Secure Key Encapsulation Mechanism with Compact Ciphertext and Public Key from Generalized Srivastava Code

Jayashree Dey(✉) and Ratna Dutta

Department of Mathematics, Indian Institute of Technology Kharagpur, Kharagpur 721302, India
deyjayashree@iitkgp.ac.in, ratna@maths.iitkgp.ernet.in

Abstract. Code-based public key cryptosystems have been found to be an interesting option in the area of Post-Quantum Cryptography. In this work, we present a key encapsulation mechanism (KEM) using a parity check matrix of the Generalized Srivastava code as the public key matrix. Generalized Srivastava codes are privileged with the decoding technique of Alternant codes as they belong to the family of Alternant codes. We exploit the dyadic structure of the parity check matrix to reduce the storage of the public key. Our encapsulation leads to a shorter ciphertext as compared to DAGS proposed by Banegas et al. in Journal of Mathematical Cryptology which also uses Generalized Srivastava code. Our KEM provides IND-CCA security in the random oracle model. Also, our scheme can be shown to achieve post-quantum security in the quantum random oracle model.

Keywords: Key encapsulation mechanism · Generalized Srivastava code · Quasi-dyadic matrix · Alternant decoding

1 Introduction

Cryptography and coding theory are at the core of implementation of telecommunication systems, computational systems and secure networks. Cryptography based on error correcting codes is one of the main approaches to guarantee secure communication in post-quantum world. The security of current widely used classical cryptosystems relies on the difficulty of number theory problems like factorization and the discrete logarithm problem. Shor [21] showed in 1994 that most of these cryptosystems can be broken once sufficiently strong quantum computers become available. Thus, it is necessary to devise alternatives that can survive quantum attacks while offering reasonable performance with solid security guarantees.

Code-based cryptosystems are usually very fast and can be implemented on several platforms, both software and hardware. They do not require special-purpose hardware, specifically no cryptographic co-processors. The security of code based cryptography mainly relies on the following two computational assumptions:

J. H. Seo (Ed.): ICISC 2019, LNCS 11975, pp. 175–193, 2020.
https://doi.org/10.1007/978-3-030-40921-0_11

(i) the hardness of generic decoding [8] which is NP complete and also believed to be hard on average even against quantum adversaries

(ii) the pseudorandomness of the underlying code $\mathcal{C}$ for the construction which states that it is hard to distinguish a random matrix from a generator (or parity check) matrix of $\mathcal{C}$ used as a part of the public key of the system.

Designing practical alternative cryptosystems based on difficulty of decoding unstructured or random codes is currently a major research area. The public key indistinguishability problem strongly depends on the code family. For instance, the McEliece encryption scheme [17] uses binary Goppa codes for which this indistinguishability assumption holds. On the other hand, the assumption does not hold for other families such as Reed Solomon codes, Concatenated codes, Low Density Parity Check (LDPC) codes etc. In [12], Faugere et al. devise a distinguisher for high rate Goppa codes. One of the key challenges in code-based cryptography is to come up with families of codes for which the indistinguishability assumption holds.

Constructing efficient and secure code-based cryptographic scheme is a challenging task. The crucial fact in designing code-based cryptosystems is to use a linear error-correcting code in such a way that the public key is indistinguishable from a random key. A codeword is used as ciphertext of a carefully chosen linear error-correcting code to which random errors are added. The decryptor with the knowledge of a trapdoor can perform fast polynomial time decoding, remove the errors and recover the plaintext. Attackers are reduced to a generic decoding problem and the system remains secure against an adversary equipped with a quantum computer.

Our Contribution. In this paper, we focus on designing an IND-CCA secure efficient code-based KEM that relies on the difficulty of generic decoding problem. Our starting point is the key encapsulation mechanism DAGS [5] that uses the quasi-dyadic structure of Generalized Srivastava (GS) code. Quasi-dyadic structure reduces the public key size remarkably in DAGS while the encapsulation procedure increases the size of ciphertext. We aim to design a KEM with relatively short ciphertext. We deploy the Niederreiter framework to develop our KEM using a syndrome as ciphertext and achieve IND-CCA security in the random oracle model. More precisely, we use the parity check matrix of the Generalized Srivastava code as the public key and utilize its block dyadic structure to reduce the public key size. We consider the syndrome of a vector as the ciphertext header where the vector is formed by parsing two vectors – the first vector is an error vector that is generated by a deterministic error vector generation algorithm and the second vector is constructed from a hash value of a randomly chosen message by the encapsulator. This significantly reduces the ciphertext header size that makes the scheme useful in application with limited communication bandwidth. Also, the use of the parity check matrix directly in computing the ciphertext is more fast and efficient. For decapsulation, we form an equivalent parity check matrix using the secret key to decode the ciphertext header and then proceed to get the decapsulation key. Note that, Generalized Srivastava codes belong to the class of Alternant codes which have benefits of an efficient decoding algorithm. The complexity of decoding is $O(n \log^2 n)$ [20] which is the same as that of Goppa codes where n is the length of the code.

In Table 1, we provide a theoretical comparison of our KEM with other recently proposed code-based KEMs. All the schemes in the table are based on finite fields having characteristic 2. We summarize the following features of our KEM.

- The closest related work to ours is DAGS [5]. Similar to DAGS, we also use quasi-dyadic form of Generalized Srivastava code. However, DAGS uses generator matrix whereas we use parity check matrix. Consequently, in our construction, the ciphertext size is reduced by $k \log_2 q$ bits as compared to DAGS [5] whereas the public key and the secret key sizes remain the same. Furthermore our encapsulation is faster than DAGS.
- The public key sizes in our approach are better than NTS-KEM [2], Classic McEliece [9] and BIG QUAKE [6]. Although the BIKE variants are efficient in terms of key sizes and achieve IND-CCA security, they still suffer from small decoding failure rate. The erlier BIKE variants proposed in [3] have a non-negligible decoding failure rate and only attain IND-CPA security.

Table 1. Summary of IND-CCA secure KEMs using random oracles

Scheme	pk size (in bits)	sk size (in bits)	CT size (in bits)	Code used	Cyclic/Dyadic	Correctness error
NTS-KEM [2]	$(n-k)k$	$2(n-k+r)m+nm+r$	$(n-k+r)$	Binary Goppa code	–	No
BIKE-1 [4]	n	$n+w\cdot\lceil\log_2 k\rceil$	n	MDPC code	Quasi-Cyclic	Yes
BIKE-2 [4]	k	$n+w\cdot\lceil\log_2 k\rceil$	k	MDPC code	Quasi-Cyclic	Yes
BIKE-3 [4]	n	$n+w\cdot\lceil\log_2 k\rceil$	n	MDPC code	Quasi-Cyclic	Yes
Classic McEliece [9]	$k(n-k)$	$n+mt+mn$	$(n-k)+r$	Binary Goppa code	–	No
BIG QUAKE [6]	$\frac{k}{\ell}(n-k)$	$mt+mn$	$(n-k)+2r$	Binary Goppa code	Quasi-Cyclic	No
DAGS [5]	$\frac{k}{s}(n-k)\log_2 q$	$2mn\log_2 q$	$[n+k']\log_2 q$	GS code	Quasi-Dyadic	No
This work	$\frac{k}{s}(n-k)\log_2 q$	$2mn\log_2 q$	$[k'+(n-k)]\log_2 q$	GS code	Quasi-Dyadic	No

pk = Public key, sk = Secret key, CT = Ciphertext, k = dimension of the code, n = length of the code, ℓ = length of each blocks, t = error correcting capacity, $k' < k, s, r, w, p_1, p_2$ are positive integers ($\ell << s$), $s = 2^{p_2}$, $q = 2^{p_1}$, m = the degree of field extension, r = the desired key length, GS = Generalized Srivastava, MDPC = Moderate Density Parity Check

In the comparison table, we mostly highlight the KEMs which rely on the error correcting codes that belong to the class of Alternant codes except BIKE variants which use QC (Quasi-Cyclic)-MDPC codes. We exclude the schemes like LEDAkem, RLCE-KEM, LAKE, Ouroboros-R, LOCKER, QC-MDPC, McNie etc. In fact, the schemes LAKE, Ouroboros-R, LOCKER use rank metric codes (Low Rank Parity Check (LRPC) codes) while RLCE-KEM is based on random linear codes and McNie relies on any error-correcting code, specially QC-LRPC codes. LEDAkem uses QC-LDPC codes and has a small decoding failure rate.

Moreover, it has risks in case of keypair reuse which may cause a reaction attack [11] for some particular instances. The schemes proposed in [1] are also kept out as both HQC and RQC are constructed for any decodable linear code. Also, HQC has a decryption failure and RQC uses rank metric codes. The protocol QC-MDPC may have a high decoding failure rate for some particular parameters which enhances the risk of GJS attack [14]. The scheme CAKE is another important KEM which is merged with another independent construction Ouroboros to obtain BIKE [3].

Organization of the Paper. This rest of the paper is organized as follows. In Sect. 2, we describe necessary background related to our work. We illustrate our approach to design a KEM in Sect. 3 and discuss its security in Sect. 4. Finally, we conclude in Sect. 5.

2 Preliminaries

In this section, we provide mathematical background and preliminaries that are necessary to follow the discussion in the paper.

Notation. We use the notation $\mathsf{x} \xleftarrow{U} X$ for choosing a random element from a set or distribution, $\mathsf{wt}(\mathbf{x})$ to denote the weight of a vector $\mathbf{x}$, $(\mathbf{x}||\mathbf{y})$ for the concatenation of the two vectors $\mathbf{x}$ and $\mathbf{y}$. The matrix I_n is the $n \times n$ identity matrix. We let $\mathbb{Z}^+$ to represent the set $\{a \in \mathbb{Z} | a \geq 0\}$ where $\mathbb{Z}$ is the set of integers. We denote the transpose of a matrix A by A^T and concatenation of the two matrices A and B by $[A|B]$. The uniform distribution over $(n-k) \times n$ random q-ary matrices is denoted by $U_{(n-k)\times n}$.

2.1 Hardness Assumptions

Definition 1 ((Decision) (q-ary) Syndrome Decoding (SD) Problem [8])**.** Given a full-rank matrix $H_{(n-k)\times n}$ over $\mathsf{GF}(q)$, a vector $\mathbf{e} \in (\mathsf{GF}(q))^n$ and a non-negative integer w, is it possible to distinguish between a random syndrome $\mathbf{s}$ and the syndrome $H\mathbf{e}^T$ associated to a w-weight vector $\mathbf{e}$?

