 1 & & & & 1 & 1 & 1 & 2 \\ & & 1 & & 1 & & 1 & 1 & 9 \end{array}$$

Fig. 15. The $ct = \widehat{ct}$ matrix generated to perform a plaintext recovery attack

$$\begin{array}{cccccccc|c} \hat{c}_1 & \hat{c}_2 & \hat{c}_3 & \hat{c}_4 & \hat{c}_5 & \hat{c}_6 & \hat{c}_7 & \hat{c}_8 & \\ 1 & & & & & & & -1 & 2 \\ & 1 & & & & 1 & 1 & 1 & 2 \\ & & 1 & & & -1 & & & 7 \\ & & & 1 & & & -1 & & 0 \\ & & & & 1 & 1 & 1 & 1 & 2 \\ & & & & & & & & 0 \\ & & & & & & & & 0 \\ & & & & & & & & 0 \end{array}$$

Fig. 16. PPEF matrix of $ct = \widehat{ct}$ with Gauss-Jordan Elimination

C. Pseudocode for Implementation

Each vertex $v \in V$ is represented using a one-byte index number. The polynomial f_k, generated for the input message (a 32-bit plaintext), is stored in a $t \times (k+1)$ two-dimensional array, where t is the number of polynomial terms.

The cryptographic algorithms are described in the order of their execution for implementation.

1) key generation

The key generation algorithm closely follows Algorithm 1, except for the step of storing the public and secret keys.

A public key graph is comprised of a set of vertices V and a set of edges E, which can be collectively represented through the neighborhood relationships for each vertex. As the edges of a graph carry information about the vertices at both ends, one can ascertain all the vertices from E without needing to have V.

Thus, the graph public key is stored as a $\frac{3n}{2} \times 2$ two-dimensional array, and the edge information is stored in a randomized order to prevent inference of the PDS identity. The secret key is stored as a $\frac{n}{4}$ one-dimensional array, chosen from one of the PDSes from the graph, without generating a PDF.

In the key generation algorithm, the notation $A[i]$ signifies the ith element of the array A. When arrays A and B are of the same size, $A[a] \xleftrightarrow{\$} B[b]$ denotes a random one-to-one correspondence between the two elements $A[a]$ and $B[b]$.

Algorithm 8 KeyGeneration

Input: The number of vertex n

Output: A pair of keys (pk, sk)

```
1: $V \leftarrow \{1, 2, \cdots, n\}$
2: $E \leftarrow$
3: $L_1 \leftarrow$ Shuffling($V$)
4: $D_1, D_2, D_3, D_4 \leftarrow$
5: for $i = 1$ to $n/4$                          ▷ generate PDS
6:     $D_1[i] \leftarrow L_1[4i - 3]$
```

```
7:      $D_2[i] \leftarrow L_1[4i-2]$
8:      $D_3[i] \leftarrow L_1[4i-1]$
9:      $D_4[i] \leftarrow L_1[4i]$
10: for $i = 1$ to 3                    ▷ generate graph that has PDS
11:     for $j = i+1$ to 4
12:         $L_2 \leftarrow$ Shuffling($\{1, 2, \cdots, \frac{n}{4}\}$)
13:         for $k = 1$ to $n/4$
14:             $E \leftarrow E \cup \{D_i[k] \overset{\$}{\longleftrightarrow} D_j[L_2[k]]\}$
15: $sk \overset{\$}{\longleftarrow} \{D_1, D_2, D_3, D_4\}$        ▷ Select PDS as the $sk$
16: $pk \leftarrow$ Shuffling($E$)        ▷ $G$ has a vertex information potentially
17: return $sk, pk$
```

2) Encryption

The encryption algorithm is a function that calls Algorithms 11 and Algorithm 10 to generate a k-degree invariant polynomial as a ciphertext. The Distribute function is tasked with generating the message fragment and lower degree invariant polynomials that compose F, where q is the number of polynomials that constitute F. The Combine function combines the generated subpolynomials $g_1, g_2, \cdots, g_q$ into a k-degree polynomial that follows the format of F.

Algorithm 9 Encryption

```
Input: graph $G = (V, E)$, parameter set $D$, message $m$
Output: ciphertext $ct$
1: $F \overset{\$}{\longleftarrow} \mathcal{F}_k$
2: $\{m_1, m_2, \cdots, m_q\} \leftarrow$ Distribute($F, m$)
3: $\{k_1, k_2, \cdots, k_q\} \leftarrow$ Distribute($F, k$)
4: for $i = 1$ to $q$
5:     $g_i \leftarrow$ GenPolyDk($G$, GRT, , $k_i$, $m_i$)
6: $f_k \leftarrow$ Combine($g_1, g_2, \cdots, g_q$)
7: Hiding($f_k$)
8: $ct \leftarrow f_k$
9: return $ct$
```

When generating an invariant polynomial, the encryption algorithm is designed to avoid a scenario where a polynomial with a maximum degree less than k is returned as a ciphertext. The implementation ensures that if any term of the polynomial satisfies the maximum degree k, then the ciphertext's degree also satisfies k, guaranteeing that at least one term matches the k-degree.

For a randomly selected vertex v, if the vertex v' selected during the recursive process does not belong to $N[v]$, the term that includes v and v' is neither removed nor is its degree decreased. To preserve the maximum degree, the algorithm passes the set of vertices to be considered in the reduction, i.e., the set $\widetilde{V} \subset V$ consisting of the vertices selected in the previous step.

The input parameter GRT is a flag indicating whether the generated term's maximum degree should satisfy k.

Algorithm 10 GenPolyD1

```
Input: graph $G = (V, E)$, selected vertices set $\widetilde{V}$, message $m$
Output: invariant polynomial $f$ of the degree 1
1: $n_e \overset{\$}{\longleftarrow} \{1, 2, 3\}$
2: $U \underset{n_e}{\overset{\$}{\longleftarrow}} V \setminus \widetilde{V}$        ▷ $U$ is a set consisted by $n_e$ distinct randomly selected vertices
3: for $i = 1$ to $n_e$
4:     $value \overset{\$}{\longleftarrow} \mathbb{Z}_p^\times$
5:     if $i = n_e$
6:         $value = m - sum$
7:     $sum \leftarrow sum + value$
8:     $f_1 \leftarrow f_1 + value \cdot \Sigma_{v \in N[u_i]} x_v$        ▷ $u_i \in U$
9: return $f_1$
```

Algorithm 11 GenPolyDk

```
Input: graph $G = (V, E)$, guarentee $gua$, selected vertices set $\widetilde{V}$, degree $\widetilde{k}$, message $m$
Output: invariant polynomial $f$ of the degree $\widetilde{k}$
1: if $\widetilde{k} = 1$
2:     if $gua$ = GRT                ▷ guarantees maximum degree
3:         $f_{\widetilde{k}} \leftarrow$ GenPolyD1($G$, $\widetilde{V}$, $m$)
4:     else
5:         $f_{\widetilde{k}} \leftarrow$ GenPolyD1($G$, , $m$)
6:     return $f_{\widetilde{k}}$
7: $v \overset{\$}{\longleftarrow} V$
8: if $gua$ = GRT
9:     while break        ▷ choose and cheak a vertex not in $\widetilde{V}$
10:        if $v \notin \widetilde{V}$
11:            break
12:        $v \overset{\$}{\longleftarrow} V \setminus \widetilde{V}$
13: $s \leftarrow 0$                                ▷ $|N[v]| = 4$
14: if $gua$ = GRT
15:     $s \overset{\$}{\longleftarrow} \{1, 2, 3, 4\}$
16: for $i = 1$ to 4
17:     $value \overset{\$}{\longleftarrow} \mathbb{Z}_p$                ▷ value evaluated for $f_{\widetilde{k}-1}$
18:     if $i = s$        ▷ term that guarantees maximum degree
19:         $\widetilde{V} \leftarrow \widetilde{V} \cup \{u : u \in N[v[s]]\}$
20:         $f_{\widetilde{k}-1} \leftarrow$ GenPolyDk($G$, GRT, $\widetilde{V}$, $\widetilde{k}-1$, $value$)
21:     else
22:         $f_{\widetilde{k}-1} \leftarrow$ GenPolyDk($G$, NOT, , $\widetilde{k}-1$, $value$)
23:     $const = m - value \pmod p$
24:     $f_{\widetilde{k}} \leftarrow f_{\widetilde{k}} + (f_{\widetilde{k}-1} + const) x_{v[s]}$
25: return $f_{\widetilde{k}}$
```