Suppose $\mathcal{D}$ is a probabilistic polynomial time algorithm. For every positive integer λ, we define the advantage of $\mathcal{D}$ in solving the decisional SD problem by

$$\begin{aligned}\mathsf{Adv}^{\mathsf{DEC}}_{\mathcal{D},\mathsf{SD}}(\lambda) = |\Pr[\mathcal{D}(H, H\mathbf{e}^T) = 1 \mid \mathbf{e} \in (\mathsf{GF}(q))^n, H \xleftarrow{U} U_{(n-k)\times n}] \\ - \Pr[\mathcal{D}(H, \mathbf{s}) = 1 \mid \mathbf{s} \xleftarrow{U} U_{(n-k)\times 1}, H \xleftarrow{U} U_{(n-k)\times n}]|.\end{aligned}$$

Also, we define $\mathsf{Adv}^{\mathsf{DEC}}_{\mathsf{SD}}(\lambda) = \max_{\mathcal{D}}[\mathsf{Adv}^{\mathsf{DEC}}_{\mathcal{D},\mathsf{SD}}(\lambda)]$ where the maximum is taken over all $\mathcal{D}$. The decisional SD problem is said to be hard if $\mathsf{Adv}^{\mathsf{DEC}}_{\mathsf{SD}}(\lambda) < \delta$ where $\delta > 0$ is arbitrarily small.

In addition, some code based schemes require the following computational assumption. Most of the schemes output a public key that is either a generator matrix or a parity check matrix by running key generation algorithm.

Definition 2 (Indistinguishability of public key matrix H [18]). Let $\mathcal{D}$ be a probabilistic polynomial time algorithm and $\mathsf{PKE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a public key encryption scheme that uses an $(n-k) \times n$ matrix H as a public key over $\mathsf{GF}(q)$. For every positive integer λ, we define the advantage of $\mathcal{D}$ in distinguishing the public key matrix H from a random matrix R as $\mathsf{Adv}^{\mathsf{IND}}_{\mathcal{D},H}(\lambda) = \Pr[\mathcal{D}(H) = 1 | (\mathsf{pk} = H, \mathsf{sk}) \longleftarrow \mathsf{PKE.KeyGen}(\mathsf{param}), \mathsf{param} \longleftarrow \mathsf{PKE.Setup}(1^{\lambda})] - \Pr[\mathcal{D}(R) = 1 | R \xleftarrow{U} U_{(n-k)\times n}]$.

We define $\mathsf{Adv}^{\mathsf{IND}}_{H}(\lambda) = \max_{\mathcal{D}}[\mathsf{Adv}^{\mathsf{IND}}_{\mathcal{D},H}(\lambda)]$ where the maximum is over all $\mathcal{D}$. The matrix H is said to be indistinguishable if $\mathsf{Adv}^{\mathsf{IND}}_{H}(\lambda) < \delta$ where $\delta > 0$ is arbitrarily small.

2.2 Basic Definitions from Coding Theory

Definition 3 (Dyadic Matrix and Quasi-Dyadic Matrix [7]). Given a ring $\mathbf{R}$ and a vector $\mathbf{h} = (h_0, h_1, \ldots, h_{n-1}) \in \mathbf{R}^n$, the *dyadic* matrix $\Delta(\mathbf{h}) \in \mathbf{R}^{n\times n}$ is a symmetric matrix having components $\Delta_{ij} = h_{i\oplus j}$ where $\oplus$ stands for bitwise exclusive-or. The vector $\mathbf{h}$ is called a signature of the dyadic matrix. The signature of a dyadic matrix forms its first row. A matrix is called *quasi-dyadic* if it is a block matrix whose component blocks are $s \times s$ dyadic submatrices. An $s \times s$ dyadic matrix block can be generated from its first row.

Generating the dyadic signature [7]: A valid dyadic signature $\mathbf{h} = (h_0, h_1, \ldots, h_{n-1})$ over $\mathbf{R} = \mathsf{GF}(q^m)$ is derived using Algorithm 1.

Definition 4 (The Generalized Srivastava (GS) Code [16]). Let $m, n, s, t \in \mathbb{N}$ and q be a prime power. Let $\alpha_1, \alpha_2, \ldots, \alpha_n, w_1, w_2, \ldots, w_s$ be $n+s$ distinct elements of $\mathsf{GF}(q^m)$ and $z_1, z_2, \ldots, z_n$ be nonzero elements of $\mathsf{GF}(q^m)$. The Generalized Srivastava (GS) code of length n is a linear code with $st \times n$ parity-check matrix of the form $H = \begin{bmatrix} H_1 & H_2 & \cdots & H_s \end{bmatrix}^T$ where

$$H_i = \begin{bmatrix} \frac{z_1}{\alpha_1 - w_i} & \frac{z_2}{\alpha_2 - w_i} & \cdots & \frac{z_n}{\alpha_n - w_i} \\ \frac{z_1}{(\alpha_1 - w_i)^2} & \frac{z_2}{(\alpha_2 - w_i)^2} & \cdots & \frac{z_n}{(\alpha_n - w_i)^2} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{z_1}{(\alpha_1 - w_i)^t} & \frac{z_2}{(\alpha_2 - w_i)^t} & \cdots & \frac{z_n}{(\alpha_n - w_i)^t} \end{bmatrix}$$

is a $t \times n$ matrix block. The code is of length $n \le q^m - s$, dimension $k \ge n - mst$ and minimum distance $d \ge st + 1$. It can correct at most $w = \left\lfloor \frac{d-1}{2} \right\rfloor = \frac{st}{2}$ errors and is an Alternant code. In the parity check matrix

$$H = \begin{bmatrix} y_1 g_1(\alpha_1) & y_2 g_1(\alpha_2) & \cdots & y_n g_1(\alpha_n) \\ y_1 g_2(\alpha_1) & y_2 g_2(\alpha_2) & \cdots & y_n g_2(\alpha_n) \\ y_1 g_3(\alpha_1) & y_2 g_3(\alpha_2) & \cdots & y_n g_3(\alpha_n) \\ \cdots & \cdots & \cdots & \cdots \\ y_1 g_r(\alpha_1) & y_2 g_r(\alpha_2) & \cdots & y_n g_r(\alpha_n) \end{bmatrix}$$

Algorithm 1. Constructing a dyadic signature

Input: q, m, s, n.
Output: A dyadic signature $\mathbf{h} = (h_0, h_1, \ldots, h_{n-1})$ over $\mathsf{GF}(q^m)$.

repeat
  $X = \mathsf{GF}(q^m) \setminus \{0\}$; $\widehat{h}_0 \xleftarrow{U} X$; $X = X \setminus \{\widehat{h}_0\}$;
  for ($l = 0$ **to** $\lfloor \log q^m \rfloor$) **do**
    $i = 2^l$; $\widehat{h}_i \xleftarrow{U} X$; $X = X \setminus \{\widehat{h}_i\}$;
    for ($j = 1$ **to** $i - 1$) **do**
      if ($\widehat{h}_i \neq 0 \wedge \widehat{h}_j \neq 0 \wedge \frac{1}{\widehat{h}_i} + \frac{1}{\widehat{h}_j} + \frac{1}{\widehat{h}_0} \neq 0$) **then**
        $\widehat{h}_{i+j} = 1/(\frac{1}{\widehat{h}_i} + \frac{1}{\widehat{h}_j} + \frac{1}{\widehat{h}_0})$;
      else
        $\widehat{h}_{i+j} = 0$; // undefined entry
      end if
      $X = X \setminus \{\widehat{h}_{i+j}\}$;
    end for
  end for
  $c = 0$;
  if ($0 \notin \{\widehat{h}_0, \widehat{h}_1, \ldots, \widehat{h}_{s-1}\}$) **then**
    $b_0 = 0; c = 1$; $B_0 = \{\widehat{h}_0, \widehat{h}_1, \ldots, \widehat{h}_{s-1}\}$;
    for ($j = 1$ **to** $\lfloor q^m/s \rfloor - 1$) **do**
      if ($0 \notin \{\widehat{h}_{js}, \widehat{h}_{js+1}, \ldots, \widehat{h}_{(j+1)s-1}\}$) **then**
        $b_c = j$; $c = c + 1$; $B_c = \{\widehat{h}_{js}, \widehat{h}_{js+1}, \ldots, \widehat{h}_{(j+1)s-1}\}$;
      end if
    end for
  end if
until ($cs \geq n$)
return $\mathbf{h} = (h_0, h_1, \ldots, h_{n-1}) = (B_0, B_1, \ldots, B_{c-1})$

where $g_i(x) = c_{i1} + c_{i2}x + \cdots + c_{ir}x^{r-1}$, $i = 1, 2, \ldots, r$ is a polynomial of degree $< r$ over $\mathsf{GF}(q^m)$ for the Alternant code $\mathcal{A}(\boldsymbol{\alpha}, \mathbf{y})$, let $r = st$. Also set $g_{(l-1)t+k}(x) = \prod_{j=1}^{s}(x - w_j)^t/(x - w_l)^k$, $l = 1, 2, \ldots, s$ and $k = 1, 2, \ldots, t$ and $y_i = z_i/\prod_{j=1}^{s}(\alpha_i - w_j)^t$, $i = 1, 2, \ldots, n$ so that $y_i g_{(l-1)t+k}(\alpha_i) = z_i/(\alpha_i - w_l)^k$. The resulting code will be a Generalized Srivastava code.

3 Our KEM Protocol

We construct a key encapsulation mechanism $\mathsf{KEM} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps})$ as described below.

- $\mathsf{KEM.Setup}(1^\lambda) \longrightarrow \mathsf{param}$: Taking security parameter λ as input, a trusted authority proceeds as follows to generate the global public parameters param.

(i) Sample $n_0, p_1, p_2, m \in \mathbb{Z}^+$, set $q = 2^{p_1}$, $s = 2^{p_2}$ and $n = n_0 s < q^m$.
(ii) Select $t \in \mathbb{Z}^+$ such that $mst < n$. Set $w \leq st/2$ and $k = n - mst$.
(iii) Sample $k' \in \mathbb{Z}^+$ with $k' < k$.
(iv) Select three cryptographically secure hash functions $\mathcal{G} : (\mathsf{GF}(q))^{k'} \longrightarrow (\mathsf{GF}(q))^k$, $\mathcal{H} : (\mathsf{GF}(q))^{k'} \longrightarrow (\mathsf{GF}(q))^{k'}$ and $\mathcal{H}' : \{0,1\}^* \longrightarrow \{0,1\}^r$ where $r \in \mathbb{Z}^+$ denotes the desired key length.