The hiding algorithm operates on the polynomial returned by Algorithm 11.

Algorithm 12 Hiding

```
Input: graph $G = (V, E)$, invariant polynomial $f_k$, maximum degree $k$
Output: reducted invariant polynomial $f'_k$
1: $g_0 \leftarrow f_k$
2: $g_1 \leftarrow$ReduceDegree($G, g_0, k$)
3: $g_2 \leftarrow$SortVariable($g_1, k$)
4: $g_3 \leftarrow$ReduceTerms($G, g_2, k$)
```

```
5: $g_4 \leftarrow$SumCoefficients$(g_3, k)$
6: $f'_k \leftarrow$ShufflingTerms$(g_4, k)$
7: return $f'_k$
```

The following notation is used to describe the subalgorithms within the hiding algorithm:

$\lvert f\rvert$	: the number of terms in f
$f[\alpha]$	: α-th term of f
. $\lvert f[\alpha]\rvert$	: the number of variables in the α-th term of f
$f[\alpha][\beta]$	: vertex for the β-th variable of the α-th term in f
$y_{\alpha,\beta}$	: the β-th variable of the α-th term of f, i.e., $x_{f[\alpha][\beta]}$
c_α	: the coefficient of the α-th term in f.

The detailed algorithms for the SortVariable and ShufflingTerms functions are not included.

Algorithm 13 ReduceDegree

Input: graph $G = (V, E)$, invariant polynomial f_k, maximum degree k

Output: reducted invariant polynomial f'_k

```
1: $f'_k \leftarrow 0$
2: for $\alpha = 1$ to $|f_k|$
3:     $T \leftarrow \{y_{\alpha,1}\}$
4:     term $\leftarrow c_\alpha y_{\alpha,1}$
5:     for $\beta = 2$ to $k-1$
6:         for $\gamma = \beta+1$ to $k$
7:             if $y_{\alpha,\beta} \neq y_{\alpha,\gamma}$
8:                 check $\leftarrow 0$
9:                 if $y_{\alpha,\beta} \in T$
10:                    check $\leftarrow 1$
11:                if check $= 0$
12:                    $T \leftarrow T \cup \{y_{\alpha,\beta}\}$
13:                    term $\leftarrow$ term $\cdot\, y_{\alpha,\beta}$
14:                if $|T| = k$
15:                    break
16:                check $\leftarrow 0$
17:                if $y_{\alpha,\gamma} \in T$
18:                    check $\leftarrow 1$
19:                if check $= 0$
20:                    $T \leftarrow T \cup \{y_{\alpha,\gamma}\}$
21:                    term $\leftarrow$ term $\cdot\, y_{\alpha,\gamma}$
22:                if $|T| = k$
23:                    break
24:    $f'_k \leftarrow f'_k +$ term
25: return $f'_k$
```

Algorithm 14 ReduceTerms

Input: graph $G = (V, E)$, invariant polynomial f_k, maximum degree k

Output: reducted invariant polynomial f_k

```
1: for $\alpha = 1$ to $|f_k|$
2:     for $\beta = 1$ to $k-1$
3:         for $\gamma = \beta$ to $k$
4:             for $i = 1$ to 4
5:                 $u_i \in N[f_k[\alpha][\beta]]$
6:                 for $j = 1$ to 4
7:                     $w_j \in N[f_k[\alpha][\gamma]]$
8:                     if $N[u_i] \cap N[w_j] \neq$
9:                         $f_k[\alpha] \leftarrow 0$
10:                        break
11: return $f_k$
```

Algorithm 15 SumCoefficients

Input: invariant polynomial f_k, maximum degree k

Output: reducted invariant polynomial f'_k

```
1: $f'_k \leftarrow f_k[1]$
2: for $\alpha = 2$ to $|f_k|$
3:     for $\beta = 1$ to $|f'_k|$
4:         dup $\leftarrow 0$
5:         for $\gamma = 1$ to $k$
6:             if $f_k[\alpha][\gamma] = f'_k[\beta][\gamma]$
7:                 dup $\leftarrow$ dup $+ 1$
8:         if dup $= k$
9:             $c'_\beta \leftarrow c'_\beta + c_\alpha$
10:            goto line 2
11:    $f'_k \leftarrow f'_k + f_k[\alpha]$
12: return $f'_k$
```

3) Decryption

The decryption algorithm compares the variables of the terms in the ciphertext polynomial with the variables corresponding to the vertices stored in the PDS. Let U denote the set of vertex variables that form a monomial for any term in the ciphertext. If $|U \cap sk| = |U|$, then the coefficients of that term are meaningful, otherwise they are meaningless. In the algorithm below, *tmp* is a temporary variable that stores information about whether the variables in the monomial are meaningful vertex variables.

In line 6, u is the j-th vertex variable of the i-th term. In line 7, (**if** $u \in PDS$? $1 : 0$) is a ternary operator that returns 1 if u is an element of PDS and 0 otherwise. In the iteration by line 5, *tmp* $= 1$ if all vertex variables in the term are in PDS, and *tmp* $= 0$ if any of them are not, allowing us to decide whether to add c_i to pt in line 8.

$|ct|$ is the number of terms in the ciphertext ct.

Algorithm 16 Decryption

Input: Private key $sk(PDS)$, ciphertext ct

Output: message m

```
1: $pt = 0$
2: for $i = 1$ to $|ct|$
3:     if $c_i \neq 0$
4:         tmp $= 0$
5:         for $j = 1$ to $k$
6:             tmp $\leftarrow$ tmp $\times$ (if $u \in PDS$ ? $1 : 0$)
7:         $pt \leftarrow pt + c_i \times$ tmp
8: $m \leftarrow pt$
9: return $m$
```

D. Example of ciphertext

The example ciphertext ct given below is a degree 7 invariant polynomial, resulting from the encryption of the message $pt = 4410$ using the proposed parameters in Table I. The total number of terms in ct is 22,641, and the size of the ciphertext polynomial is 249,051 bytes. In this paper, we present 50 randomly chosen terms, which constitute 0.2% of the total number of terms in the ciphertext. The full transcript can be viewed at https://github.com/KMURASEofficial/ipcc/tree/master/ipcc7.