(v) Publish the global parameters $\mathsf{param} = (n, n_0, k, k', w, q, s, t, r, m, \mathcal{G}, \mathcal{H}, \mathcal{H}')$.

- $\mathsf{KEM.KeyGen}(\mathsf{param}) \longrightarrow (\mathsf{pk}, \mathsf{sk})$: A user on input param, performs the following steps to generate the public key pk and secret key sk.

(i) Generate dyadic signature $\mathbf{h} = (h_0, h_1, \ldots, h_{n-1})$ using Algorithm 1 where $h_i \in \mathsf{GF}(q^m)$ for $i = 0, 1, \ldots, n-1$.

(ii) Select $\omega \xleftarrow{U} \mathsf{GF}(q^m)$ with $\omega \neq \frac{1}{h_j} + \frac{1}{h_0}$, $j = 0, 1, \ldots, n-1$ and compute $u_i = \frac{1}{h_i} + \omega$, $i = 0, 1, \ldots, s-1$ and $v_j = \frac{1}{h_j} + \frac{1}{h_0} + \omega$, $j = 0, 1, \ldots, n-1$. Set $\mathbf{u} = (u_0, u_1, \ldots, u_{s-1})$ and $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$.

(iii) Construct $st \times n$ quasi-dyadic matrix $A = \begin{bmatrix} A_1 & A_2 & \cdots & A_t \end{bmatrix}^T$ where

$$A_i = \begin{bmatrix} \frac{1}{(u_0-v_0)^i} & \frac{1}{(u_0-v_1)^i} & \cdots & \frac{1}{(u_0-v_{n-1})^i} \\ \frac{1}{(u_1-v_0)^i} & \frac{1}{(u_1-v_1)^i} & \cdots & \frac{1}{(u_1-v_{n-1})^i} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{1}{(u_{s-1}-v_0)^i} & \frac{1}{(u_{s-1}-v_1)^i} & \cdots & \frac{1}{(u_{s-1}-v_{n-1})^i} \end{bmatrix} = \begin{bmatrix} \frac{1}{(v_0-u_0)^i} & \frac{1}{(v_1-u_0)^i} & \cdots & \frac{1}{(v_{n-1}-u_0)^i} \\ \frac{1}{(v_0-u_1)^i} & \frac{1}{(v_1-u_1)^i} & \cdots & \frac{1}{(v_{n-1}-u_1)^i} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{1}{(v_0-u_{s-1})^i} & \frac{1}{(v_1-u_{s-1})^i} & \cdots & \frac{1}{(v_{n-1}-u_{s-1})^i} \end{bmatrix}$$

is the $s \times n$ matrix block that can be written as $A_i = [\hat{A}_{i_1} | \hat{A}_{i_2} | \cdots | \hat{A}_{i_{n_0}}]$. Each block $\hat{A}_{i_k}$ is an $s \times s$ dyadic matrix for $k = 1, 2, \ldots, n_0$. For instance, take the first block

$$\hat{A}_{i_1} = \begin{bmatrix} \frac{1}{(u_0-v_0)^i} & \frac{1}{(u_0-v_1)^i} & \cdots & \frac{1}{(u_0-v_{s-1})^i} \\ \frac{1}{(u_1-v_0)^i} & \frac{1}{(u_1-v_1)^i} & \cdots & \frac{1}{(u_1-v_{s-1})^i} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{1}{(u_{s-1}-v_0)^i} & \frac{1}{(u_{s-1}-v_1)^i} & \cdots & \frac{1}{(u_{s-1}-v_{s-1})^i} \end{bmatrix}$$

which is symmetric as $u_i - v_j = \frac{1}{h_i} + \frac{1}{h_j} + \frac{1}{h_0} = u_j - v_i$ and dyadic of order s as the $s \times s$ matrix

$$\begin{bmatrix} \frac{1}{(u_0-v_0)} & \frac{1}{(u_0-v_1)} & \frac{1}{(u_0-v_2)} & \cdots & \frac{1}{(u_0-v_{s-1})} \\ \frac{1}{(u_1-v_0)} & \frac{1}{(u_1-v_1)} & \frac{1}{(u_1-v_2)} & \cdots & \frac{1}{(u_1-v_{s-1})} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \frac{1}{(u_{s-1}-v_0)} & \frac{1}{(u_{s-1}-v_1)^i} & \frac{1}{(u_{s-1}-v_2)^i} & \cdots & \frac{1}{(u_{s-1}-v_{s-1})} \end{bmatrix}$$

$$= \begin{bmatrix} h_0 & h_1 & h_2 & \cdots & h_{s-1} \\ h_1 & h_0 & h_3 & \cdots & h_{s-2} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ h_{s-1} & h_{s-2} & h_{s-3} & \cdots & h_0 \end{bmatrix} = \begin{bmatrix} h_{0\oplus 0} & h_{0\oplus 1} & h_{0\oplus 2} & \cdots & h_{0\oplus(s-1)} \\ h_{1\oplus 0} & h_{1\oplus 1} & h_{1\oplus 2} & \cdots & h_{1\oplus(s-1)} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ h_{(s-1)\oplus 0} & h_{(s-1)\oplus 1} & h_{(s-1)\oplus 2} & \cdots & h_{(s-1)\oplus(s-1)} \end{bmatrix}$$

can be derived from the first row of the block using the relation $\frac{1}{h_{i\oplus j}} = \frac{1}{h_i} + \frac{1}{h_j} + \frac{1}{h_0}$. Since the powering process acts on every single element, $\hat{A}_{i_1}$ preserves its dyadic structure.

(iv) Choose $z_{is} \xleftarrow{U} \mathsf{GF}(q^m)$, $i = 0, 1, \ldots, n_0 - 1$ and set $z_{is+p} = z_{is}$, $p = 0, 1, \ldots, s-1$ where $n = n_0 s$. Also set

$$\mathbf{z} = (z_{0s}, z_{0s+1}, \ldots, z_{0s+s-1}; z_{1s}, z_{1s+1}, \ldots, z_{1s+s-1}; \ldots; z_{(n_0-1)s}, z_{(n_0-1)s+1}, \ldots, z_{(n_0-1)s+s-1}) = (z_0, z_1, \ldots, z_{n-1}) \in (\mathsf{GF}(q^m))^n.$$

(v) Compute $y_j = z_j / \prod_{i=0}^{s-1} (u_i - v_j)^t$ for $j = 0, 1, \ldots, n-1$ and set $\mathbf{y} = (y_0, y_1, \ldots, y_{n-1}) \in (\mathsf{GF}(q^m))^n$.

(vi) Construct $st \times n$ matrix $B = \begin{bmatrix} B_1 & B_2 & \cdots & B_t \end{bmatrix}^T$ where

$$B_i = \begin{bmatrix} \frac{z_0}{(v_0 - u_0)^i} & \frac{z_1}{(v_1 - u_0)^i} & \cdots & \frac{z_{n-1}}{(v_{n-1} - u_0)^i} \\ \frac{z_0}{(v_0 - u_1)^i} & \frac{z_1}{(v_1 - u_1)^i} & \cdots & \frac{z_{n-1}}{(v_{n-1} - u_1)^i} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{z_0}{(v_0 - u_{s-1})^i} & \frac{z_1}{(v_1 - u_{s-1})^i} & \cdots & \frac{z_{n-1}}{(v_{n-1} - u_{s-1})^i} \end{bmatrix}$$

is $s \times n$ matrix block. Sample a permutation matrix P of order st and compute $st \times n$ matrix $\overline{B} = PB$. The matrix $\overline{B}$ is a parity-check matrix of the GS code equivalent to its parity check matrix as in Definition 4, Subsect. 2.2.

(vii) Project $\overline{B}$ onto $\mathsf{GF}(q)$ using the $\mathsf{co\text{-}trace}$ function to form a $mst \times n$ matrix C where $\mathsf{co\text{-}trace}$ function converts an element of $\mathsf{GF}(q^m)$ to an element of $\mathsf{GF}(q)$ with respect to a basis of $\mathsf{GF}(q^m)$ over $\mathsf{GF}(q)$. For $\mathbf{a} \in \mathsf{GF}(q^m)$, $\mathsf{co\text{-}trace}(\mathbf{a}) = (a_0, a_1, \ldots, a_{m-1}) \in (\mathsf{GF}(q))^m$ satisfying $< \mathbf{g}, \mathbf{a} >= a_0 + a_1 q + a_2 q^2 + \cdots + a_{m-1} q^{m-1}$ where $a_i \in \mathsf{GF}(q)$ and $\mathbf{g} = (1, q, q^2, \ldots, q^{m-1})$ is a basis of $\mathsf{GF}(q^m)$ over $\mathsf{GF}(q)$. Thus if $\overline{B} = (\overline{b}_{ij})$ where $\overline{b}_{ij} \in \mathsf{GF}(q^m)$, then $C = (c_{ij})$ is obtained from $\overline{B}$ by replacing $\overline{b}_{ij}$ by $\mathsf{co\text{-}trace}(\overline{b}_{ij})$. Write the matrix C in the systematic form $(M | I_{n-k})$ where M is $(n-k) \times k$ matrix with $k = n - mst$. Note that, the z_i are chosen to be equally having s-length block and all the operations during the row reduction are performed block-wise. Consequently, the dyadic structure is maintained in C and in particular in M. Let $M = \begin{bmatrix} M_{0,0} & M_{0,1} & \cdots & M_{0,\frac{k}{s}-1} \\ M_{1,0} & M_{1,1} & \cdots & M_{1,\frac{k}{s}-1} \\ \cdots & \cdots & \cdots & \cdots \\ M_{mt-1,0} & M_{mt-1,1} & \cdots & M_{mt-1,\frac{k}{s}-1} \end{bmatrix}$ where each block matrix $M_{i,j}$ is $s \times s$ dyadic matrix with dyadic signature $\boldsymbol{\psi}_{i,j} \in (\mathsf{GF}(q))^s$ which is the first row of $M_{i,j}$, $i = 0, 1, \ldots, mt-1$, $j = 0, 1, \ldots, \frac{k}{s} - 1$.

(viii) Publish the public key $\mathsf{pk} = \{\boldsymbol{\psi}_{i,j} \mid i = 0, 1, \ldots, mt-1,\ j = 0, 1, \ldots, \frac{k}{s} - 1\}$ and keep the secret key $\mathsf{sk} = (\mathbf{v}, \mathbf{y})$ to itself.