$$\begin{aligned}
ct = {} & 485730577\ x_{v_{53}}x_{v_{119}}x_{v_{134}}x_{v_{169}}x_{v_{218}}x_{v_{229}} \\
& +1185214412\ x_{v_{16}}x_{v_{31}}x_{v_{38}}x_{v_{63}}x_{v_{93}}x_{v_{193}}x_{v_{220}} \\
& +1187133452\ x_{v_{53}}x_{v_{56}}x_{v_{120}}x_{v_{134}}x_{v_{169}}x_{v_{250}}x_{v_{255}} \\
& +\ 252775020\ x_{v_{53}}x_{v_{73}}x_{v_{164}}x_{v_{166}}x_{v_{206}}x_{v_{235}}x_{v_{250}} \\
& +1280749315\ x_{v_{7}}x_{v_{60}}x_{v_{83}}x_{v_{105}}x_{v_{191}}x_{v_{217}} \\
& +1393233314\ x_{v_{18}}x_{v_{53}}x_{v_{58}}x_{v_{85}}x_{v_{131}}x_{v_{206}}x_{v_{211}} \\
& +\ 832293056\ x_{v_{51}}x_{v_{60}}x_{v_{63}}x_{v_{196}}x_{v_{213}}x_{v_{247}} \\
& +\ 484242184\ x_{v_{63}}x_{v_{93}}x_{v_{108}}x_{v_{183}}x_{v_{206}} \\
& +\ 93293340\ x_{v_{31}}x_{v_{63}}x_{v_{65}}x_{v_{130}}x_{v_{146}}x_{v_{228}}x_{v_{229}} \\
& +1926696176\ x_{v_{1}}x_{v_{38}}x_{v_{53}}x_{v_{68}}x_{v_{134}}x_{v_{141}} \\
& +1442193836\ x_{v_{31}}x_{v_{63}}x_{v_{67}}x_{v_{124}}x_{v_{125}}x_{v_{166}} \\
& +2017528070\ x_{v_{6}}x_{v_{24}}x_{v_{31}}x_{v_{124}}x_{v_{212}}x_{v_{225}}x_{v_{227}} \\
& +1582964880\ x_{v_{31}}x_{v_{53}}x_{v_{193}}x_{v_{229}}x_{v_{236}}x_{v_{255}} \\
& +\ 403276964\ x_{v_{31}}x_{v_{47}}x_{v_{120}}x_{v_{122}}x_{v_{124}}x_{v_{152}}x_{v_{191}} \\
& +1866835366\ x_{v_{53}}x_{v_{139}}x_{v_{140}}x_{v_{183}}x_{v_{206}}x_{v_{239}}x_{v_{250}} \\
& +\ 79727103\ x_{v_{3}}x_{v_{17}}x_{v_{38}}x_{v_{120}}x_{v_{134}}x_{v_{191}}x_{v_{193}} \\
& +1569908464\ x_{v_{53}}x_{v_{60}}x_{v_{73}}x_{v_{78}}x_{v_{229}}x_{v_{255}} \\
& +\ 733183104\ x_{v_{5}}x_{v_{38}}x_{v_{48}}x_{v_{105}}x_{v_{134}}x_{v_{191}}x_{v_{236}} \\
& +1335896384\ x_{v_{5}}x_{v_{38}}x_{v_{48}}x_{v_{105}}x_{v_{134}}x_{v_{190}}x_{v_{191}} \\
& +1531249596\ x_{v_{31}}x_{v_{51}}x_{v_{63}}x_{v_{169}}x_{v_{181}}x_{v_{193}}x_{v_{247}} \\
& +1597157840\ x_{v_{25}}x_{v_{60}}x_{v_{63}}x_{v_{122}}x_{v_{146}}x_{v_{202}} \\
& +\ 272552863\ x_{v_{78}}x_{v_{134}}x_{v_{143}}x_{v_{157}}x_{v_{171}}x_{v_{191}} \\
& +1034371758\ x_{v_{63}}x_{v_{125}}x_{v_{134}}x_{v_{159}}x_{v_{166}}x_{v_{169}} \\
& +\ 891399469\ x_{v_{7}}x_{v_{83}}x_{v_{85}}x_{v_{131}}x_{v_{191}}x_{v_{206}}x_{v_{231}} \\
& +1943924752\ x_{v_{38}}x_{v_{82}}x_{v_{134}}x_{v_{143}}x_{v_{171}}x_{v_{191}}x_{v_{209}} \\
& +1804266088\ x_{v_{18}}x_{v_{19}}x_{v_{31}}x_{v_{53}}x_{v_{55}}x_{v_{124}} \\
& +1485573648\ x_{v_{7}}x_{v_{83}}x_{v_{166}}x_{v_{191}}x_{v_{206}}x_{v_{217}}x_{v_{244}} \\
& +\ 777739680\ x_{v_{25}}x_{v_{53}}x_{v_{60}}x_{v_{68}}x_{v_{154}} \\
& +\ 987761874\ x_{v_{53}}x_{v_{68}}x_{v_{134}}x_{v_{152}}x_{v_{169}}x_{v_{248}} \\
& +\ 665067856\ x_{v_{53}}x_{v_{60}}x_{v_{68}}x_{v_{76}}x_{v_{211}} \\
& +2069066654\ x_{v_{16}}x_{v_{31}}x_{v_{63}}x_{v_{166}}x_{v_{193}} \\
& +\ 328203030\ x_{v_{63}}x_{v_{75}}x_{v_{134}}x_{v_{139}}x_{v_{166}}x_{v_{169}} \\
& +1804266088\ x_{v_{18}}x_{v_{31}}x_{v_{53}}x_{v_{55}}x_{v_{124}}x_{v_{157}} \\
& +\ 94944640\ x_{v_{25}}x_{v_{26}}x_{v_{33}}x_{v_{63}}x_{v_{93}}x_{v_{183}}x_{v_{206}} \\
& +1280749315\ x_{v_{5}}x_{v_{7}}x_{v_{60}}x_{v_{83}}x_{v_{191}}x_{v_{210}} \\
& +\ 356289050\ x_{v_{4}}x_{v_{60}}x_{v_{120}}x_{v_{191}}x_{v_{194}}x_{v_{210}} \\
& +\ 615627056\ x_{v_{20}}x_{v_{63}}x_{v_{75}}x_{v_{166}}x_{v_{206}}x_{v_{244}} \\
& +\ 840805087\ x_{v_{18}}x_{v_{53}}x_{v_{58}}x_{v_{69}}x_{v_{126}}x_{v_{131}}x_{v_{206}} \\
& +1422995562\ x_{v_{14}}x_{v_{31}}x_{v_{53}}x_{v_{68}} \\
& +\ 83463576\ x_{v_{16}}x_{v_{31}}x_{v_{53}}x_{v_{54}}x_{v_{193}}x_{v_{225}}x_{v_{229}} \\
& +1301019424\ x_{v_{15}}x_{v_{25}}x_{v_{60}}x_{v_{171}}x_{v_{191}} \\
& +1026991232\ x_{v_{25}}x_{v_{31}}x_{v_{33}}x_{v_{63}}x_{v_{93}}x_{v_{164}}x_{v_{228}} \\
& +1399391380\ x_{v_{53}}x_{v_{68}}x_{v_{134}}x_{v_{154}}x_{v_{169}}x_{v_{255}} \\
& +\ 544586942\ x_{v_{60}}x_{v_{63}}x_{v_{122}}x_{v_{146}}x_{v_{174}} \\
& +\ 737529456\ x_{v_{38}}x_{v_{60}}x_{v_{63}}x_{v_{92}}x_{v_{93}}x_{v_{227}} \\
& +\ 308534452\ x_{v_{53}}x_{v_{134}}x_{v_{169}}x_{v_{200}}x_{v_{218}}x_{v_{229}} \\
& +1928067505\ x_{v_{63}}x_{v_{144}}x_{v_{146}}x_{v_{183}}x_{v_{206}}x_{v_{226}}x_{v_{250}} \\
& +\ 415282332\ x_{v_{6}}x_{v_{31}}x_{v_{49}}x_{v_{114}}x_{v_{212}}x_{v_{228}}x_{v_{237}} \\
& +\ 84079493\ x_{v_{31}}x_{v_{53}}x_{v_{164}}x_{v_{193}}x_{v_{226}}x_{v_{227}}x_{v_{250}} \\
& +\ 262643568\ x_{v_{9}}x_{v_{24}}x_{v_{31}}x_{v_{53}}x_{v_{124}}x_{v_{225}}x_{v_{229}} + \cdots
\end{aligned}$$