- $\mathsf{KEM.Encaps}(\mathsf{param}, \mathsf{pk}) \longrightarrow (\mathsf{CT}, K)$: Given system parameters param and public key pk, an encapsulator proceeds as follows to generate a ciphertext header $\mathsf{CT} \in (\mathsf{GF}(q))^{n-k+k'}$ and an encapsulation key $K \in \{0,1\}^r$.

Algorithm 2. Error vector derivation

Input: q, n, a seed $\bar{\mathbf{s}} = (\bar{s}_0, \bar{s}_1, \ldots, \bar{s}_{k-1}) \in (\mathsf{GF}(q))^k$, a weight w, a function $\mathcal{F} : \mathsf{GF}(q) \longrightarrow \mathbb{Z}^+$.
Output: An error vector $\mathbf{e}$ of length n and weight w.

1: $\mathbf{s} = (s_0, s_1, \ldots, s_{n-1})$=$\mathsf{Expand}(\bar{\mathbf{s}})$; // Expand is an expansion function
2: $j = 0$; $\mathsf{temp} = 0$; $d = 0$; $\mathbf{e} = \mathbf{0}$; $\mathbf{v} = \mathbf{0}$;
3: **for** $(i = 0$ **to** $n - 1)$ **do**
4:     **if** $(s_i \bmod q \neq 0)$ **then**
5:         **if** $(j = w)$ **then**
6:             **break**;
7:         **end if**
8:         $\mathsf{temp} = \mathcal{F}(s_d) \bmod n$; $d = d + 1$;
9:         **for** $(\nu = 0$ **to** $j)$ **do**
10:             **if** $(\mathsf{temp} = v_\nu)$ **then**
11:                 **goto** step 9;
12:             **end if**
13:         **end for**
14:         $v_j = \mathsf{temp}$; $e_{\mathsf{temp}} = s_i \bmod q$; $\mathsf{temp} = 0$; $j = j + 1$;
15:     **end if**
16: **end for**
17: **return** $\mathbf{e} = (e_0, e_1, \ldots, e_{n-1})$

(i) Sample $\mathbf{m} \xleftarrow{U} (\mathsf{GF}(q))^{k'}$ and compute $\mathbf{r} = \mathcal{G}(\mathbf{m}) \in (\mathsf{GF}(q))^k$, $\mathbf{d} = \mathcal{H}(\mathbf{m}) \in (\mathsf{GF}(q))^{k'}$ where $\mathcal{G}$ and $\mathcal{H}$ are the hash functions given in param.

(ii) Parse $\mathbf{r}$ as $\mathbf{r} = (\boldsymbol{\rho}||\boldsymbol{\sigma})$ where $\boldsymbol{\rho} \in (\mathsf{GF}(q))^{k-k'}$, $\boldsymbol{\sigma} \in (\mathsf{GF}(q))^{k'}$. Set $\boldsymbol{\mu} = (\boldsymbol{\rho}||\mathbf{m}) \in (\mathsf{GF}(q))^k$.

(iii) Run Algorithm 2 to generate a error vector $\mathbf{e}$ of length $n - k$ and weight $w - \mathsf{wt}(\boldsymbol{\mu})$ using $\boldsymbol{\sigma}$ as a seed. Note that Algorithm 2 uses an expansion function[1]. Set $\mathbf{e}' = (\mathbf{e}||\boldsymbol{\mu}) \in (\mathsf{GF}(q))^n$.

(iv) Using the public key $\mathsf{pk}=\{\boldsymbol{\psi}_{i,j} \mid i = 0, 1, \ldots, mt - 1,\ j = 0, 1, \ldots, \frac{k}{s} - 1\}$, compute $s \times s$ dyadic matrix $M_{i,j}$ with signature $\boldsymbol{\psi}_{i,j} \in (\mathsf{GF}(q))^s$ and reconstruct the parity check matrix $H = (M|I_{n-k})$ for the the GS code where $n - k = mst$ and

$$M = \begin{bmatrix} M_{0,0} & M_{0,1} & \cdots & M_{0,\frac{k}{s}-1} \\ M_{1,0} & M_{1,1} & \cdots & M_{1,\frac{k}{s}-1} \\ \cdots & \cdots & \cdots & \cdots \\ M_{mt-1,0} & M_{mt-1,1} & \cdots & M_{mt-1,\frac{k}{s}-1} \end{bmatrix}$$

(v) Compute the syndrome $\mathbf{c} = H(\mathbf{e}')^T$ and the encapsulation key $K = \mathcal{H}'(\mathbf{m})$ where $\mathcal{H}'$ is the hash function given in param.

(vi) Publish the ciphertext header $\mathsf{CT} = (\mathbf{c}, \mathbf{d})$ and keep K as secret.

- $\mathsf{KEM.Decaps}(\mathsf{param}, \mathsf{sk}, \mathsf{CT}) \longrightarrow K$: On receiving a ciphertext header $\mathsf{CT} = (\mathbf{c}, \mathbf{d})$, a decapsulator executes the following steps using public parameters param and its secret key $\mathsf{sk} = (\mathbf{v}, \mathbf{y})$ where $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ and $\mathbf{y} = (y_0, y_1, \ldots, y_{n-1})$.

(i) First proceed as follows to decode $\mathbf{c}$ and find error vector $\mathbf{e}''$ of length n and weight w:

[1] For example, kangaroo twelve function [10] can be used as an expansion function.

(a) Use $\mathsf{sk} = (\mathbf{v}, \mathbf{y})$ to construct $st \times n$ matrix H' in the form

$$\begin{bmatrix} y_0 & y_1 & \cdots & y_{n-1} \\ v_0 y_0 & v_1 y_1 & \cdots & v_{n-1} y_{n-1} \\ v_0^2 y_0 & v_1^2 y_1 & \cdots & v_{n-1}^2 y_{n-1} \\ \cdots & \cdots & \cdots & \cdots \\ v_0^{st-1} y_0 & v_1^{st-1} y_1 & \cdots & v_{n-1}^{st-1} y_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ v_0 & v_1 & \cdots & v_{n-1} \\ v_0^2 & v_1^2 & \cdots & v_{n-1}^2 \\ \cdots & \cdots & \cdots & \cdots \\ v_0^{st-1} & v_1^{st-1} & \cdots & v_{n-1}^{st-1} \end{bmatrix} \begin{bmatrix} y_0 & 0 & \cdots & 0 \\ 0 & y_1 & \cdots & 0 \\ 0 & 0 & y_2 & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & y_{n-1} \end{bmatrix}.$$

Note that, H' is a parity check matrix in alternant form of the GS code over $\mathsf{GF}(q^m)$ whereas the matrix $H = [M|I_{(n-k)}]$ constructed during KEM.KeyGen or KEM.Encaps is a parity check matrix in the systematic form of the GS code over $\mathsf{GF}(q)$.

(b) As the GS code is an Alternant code, the parity check matrix H' is used to decode $\mathbf{c}$ by first computing the syndrome $S = H'(\mathbf{c}||\mathbf{0})^T$ where $\mathbf{0}$ represents the vector $(0, 0, \ldots, 0)$ of length k and then by running decoding algorithm for the Alternant code to find the error locator polynomial $\omega(z) = \sum_{\nu=1}^{w} Y_\nu y_{i_\nu} \prod_{\mu=1, \mu\neq\nu}^{w} (1 - X_\mu z)$ and error evaluator polynomial $\sigma(z) = \prod_{i=1}^{w}(1 - X_i z)$. Let $X_1 = v_{i_1}, X_2 = v_{i_2}, \ldots, X_w = v_{i_w}$ be the error locations and $Y_1 = e_{X_1}, Y_2 = e_{X_2}, \ldots, Y_w = e_{X_w}$ be the error values.

(c) Set $\mathbf{e}'' = (e_1, e_2, \ldots, e_n)$ with $e_j = \begin{cases} 0 & \text{if } j \neq X_i, 1 \leq i \leq w \\ Y_i & \text{if } j = X_i, 1 \leq i \leq w \end{cases}$.

(ii) Let $\mathbf{e}'' = (\mathbf{e}_0||\boldsymbol{\mu}') \in (\mathsf{GF}(q))^n$ and $\boldsymbol{\mu}' = (\boldsymbol{\rho}'||\mathbf{m}') \in (\mathsf{GF}(q))^k$ where $\mathbf{e}_0 \in (\mathsf{GF}(q))^{n-k}$, $\boldsymbol{\rho}' \in (\mathsf{GF}(q))^{k-k'}$, $\mathbf{m}' \in (\mathsf{GF}(q))^{k'}$.

(iii) Compute $\mathbf{r}' = \mathcal{G}(\mathbf{m}') \in (\mathsf{GF}(q))^k$ and $\mathbf{d}' = \mathcal{H}(\mathbf{m}') \in (\mathsf{GF}(q))^{k'}$ where $\mathcal{G}$ and $\mathcal{H}$ are the hash functions given in param.

(iv) Parse $\mathbf{r}'$ as $\mathbf{r}' = (\boldsymbol{\rho}''||\boldsymbol{\sigma}')$ where $\boldsymbol{\rho}'' \in (\mathsf{GF}(q))^{k-k'}$, $\boldsymbol{\sigma}' \in (\mathsf{GF}(q))^{k'}$.

(v) Run Algorithm 2 to generate deterministically an error vector $\mathbf{e}_0'$ of length $n - k$ and weight $w - \mathsf{wt}(\boldsymbol{\mu}')$ using $\boldsymbol{\sigma}'$ as seed.

(vi) If $(\mathbf{e}_0 \neq \mathbf{e}_0') \vee (\boldsymbol{\rho}' \neq \boldsymbol{\rho}'') \vee (\mathbf{d} \neq \mathbf{d}')$, output $\perp$ indicating decapsulation failure. Otherwise, compute the encapsulation key $K = \mathcal{H}'(\mathbf{m}')$ where $\mathcal{H}'$ is the hash function given in param.