References

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

[2] M. Fellows, and N. Koblitz, "Combinatorially based cryptography for children (and adults)," *Congressus Numerantium*, vol. 99, pp. 9–41, 1994.

[3] M. Fellows, and N. Koblitz, "Kid krypto," in *Annual International Cryptology Conference,* Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 371–389.

[4] S. Kwon, J. Kang, and Y. Yeom, "Analysis of public-key cryptography using a 3-regular graph with a perfect dominating set," in *2021 IEEE Region 10 Symposium (TENSYMP),* Jeju, Republic of Korea: Grand Hyatt Jeju, 2021, pp. 1–6.

[5] S. Kwon, "A study on Graph-based Public-key Cryptographic Primitives and Transition to Post-quantum Cryptography," M.S. thesis, Dept. Financial Information Security, Kookmin Univ., Seoul, Republic of Korea, 2021.

[6] J. Ryu *et. al*, "IPCC," KpqC Competition Round 1, Available at, https://www.kpqc.or.kr/competition.html.

[7] S. Yoon, "(1,-1,0)-Perfect minus dominating function and its application to the public key cryptosystem," Ph.D. thesis, Dept. of Mathematics Education, Seoul National Univ., Seoul, Republic of Korea, 2001.

[8] NIST Post-Quantum Cryptography Standardization, Available at, https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization.

[9] The IBM Quantum Development Roadmap, Available at, https://www.ibm.com/quantum/roadmap.

[10] H.M. Freedman, "P/NP, and the quantum field computer," *Proceedings of the National Academy of Sciences*, vol. 95, no. 1, pp. 98–101, 1998.

[11] D. Moody, "The 2nd Round of the NIST PQC Standardization Process," the second PQC standardization conference, 2019.

[12] J.A. Bondy, and U.S.R. Murty, *Graph theory with applications*, London: Macmillan, 1976.

[13] T.W. Haynes, S. Hedetniemi, and P. Slater, *Fundamentals of domination in graphs*, Boca Raton: CRC press, 1998.

[14] M.R. Garey and D.S. Johnson, *Computers and intractability: a guide to the theory of np-completeness*, San Francisco: freeman, 1979.

[15] J. Kratochvil, "Regular codes in regular graphs are difficult," *Discrete Mathematics*, vol. 133, no. 1-3, pp. 191–205, Oct. 1994.

[16] F. Rosamond, "Computational thinking enrichment: public-key cryptography," *Informatics in Education-An International Journal.*, vol. 17, no. 1, pp. 93–103, 2018.

[17] D.W. Bange, A.E. Barkauskas, L.H. Host, and P.J. Slater, "Generalized domination and efficient domination in graphs," *Discrete Mathematics*, vol. 159, no. 1-3, pp. 1–11, 1996.

[18] J. Dunbar, W. Goddard, S. Hedetniemi, A. McRae, and M.A. Henning, "The algorithmic complexity of minus domination in graphs," *Discrete Applied Mathematics*, vol. 68, no. 1-2, pp. 73–84, 1996.

[19] X. Bultel, J. Dreier, P. Lafourcade, and M. More, "How to explain modern security concepts to your children," *Cryptologia*, vol. 41, no. 5, pp. 422–447, 2017.

[20] L. Wang, and W. Wang. "An Approximation Algorithm for a Variant of Dominating Set Problem," *Axioms*, vol. 12, no. 6: 506. May. 2023.

[21] J. Kratochvíl, "Perfect codes in general graphs," in *Combinatorics,* COLLOQUIA MATHEMATICA SOCIETATIS JANOS BOLYAI, vol. 52, Amsterdam ; NY : North-Holland Pub. Co., 1988, pp. 357–364.

[22] C. Wouter, and D. Thomas, "An efficient key recovery attack on SIDH," in *Advances in Cryptology – EUROCRYPT 2023,* Lecture Notes in Computer Science, vol. 14008, H. Carmit, and S. Martijn, Cham: Springer Nature Switzerland, 2023, pp. 423–447.

[23] J. Kratochvíl, "Regular codes in regular graphs are difficult," *Discrete Mathematics,* vol. 133, no. 1-3, pp. 191–205, 1994.

[24] M. T. Dickerson, "The functional decomposition of polynomials," Ph.D. thesis, Dept. of Computer Science, Cornell Univ., Ithaca, NY, USA, 1989.

[25] J. Ding, and D. Schmidt, "Rainbow, a new multivariable polynomial signature scheme," in *International conference on applied cryptography and network security*, Heidelberg: Springer Berlin Heidelberg, Berlin, 2005, pp. 164–175.