Correctness: While decoding $\mathbf{c}$, we form an $st \times n$ parity check matrix H' over $\mathsf{GF}(q^m)$ using the secret key $\mathsf{sk} = (\mathbf{v}, \mathbf{y})$ and find the syndrome $H'(\mathbf{c}||\mathbf{0})^T$ to estimate the error vector $\mathbf{e}'' \in (\mathsf{GF}(q))^n$ with $\mathsf{wt}(\mathbf{e}'') = w$. Note that, the ciphertext component $\mathbf{c} = H(\mathbf{e}')^T$ is the syndrome of $\mathbf{e}'$ where the matrix H is a parity check matrix in the systemetic form over $\mathsf{GF}(q)$ which is indistinguishable from a random matrix over $\mathsf{GF}(q)$. At the time of decoding $\mathbf{c}$, we need a parity check matrix in alternant form over $\mathsf{GF}(q^m)$. The parity check matrix H, a parity check matrix of GS code in the systemetic form derived from the public key pk, does not help to decode $\mathbf{c}$ as the SD problem is hard over $\mathsf{GF}(q)$. The decoding algorithm in our decapsulation procedure uses the parity check matrix H' (derived from the secret key sk) which is in alternant form over $\mathsf{GF}(q^m)$. This procedure can correct upto $st/2$ errors. In our scheme, the error vector $\mathbf{e}'$

used in the procedure $\mathsf{KEM.Encaps}$ satisfies $\mathsf{wt}(\mathbf{e}') = w \leq st/2$. Consequently, the decoding procedure will recover the correct $\mathbf{e}'$. We regenerate $\mathbf{e}_0'$ and $\boldsymbol{\rho}''$ and compare it with $\mathbf{e}_0$ and $\boldsymbol{\rho}'$ obtained after decoding. Since the error vector generation uses a deterministic function to get a fixed low weight error vector, $\mathbf{e}_0 = \mathbf{e}_0'$ and $\boldsymbol{\rho}' = \boldsymbol{\rho}''$ occurs.

4 Security

4.1 Security Notions

Definition 5 (Indistinguishability under Chosen Plaintext Attack (IND-CPA) [13])**.** The IND-CPA game between a challenger $\mathcal{S}$ and a PPT adversary $\mathcal{A}$ for a public key encryption scheme PKE=(Setup, KeyGen, Enc, Dec) is described below.

1. The challenger $\mathcal{S}$ generates $\mathsf{param} \longleftarrow \mathsf{PKE.Setup}(1^\lambda)$, $(\mathsf{pk}, \mathsf{sk}) \longleftarrow \mathsf{PKE.Key\text{-}Gen}(\mathsf{param})$ where λ is a security parameter and sends $\mathsf{param}, \mathsf{pk}$ to $\mathcal{A}$.
2. The adversary $\mathcal{A}$ sends a pair of messages $\mathbf{m}_0, \mathbf{m}_1 \in \mathcal{M}$ of the same length to $\mathcal{S}$.
3. The challenger $\mathcal{S}$ picks a random bit $b \in \{0, 1\}$, computes a challenge ciphertext $\mathsf{CT} \longleftarrow \mathsf{PKE.Enc}(\mathsf{param}, \mathsf{pk}, \mathbf{m}_b; \mathbf{r}_b)$ and sends it to $\mathcal{A}$.
4. The adversary outputs a bit b'.

The adversary $\mathcal{A}$ wins the game if $b' = b$. We define the advantage of $\mathcal{A}$ against the above IND-CPA security game for the PKE scheme as

$$\mathsf{Adv}_{\mathsf{PKE}}^{\mathsf{IND\text{-}CPA}}(\mathcal{A}) = |\Pr[b' = b] - 1/2|.$$

A PKE scheme is IND-CPA secure if $\mathsf{Adv}_{\mathsf{PKE}}^{\mathsf{IND\text{-}CPA}}(\mathcal{A}) < \epsilon$ where $\epsilon > 0$ is arbitrarily small.

We also define the following four security notions for PKE scheme that are (i) One-Wayness under Chosen Plaintext Attacks (OW-CPA), (ii) One-Wayness under Plaintext Checking Attacks (OW-PCA), (iii) One-Wayness under Validity Checking Attacks (OW-VA) and (iv) One-Wayness under Plaintext and Validity Checking Attacks (OW-PCVA).

Definition 6 (OW-ATK [15])**.** For $\mathsf{ATK} \in \{\mathsf{CPA}, \mathsf{PCA}, \mathsf{VA}, \mathsf{PCVA}\}$, the OW-ATK game between a challenger $\mathcal{S}$ and a PPT adversary $\mathcal{A}$ for a public key encryption scheme PKE = (Setup, KeyGen, Enc, Dec) is outlined below where $\mathcal{A}$ can make polynomially many queries to the oracle O_{ATK} given by

$$O_{\mathsf{ATK}} = \begin{cases} - & \mathsf{ATK} = \mathsf{CPA} \\ \mathsf{PCO}(\cdot,\cdot) & \mathsf{ATK} = \mathsf{PCA} \\ \mathsf{CVO}(\cdot) & \mathsf{ATK} = \mathsf{VA} \\ \mathsf{PCO}(\cdot,\cdot), \mathsf{CVO}(\cdot) & \mathsf{ATK} = \mathsf{PCVA} \end{cases}$$

where the oracle $\mathsf{PCO}(\cdot,\cdot)$ takes a message $\mathbf{m}$ and a ciphertext CT as input and checks if the message recovered from CT is $\mathbf{m}$ or not while the oracle $\mathsf{CVO}(\cdot)$

takes a ciphertext CT as input distinct from the challenge ciphertext CT^* and checks whether the message recovered from CT belongs to the message space or not.

1. The challenger $\mathcal{S}$ generates $\mathsf{param} \longleftarrow \mathsf{PKE.Setup}(1^\lambda)$, $(\mathsf{pk}, \mathsf{sk}) \longleftarrow \mathsf{PKE.Key\text{-}Gen}(\mathsf{param})$ where λ is a security parameter and sends $\mathsf{param}, \mathsf{pk}$ to $\mathcal{A}$.
2. The challenger $\mathcal{S}$ chooses a message $\mathbf{m}^* \in \mathcal{M}$, computes the challenge ciphertext $\mathsf{CT}^* \longleftarrow \mathsf{PKE.Enc}(\mathsf{param}, \mathsf{pk}, \mathbf{m}^*; \mathbf{r}^*)$ and sends it to $\mathcal{A}$.
3. The adversary $\mathcal{A}$ having access to the oracle O_{ATK}, outputs $\mathbf{m}'$.

The adversary $\mathcal{A}$ wins the game if $\mathbf{m}' = \mathbf{m}^*$. We define the advantage of $\mathcal{A}$ against the above OW-ATK security game for PKE scheme as $\mathsf{Adv}_{\mathsf{PKE}}^{\mathsf{OW\text{-}ATK}}(\mathcal{A}) = \Pr[\mathbf{m}' = \mathbf{m}^*]$. The PKE scheme is said to be OW-ATK secure if $\mathsf{Adv}_{\mathsf{PKE}}^{\mathsf{OW\text{-}ATK}}(\mathcal{A}) < \epsilon$ for arbitrarily small non zero ϵ.

Definition 7 (Indistinguishability under Chosen Ciphertext Attack (IND-CCA) [19]). The IND-CCA game between a challenger $\mathcal{S}$ and a PPT adversary $\mathcal{A}$ for a key encapsulation mechanism $\mathsf{KEM} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps})$ is described below.

1. The challenger $\mathcal{S}$ generates $\mathsf{param} \longleftarrow \mathsf{KEM.Setup}(1^\lambda)$ and $(\mathsf{pk}, \mathsf{sk}) \longleftarrow \mathsf{KEM.KeyGen}(\mathsf{param})$ where λ is a security parameter and sends $\mathsf{param}, \mathsf{pk}$ to $\mathcal{A}$.
2. The PPT adversary $\mathcal{A}$ has access to the decapsulation oracle $\mathsf{KEM.Decaps}$ to which $\mathcal{A}$ can make polynomially many ciphertext queries CT_i and gets the corresponding key $K_i \in \mathcal{K}$ from $\mathcal{S}$.
3. The challenger $\mathcal{S}$ picks a random bit b from $\{0, 1\}$, runs $\mathsf{KEM.Encaps}(\mathsf{param}, \mathsf{pk})$ to generate a ciphertext-key pair (CT^*, K_0^*) with $\mathsf{CT}^* \neq \mathsf{CT}_i$, selects randomly $K_1^* \in \mathcal{K}$ and sends the pair (CT^*, K_b^*) to $\mathcal{A}$.
4. The adversary $\mathcal{A}$ having the pair (CT^*, K_b^*) keeps performing polynomially many decapsulation queries on $\mathsf{CT}_i \neq \mathsf{CT}^*$ and outputs b'.

The adversary succeeds the game if $b' = b$. We define the advantage of $\mathcal{A}$ against the above IND-CCA security game for the KEM as

$$\mathsf{Adv}_{\mathsf{KEM}}^{\mathsf{IND\text{-}CCA}}(\mathcal{A}) = |\Pr[b' = b] - 1/2|.$$

A KEM is IND-CCA secure if $\mathsf{Adv}_{\mathsf{KEM}}^{\mathsf{IND\text{-}CCA}}(\mathcal{A}) < \epsilon$ where $\epsilon > 0$ is arbitrarily small.

4.2 Security Proof

Our KEM provides IND-CCA security in random oracle model by Theorem 1.

Theorem 1. *Assuming the hardness of decisional* SD *problem (Definition 1, Sect. 2.1) and indistinguishability of the public key matrix H (derived from the public key* pk *by running* $\mathsf{KEM.KeyGen}(\mathsf{param})$ *where* $\mathsf{param} \longleftarrow \mathsf{KEM.}\mathit{Setup}(1^\lambda)$, *$\lambda$ being the security parameter), our* $\mathsf{KEM} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps})$ *described in Sect. 3 provides* IND-CCA *security (Definition 7, Sect. 4.1) when the hash functions $\mathcal{H}'$ and $\mathcal{G}$ are modeled as random oracles.*

- $\mathsf{PKE}_1.\mathsf{Setup}(1^\lambda) \longrightarrow \mathsf{param}$: A trusted authority runs $\mathsf{KEM.Setup}(1^\lambda)$ to get global parameters $\mathsf{param} = (n, n_0, k, k', w, q, s, t, r, m, \mathcal{G}, \mathcal{H}, \mathcal{H}')$ taking security parameter λ as input.

- $\mathsf{PKE}_1.\mathsf{KeyGen}(\mathsf{param}) \longrightarrow (\mathsf{pk}, \mathsf{sk})$: A user generates public-secret key pair $(\mathsf{pk}, \mathsf{sk})$ by running $\mathsf{KEM.KeyGen}(\mathsf{param})$ where $\mathsf{pk} = \{\boldsymbol{\psi}_{i,j} | i = 0, 1, \ldots, mt-1,\ j = 0, 1, \ldots, \frac{k}{s}-1\}$, $\boldsymbol{\psi}_{i,j} \in (\mathsf{GF}(q))^s$ and $\mathsf{sk} = (\mathbf{v}, \mathbf{y})$.

- $\mathsf{PKE}_1.\mathsf{Enc}(\mathsf{param}, \mathsf{pk}, \mathbf{m}; \mathbf{r}) \longrightarrow \mathsf{CT}$: An encryptor encrypts a message $\mathbf{m} \in \mathcal{M} = (\mathsf{GF}(q))^{k'}$ and produces a ciphertext CT as follows.
  1. Compute $\mathbf{r} = \mathcal{G}(\mathbf{m}) \in (\mathsf{GF}(q))^k$, $\mathbf{d} = \mathcal{H}(\mathbf{m}) \in (\mathsf{GF}(q))^{k'}$.
  2. Parse $\mathbf{r} = (\boldsymbol{\rho}||\boldsymbol{\sigma})$ where $\boldsymbol{\rho} \in (\mathsf{GF}(q))^{k-k'}$, $\boldsymbol{\sigma} \in (\mathsf{GF}(q))^{k'}$. Set $\boldsymbol{\mu} = (\boldsymbol{\rho}||\mathbf{m}) \in (\mathsf{GF}(q))^k$.
  3. Run Algorithm 2 using $\boldsymbol{\sigma}$ as a seed to obtain an error vector $\mathbf{e}$ of length $n-k$ and weight $w - \mathsf{wt}(\boldsymbol{\mu})$ and set $\mathbf{e}' = (\mathbf{e}||\boldsymbol{\mu}) \in (\mathsf{GF}(q))^n$.
  4. Use the public key $\mathsf{pk} = \{\boldsymbol{\psi}_{i,j} | i = 0, 1, \ldots, mt-1,\ j = 0, 1, \ldots, \frac{k}{s} - 1\}$ as in $\mathsf{KEM.Encaps}(\mathsf{param},\mathsf{pk})$ and construct the matrix $H_{(n-k)\times n} = (M|I_{n-k})$ where $M = (M_{i,j})$, $M_{i,j}$ is a $s \times s$ dyadic matrix with signature $\boldsymbol{\psi}_{i,j}$, $i = 0, 1, \ldots, mt-1$, $j = 0, 1, \ldots, \frac{k}{s} - 1$.
  5. Compute $\mathbf{c} = H(\mathbf{e}')^T$. Return the ciphertext $\mathsf{CT} = (\mathbf{c},\ \mathbf{d}) \in \mathcal{C} = (\mathsf{GF}(q))^{n-k+k'}$.

- $\mathsf{PKE}_1.\mathsf{Dec}(\mathsf{param}, \mathsf{sk}, \mathsf{CT}) \longrightarrow \mathbf{m}'$: On receiving the ciphertext CT, the decryptor executes the following steps using public parameters param and its secret key $\mathsf{sk} = (\mathbf{v}, \mathbf{y})$.
  1. Use the secret key $\mathsf{sk} = (\mathbf{v}, \mathbf{y})$ to form a parity check matrix H' as in the procedure $\mathsf{KEM.Decaps}(\mathsf{param},\mathsf{sk},\mathsf{CT})$.
  2. To decode $\mathbf{c}$ (extracted from CT), find error $\mathbf{e}''$ of weight w and length n by running the decoding algorithm for Alternant codes with syndrome $H'(\mathbf{c}||\mathbf{0})^T$.
  3. Parse $\mathbf{e}'' = (\mathbf{e}_0||\boldsymbol{\mu}') \in (\mathsf{GF}(q))^n$ and $\boldsymbol{\mu}'=(\boldsymbol{\rho}'||\mathbf{m}') \in (\mathsf{GF}(q))^k$ where $\mathbf{e_0} \in (\mathsf{GF}(q))^{n-k}$, $\boldsymbol{\rho}' \in (\mathsf{GF}(q))^{k-k'}$, $\mathbf{m}' \in (\mathsf{GF}(q))^{k'}$.
     Compute $\mathbf{r}' = \mathcal{G}(\mathbf{m}') \in (\mathsf{GF}(q))^k$ and $\mathbf{d}' = \mathcal{H}(\mathbf{m}') \in (\mathsf{GF}(q))^{k'}$.
  4. Parse $\mathbf{r}' = (\boldsymbol{\rho}''||\boldsymbol{\sigma}')$ where $\boldsymbol{\rho}'' \in (\mathsf{GF}(q))^{k-k'}$, $\boldsymbol{\sigma}' \in (\mathsf{GF}(q))^{k'}$.
  5. Generate error vector $\mathbf{e}_0'$ of length $n-k$ and weight $w - \mathsf{wt}(\boldsymbol{\mu}')$ by running Algorithm 2 with $\boldsymbol{\sigma}'$ as seed.
  6. If $(\mathbf{e}_0 \neq \mathbf{e}_0') \vee (\boldsymbol{\rho}' \neq \boldsymbol{\rho}'') \vee (\mathbf{d} \neq \mathbf{d}')$, output $\perp$ indicating decryption failure. Otherwise, return $\mathbf{m}'$.

Fig. 1. Scheme $\mathsf{PKE}_1 = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$

The proof of the above theorem is the immediate consequence of Theorem 2, Corollary 1 and Theorem 4.

Theorem 2. *If the public key encryption scheme* $\mathsf{PKE}_1 = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *described in Fig. 1 is* OW-VA *secure (Definition 6, Sect. 4.1) and there exist cryptographically secure hash functions, then the key encapsulation mechanism* $\mathsf{KEM} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps})$ *as described in Sect. 3 achieves* IND-CCA *security (Definition 7, Sect. 4.1) when the hash function* $\mathcal{H}'$ *is modeled as a random oracle.*

Proof. Let $\mathcal{B}$ be a PPT adversary against the IND-CCA security of KEM providing at most n_D queries to KEM.Decaps oracle and at most $n_{\mathcal{H}'}$ queries to the hash oracle $\mathcal{H}'$. We show that $\exists$ a PPT adversary $\mathcal{A}$ against the OW-VA security of the scheme PKE_1. We start with a sequence of games and the view of the adversary $\mathcal{B}$ is shown to be computationally indistinguishable in any of the consecutive games. Finally, we end up in a game that statistically hides the challenge bit as required. All the games are defined in Figs. 2 and 3. Let E_j be the event that $b = b'$ in game $\mathbf{G}_j$, $j = 0, 1, 2, 3$.

- The challenger $\mathcal{S}$ generates $\mathsf{param} \longleftarrow \mathsf{KEM.Setup}(1^\lambda)$ and $(\mathsf{pk}, \mathsf{sk}) \longleftarrow \mathsf{KEM.KeyGen}(\mathsf{param})$ for a security parameter λ and sends $\mathsf{param}, \mathsf{pk}$ to $\mathcal{B}$.
- The PPT adversary $\mathcal{B}$ has access to the decapsulation oracle $\mathsf{KEM.Decaps}$ to which $\mathcal{B}$ can make polynomially many ciphertext queries CT_i and gets the corresponding key $K_i \in \mathcal{K} = \{0,1\}^r$ from $\mathcal{S}$.
- The challenger $\mathcal{S}$ picks a random bit b from $\{0,1\}$, runs $\mathsf{KEM.Encaps}(\mathsf{param}, \mathsf{pk})$ to generate a ciphertext-key pair (CT^*, K_0^*) with $\mathsf{CT}^* \neq \mathsf{CT}_i$, selects randomly $K_1^* \in \mathcal{K}$ and sends the pair (CT^*, K_b^*) to $\mathcal{B}$.
- The adversary $\mathcal{B}$ having the pair (CT^*, K_b^*) keeps performing polynomially many decapsulation queries on $\mathsf{CT}_i \neq \mathsf{CT}^*$ and outputs b'.

Fig. 2. Game $\mathbf{G}_0$ in the proof of Theorem 2

- The challenger $\mathcal{S}$ generates $\mathsf{param} \longleftarrow \mathsf{PKE}_1.\mathsf{Setup}(1^\lambda)$, $(\mathsf{pk}, \mathsf{sk}) \longleftarrow \mathsf{PKE}_1.\mathsf{KeyGen}(\mathsf{param})$ for a security parameter λ and sends $\mathsf{param}, \mathsf{pk}$ to $\mathcal{B}$.
- The PPT adversary $\mathcal{B}$ has access to the decapsulation oracle Decaps (see Figure 4) to which $\mathcal{B}$ can make polynomially many ciphertext queries CT_i and gets the corresponding key $K_i \in \mathcal{K}$ from $\mathcal{S}$.
- The challenger $\mathcal{S}$ picks a random bit b from $\{0,1\}$, chooses a message $\mathbf{m}^* \xleftarrow{U} \mathcal{M}$, runs $\mathsf{PKE}_1.\mathsf{Enc}(\mathsf{param}, \mathsf{pk}, \mathbf{m}^*; \mathbf{r}^*)$ to generate a ciphertext CT^*, computes $K_0^* = \mathcal{H}'(\mathbf{m}^*)$, selects randomly $K_1^* \in \mathcal{K}$ and sends the pair (CT^*, K_b^*) to $\mathcal{B}$.
- The adversary $\mathcal{B}$ having the pair (CT^*, K_b^*) keeps performing polynomially many decapsulation queries on $\mathsf{CT}_i \neq \mathsf{CT}^*$ to Decaps oracle and hash queries on $\mathbf{m}_i$ to hash oracle $\mathcal{H}'$and outputs b' (see Figure 4 for hash oracle $\mathcal{H}'$ and decapsulation oracle Decaps).

Fig. 3. Sequence of games $\mathbf{G}_j, j = 1, 2, 3$ in the proof of Theorem 2

Game $\mathbf{G}_0$: As usual, game $\mathbf{G}_0$ (Fig. 2) is the standard IND-CCA security game (Definition 7, Sect. 4.1) for the KEM and we have $|\Pr[E_0] - 1/2| = \mathsf{Adv}_{\mathsf{KEM}}^{\mathsf{IND\text{-}CCA}}(\mathcal{B})$.
Game $\mathbf{G}_1$: In game $\mathbf{G}_1$, a message $\mathbf{m}^*$ is chosen randomly and the ciphertext CT^* is computed by running $\mathsf{PKE}_1.\mathsf{Enc}(\mathsf{param}, \mathsf{pk}, \mathbf{m}^*; \mathbf{r}^*)$. The challenger $\mathcal{S}$ maintains a hash list $Q_{\mathcal{H}'}$ (initially empty) and records all entries of the form $(\mathbf{m}, K)$ where hash oracle $\mathcal{H}'$ is queried on some message $\mathbf{m} \in \mathcal{M}$. Note that both games $\mathbf{G}_0$ and $\mathbf{G}_1$ proceed identically and we get $\Pr[E_0] = \Pr[E_1]$.
Game $\mathbf{G}_2$: In game $\mathbf{G}_2$, the hash oracle $\mathcal{H}'$ and the decapsulation oracle Decaps are answered in such a way that they no longer make use of the secret key sk except for testing whether $\mathsf{PKE}_1.\mathsf{Dec}(\mathsf{param}, \mathsf{sk}, \mathsf{CT}) \in \mathcal{M}$ for a given ciphertext CT (line 12 of Decaps oracle in Fig. 4). The hash list $Q_{\mathcal{H}'}$ records all entries of the form $(\mathbf{m}, K)$ where hash oracle $\mathcal{H}'$ is queried on some message $\mathbf{m} \in \mathcal{M}$. Another list Q_D stores entries of the form (CT, K) where either Decaps oracle is queried on some ciphertext CT or the hash oracle $\mathcal{H}'$ is queried on some message $\mathbf{m} \in \mathcal{M}$ satisfying $\mathsf{CT} \longleftarrow \mathsf{PKE}_1.\mathsf{Enc}(\mathsf{param}, \mathsf{pk}, \mathbf{m}; \mathbf{r})$ with $\mathsf{PKE}_1.\mathsf{Dec}(\mathsf{param}, \mathsf{sk}, \mathsf{CT}) \longrightarrow \mathbf{m}$.

Let X denotes the event that a correctness error has occurred in the underlying PKE_1 scheme. More specifically, X is the event that either the list $Q_{\mathcal{H}'}$ contains an entry $(\mathbf{m}, K)$ with the condition $\mathsf{PKE}_1.\mathsf{Dec}(\mathsf{param}, \mathsf{sk}, \mathsf{PKE}_1.\mathsf{Enc}(\mathsf{param}, \mathsf{pk}, \mathbf{m}; \mathbf{r})) \neq \mathbf{m}$ or the list Q_D contains an entry (CT, K) with the condition $\mathsf{PKE}_1.\mathsf{Enc}(\mathsf{param}, \mathsf{pk}, \mathsf{PKE}_1.\mathsf{Dec}(\mathsf{param}, \mathsf{sk}, \mathsf{CT}); \mathbf{r}) \neq \mathsf{CT}$ or both.
Claim: The view of $\mathcal{B}$ is identical in games $\mathbf{G}_1$ and $\mathbf{G}_2$ unless the event X occurs.

$\underline{\mathcal{H}'(\mathbf{m})}$

1. **for** the game $\mathbf{G}_1, \mathbf{G}_2, \mathbf{G}_3$ **do**
2. **if** $\exists K$ such that $(\mathbf{m}, K) \in Q_{\mathcal{H}'}$
3. **return** K;
4. **end if**
5. $\mathsf{CT} = (\mathbf{c}, \mathbf{d}) \longleftarrow \mathsf{PKE}_1.\mathsf{Enc}(\mathsf{param}, \mathsf{pk}, \mathbf{m}; \mathbf{r})$;
6. $K \xleftarrow{U} \mathcal{K}$;
7. **end for**
8. **for** the game $\mathbf{G}_3$ **do**
9. **if** $\mathbf{m} = \mathbf{m}^*$ and CT^* defined
10. $Y = \mathsf{true}$;
11. **abort**;
12. **end if**
13. **end for**
14. **for** the game $\mathbf{G}_2, \mathbf{G}_3$ **do**
15. **if** $\exists K'$ such that $(\mathsf{CT}, K') \in Q_D$
16. $K = K'$;
17. **else**
18. $Q_D = Q_D \cup \{(\mathsf{CT}, K)\}$;
19. **end if**
20. **end for**
21. **for** the game $\mathbf{G}_1, \mathbf{G}_2, \mathbf{G}_3$ **do**
22. $Q_{\mathcal{H}'} = Q_{\mathcal{H}'} \cup \{(\mathbf{m}, K)\}$;
23. **return** K;
24. **end for**

$\underline{\mathsf{Decaps}(\mathsf{CT} \neq \mathsf{CT}^*)}$

1. **for** game $\mathbf{G}_1$ **do**
2. $\mathbf{m}' \longleftarrow \mathsf{PKE}_1.\mathsf{Dec}(\mathsf{param}, \mathsf{sk}, \mathsf{CT})$;
3. **if** $\mathbf{m}' = \perp$
4. **return** $\perp$;
5. **end if**
6. **return** $K = \mathcal{H}'(\mathbf{m}')$;
7. **end for**
8. **for** games $\mathbf{G}_2, \mathbf{G}_3$ **do**
9. **if** $\exists K$ such that $(\mathsf{CT}, K) \in Q_D$
10. **return** K;
11. **end if**
12. **if** $\mathsf{PKE}_1.\mathsf{Dec}(\mathsf{param}, \mathsf{sk}, \mathsf{CT}) \notin \mathcal{M}$
13. **return** $\perp$;
14. **end if**
15. $K \xleftarrow{U} \mathcal{K}$;
16. $Q_D = Q_D \cup \{(\mathsf{CT}, K)\}$;
17. **return** K;
18. **end for**

Fig. 4. The hash oracles $\mathcal{H}'$ and the decapsulation oracle Decaps for games $\mathbf{G}_j, j = 1, 2, 3$ in the proof of Theorem 2

$\underline{\mathcal{A}^{\mathsf{CVO}(\cdot)}(\mathsf{param}, \mathsf{pk}, \mathsf{CT}^*)}$

1. $K^* \xleftarrow{U} \mathcal{K}$;
2. $b' \longleftarrow \mathcal{B}^{\mathsf{Decaps}(\cdot), \mathcal{H}'(\cdot)}(\mathsf{param}, \mathsf{pk}, \mathsf{CT}^*, K^*)$;
3. **if** $\exists (\mathbf{m}', K') \in Q_{\mathcal{H}'}$ such that $\mathsf{PKE}_1.\mathsf{Enc}(\mathsf{param}, \mathsf{pk}, \mathbf{m}'; \mathbf{r}) \longrightarrow \mathsf{CT}^*$
4. **return** $\mathbf{m}'$;
5. **else**
6. **abort**;
7. **end if**

$\underline{\mathsf{Decaps}(\mathsf{CT} \neq \mathsf{CT}^*)}$

1. **if** $\exists K$ such that $(\mathsf{CT}, K) \in Q_D$
2. **return** K;
3. **end if**
4. **if** $\mathsf{CVO}(\mathsf{CT}) = 0$
5. **return** $\perp$;
6. **end if**
7. $K \xleftarrow{U} \mathcal{K}$;
8. $Q_D = Q_D \cup \{(\mathsf{CT}, K)\}$;
9. **return** K;

Fig. 5. Adversary $\mathcal{A}$ against OW-VA security of PKE_1

Proof of claim. To prove this, consider a fixed PKE_1 ciphertext CT (placed as a Decaps query) with $\mathbf{m} \longleftarrow \mathsf{PKE}_1.\mathsf{Dec}(\mathsf{param}, \mathsf{sk}, \mathsf{CT})$. Note that when $\mathbf{m} \notin \mathcal{M}$, the decapsulation oracle $\mathsf{Decaps}(\mathsf{CT})$ returns $\perp$ in both games $\mathbf{G}_1$ and $\mathbf{G}_2$. Suppose $\mathbf{m} \in \mathcal{M}$. We now show that in game $\mathbf{G}_2$, $\mathsf{Decaps}(\mathsf{CT}) \longrightarrow \mathcal{H}'(\mathbf{m})$ for the PKE_1 ciphertext CT of a message $\mathbf{m} \in \mathcal{M}$ with $\mathsf{PKE}_1.\mathsf{Enc}(\mathsf{param}, \mathsf{pk}, \mathbf{m}; \mathbf{r}) \longrightarrow \mathsf{CT}$. We distinguish two cases – $\mathcal{B}$ queries hash oracle $\mathcal{H}'$ on $\mathbf{m}$ before making Decaps oracle on CT, or the other way round.

Case 1: Let the oracle $\mathcal{H}'$ be queried on $\mathbf{m}$ first by $\mathcal{B}$ before decapsulation query on PKE_1 ciphertext CT. Since Decaps oracle was not yet queried on CT, no entry of the form (CT, K) exist in the current list Q_D yet. Therefore, besides adding $(\mathbf{m}, K \xleftarrow{U} \mathcal{K})$ to the list $Q_{\mathcal{H}'}$(line 22 of $\mathcal{H}'$ oracle in Fig. 4), the challenger $\mathcal{S}$

also adds (CT, K) to the list Q_D (line 18 of $\mathcal{H}'$ oracle in Fig. 4), thereby defining $\mathsf{Decaps}(\mathsf{CT}) \longrightarrow K = \mathcal{H}'(\mathbf{m})$.
Case 2: Let the oracle Decaps be queried on PKE_1 ciphertext CT before the hash oracle $\mathcal{H}'$ is queried on $\mathbf{m}$. Then no entry of the form (CT, K) exists in Q_D yet. Otherwise, $\mathcal{H}'$ already was queried on a message $\mathbf{m}'' \neq \mathbf{m}$ (because Decaps oracle is assumed to be queried first on CT and the oracle $\mathcal{H}'$ was not yet queried on $\mathbf{m}$) satisfying $\mathsf{PKE}_1.\mathsf{Enc}(\mathsf{param}, \mathsf{pk}, \mathbf{m}''; \mathbf{r}'') \longrightarrow \mathsf{CT}$ with $\mathsf{PKE}_1.\mathsf{Dec}(\mathsf{param}, \mathsf{sk}, \mathsf{CT}) \longrightarrow \mathbf{m}''$. This is a contradiction to the fact that the same PKE_1 ciphertext CT is generated for two different messages $\mathbf{m}'', \mathbf{m}$ using randomness $\mathbf{r}, \mathbf{r}''$ respectively where $\mathbf{r} = \mathcal{G}(\mathbf{m}) \neq \mathcal{G}(\mathbf{m}'') = \mathbf{r}''$ for a cryptographically secure hash function $\mathcal{G}$. Therefore,Decaps oracle adds $(\mathsf{CT}, K \xleftarrow{U} \mathcal{K})$ to the list Q_D, thereby defining $\mathsf{Decaps}(\mathsf{CT}) \longrightarrow K$. When queried on $\mathbf{m}$ afterwards for hash oracle $\mathcal{H}'$, an entry of the form (CT, K) already exists in the list Q_D (line 15 of $\mathcal{H}'$ oracle in Fig. 4). By adding $(\mathbf{m}, K)$ to the list $Q_{\mathcal{H}'}$ and returning K, the hash oracle $\mathcal{H}'$ defines $\mathcal{H}'(\mathbf{m}) = K \longleftarrow \mathsf{Decaps}(\mathsf{CT})$.

Hence, $\mathcal{B}$'s view is identical in games $\mathbf{G}_1$ and $\mathbf{G}_2$ unless a correctness error X occurs. □ (of Claim)

As $\Pr[X] = 0$ for our KEM, we have $\Pr[E_1] = \Pr[E_2]$.
Game $\mathbf{G}_3$: In game $\mathbf{G}_3$, the challenger $\mathcal{S}$ sets a flag $Y = \mathsf{true}$ and aborts (with uniformly random output) immediately on the event when $\mathcal{B}$ queries the hash oracle $\mathcal{H}'$ on $\mathbf{m}^*$. Hence, $|\Pr[E_2] - \Pr[E_3]| \leq \Pr[Y = \mathsf{true}]$. In game $\mathbf{G}_3$, $\mathcal{H}'(\mathbf{m}^*)$ will never be given to $\mathcal{B}$ neither through a query on hash oracle $\mathcal{H}'$ nor through a query on decapsulation oracle Decaps, meaning bit b is independent from $\mathcal{B}$'s view. Thus, $\Pr[E_3] = 1/2$. To bound $\Pr[Y = \mathsf{true}]$, we construct an adversary $\mathcal{A}$ against the OW-VA security of PKE_1 simulating game $\mathbf{G}_3$ for $\mathcal{B}$ as in Fig. 5. Here $\mathcal{B}$ uses Decaps oracle given in Fig. 5 with the same hash oracle $\mathcal{H}'$ for game $\mathbf{G}_2$ in Fig. 4. Consequently, the simulation is perfect until $Y = \mathsf{true}$ occurs. Furthermore, $Y = \mathsf{true}$ ensures that $\mathcal{B}$ has queried $\mathcal{H}'(\mathbf{m}^*)$, which implies that $(\mathbf{m}^*, K') \in Q_{\mathcal{H}'}$ for some $K' \in \mathcal{K}$ where the list $Q_{\mathcal{H}'}$ is maintained by the adversary $\mathcal{A}$ simulating G_3 for $\mathcal{B}$. In this case, we have $\mathsf{PKE}_1.\mathsf{Enc}(\mathsf{param}, \mathsf{pk}, \mathbf{m}^*; \mathbf{r}^*) \longrightarrow \mathsf{CT}^*$ and hence $\mathcal{A}$ returns $\mathbf{m}^*$. Thus, $\Pr[Y = \mathsf{true}] = \mathsf{Adv}_{\mathsf{PKE}_1}^{\mathsf{OW\text{-}VA}}(\mathcal{A})$. Combining all the probabilities, we get $\mathsf{Adv}_{\mathsf{KEM}}^{\mathsf{IND\text{-}CCA}}(\mathcal{B}) = |\Pr[E_0] - 1/2| = |\Pr[E_1] - 1/2| = |\Pr[E_2] - 1/2| = |\Pr[E_2] - \Pr[E_3]| \leq \Pr[Y = \text{true}] = \mathsf{Adv}_{\mathsf{PKE}_1}^{\mathsf{OW\text{-}VA}}(\mathcal{A})$ which completes our proof. □

Theorem 3. *If the public key encryption scheme* $\mathsf{PKE}_2 = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *described in Fig. 6 is* IND-CPA *secure (Definition 5, Sect. 4.1), then the public key encryption scheme* $\mathsf{PKE}_1 = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *as described in Fig. 1 provides* OW-PCVA *security (Definition 6, Sect. 4.1) when the hash function* $\mathcal{G}$ *is modeled as a random oracle.*

- $\mathsf{PKE}_2.\mathsf{Setup}(1^\lambda) \longrightarrow \mathsf{param}$: A trusted authority takes security parameter λ as input and runs $\mathsf{PKE}_1.\mathsf{Setup}(1^\lambda)$ to get public parameters $\mathsf{param} = (n, n_0, k, k', w, q, s, t, r, m, \mathcal{G}, \mathcal{H}, \mathcal{H}')$.
- $\mathsf{PKE}_2.\mathsf{KeyGen}(\mathsf{param}) \longrightarrow (\mathsf{pk}, \mathsf{sk})$: A user generates the key pair $(\mathsf{pk}, \mathsf{sk})$ by running $\mathsf{PKE}_1.\mathsf{KeyGen}(\mathsf{param})$ where the public key $\mathsf{pk} = \{\boldsymbol{\psi}_{i,j} \mid i = 0, 1, \ldots, mt-1,\ j = 0, 1, \ldots, \frac{k}{s}-1\}$, $\boldsymbol{\psi}_{i,j} \in (\mathsf{GF}(q))^s$ and the secret key $\mathsf{sk} = (\mathbf{v}, \mathbf{y})$.
- $\mathsf{PKE}_2.\mathsf{Enc}(\mathsf{param}, \mathsf{pk}, \mathbf{m}; \mathbf{r}) \longrightarrow \mathbf{c}$: An encryptor encrypts a message $\mathbf{m} \in \mathcal{M} = (\mathsf{GF}(q))^{k'}$ and produces a ciphertext $\mathbf{c}$ as follows.
  1. Compute $\mathbf{r} = \mathcal{G}(\mathbf{m}) \in (\mathsf{GF}(q))^k$, $\mathbf{d} = \mathcal{H}(\mathbf{m}) \in (\mathsf{GF}(q))^{k'}$.
  2. Parse $\mathbf{r} = (\boldsymbol{\rho}||\boldsymbol{\sigma})$ where $\boldsymbol{\rho} \in (\mathsf{GF}(q))^{k-k'}$, $\boldsymbol{\sigma} \in (\mathsf{GF}(q))^{k'}$. Set $\boldsymbol{\mu} = (\boldsymbol{\rho}||\mathbf{m}) \in (\mathsf{GF}(q))^k$.
  3. Run Algorithm 2 using $\boldsymbol{\sigma}$ as a seed to obtain an error vector $\mathbf{e}$ of length $n-k$ and weight $w - \mathsf{wt}(\boldsymbol{\mu})$ and set $\mathbf{e}' = (\mathbf{e}||\boldsymbol{\mu}) \in (\mathsf{GF}(q))^n$.
  4. Use the public key $\mathsf{pk} = \{\boldsymbol{\psi}_{i,j} | i = 0, 1, \ldots, mt-1,\ j = 0, 1, \ldots, \frac{k}{s}-1\}$ as in $\mathsf{PKE}_1.\mathsf{Enc}(\mathsf{param}, \mathsf{pk}, \mathbf{m}; \mathbf{r})$ and construct the matrix $H_{(n-k)\times n} = (M|I_{n-k})$ where $M = (M_{i,j})$, $M_{i,j}$ is a $s \times s$ dyadic matrix with signature $\boldsymbol{\psi}_{i,j}$, $i = 0, 1, \ldots, mt-1$, $j = 0, 1, \ldots, \frac{k}{s}-1$. Compute $\mathbf{c} = H(\mathbf{e}')^T$.
- $\mathsf{PKE}_2.\mathsf{Dec}(\mathsf{param}, \mathsf{sk}, \mathbf{c}) \longrightarrow \mathbf{m}'$: On receiving the ciphertext $\mathbf{c}$, the decryptor performs the following steps using public parameters param and its secret key $\mathsf{sk} = (\mathbf{v}, \mathbf{y})$.
  1. Use the secret key $\mathsf{sk} = (\mathbf{v}, \mathbf{y})$ to form a parity check matrix H' as in the procedure $\mathsf{PKE}_1.\mathsf{Dec}(\mathsf{param}, \mathsf{sk}, \mathsf{CT})$
  2. To decode $\mathbf{c}$, find error $\mathbf{e}''$ of weight w and length n by running the decoding algorithm for Alternant codes with syndrome $H'(\mathbf{c}||\mathbf{0})^T$.
  3. Parse $\mathbf{e}'' = (\mathbf{e}_0||\boldsymbol{\mu}') \in (\mathsf{GF}(q))^n$ and $\boldsymbol{\mu}'=(\boldsymbol{\rho}'||\mathbf{m}') \in (\mathsf{GF}(q))^k$ where $\mathbf{e}_0 \in (\mathsf{GF}(q))^{n-k}$, $\boldsymbol{\rho}' \in (\mathsf{GF}(q))^{k-k'}$, $\mathbf{m}' \in (\mathsf{GF}(q))^{k'}$. Return $\mathbf{m}'$.

Fig. 6. Scheme $\mathsf{PKE}_2 = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$

The OW-PCVA security for a PKE scheme trivially implies the OW-VA security of the PKE scheme considering zero queries to the $\mathsf{PCO}(\cdot,\cdot)$ oracle. Therefore, the following corollary is an immediate consequence of Theorem 3.

Corollary 1. *If the public key encryption scheme* $\mathsf{PKE}_2 = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *described in Fig. 6 is* IND-CPA *secure (Definition 5, Sect. 4.1), then the public key encryption scheme* $\mathsf{PKE}_1 = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *as described in Fig. 1 provides* OW-VA *security (Definition 6, Sect. 4.1) when the hash function* $\mathcal{G}$ *is modeled as a random oracle.*

Theorem 4. *If the decisional* SD *problem (Definition 1, Sect. 2.1) is hard, the public key matrix H (derived from the public key* pk *which is generated by running* $\mathsf{PKE}_2.\mathsf{KeyGen}(\mathsf{param})$ *where* $\mathsf{param} \longleftarrow \mathsf{PKE}_2.\mathsf{Setup}(1^\lambda)$*) is indistinguishable (Definition 2, Sect. 2.1) and the hash function* $\mathcal{G}$ *is modeled as a random oracle, then the public key encryption scheme* $\mathsf{PKE}_2 = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *presented in Fig. 6 is* IND-CPA *secure (Definition 5, Sect. 4.1).*

Due to limited space, proofs of Theorems 3 and 4 will appear in the full version of the paper.

Remark 1. The KEM protocol also can be shown to provide security in quantum random oracle model following the work in [15] and thus we can get Theorem 5.

Theorem 5. *Assuming the hardness of decisional* SD *problem (Definition 1, Sect. 2.1) and indistinguishability of the public key matrix H (derived from the* *public key* pk *by running* $\mathsf{KEM.KeyGen(param)}$ *where* $\mathsf{param} \longleftarrow \mathsf{KEM.Setup}(1^\lambda)$*, λ being the security parameter), our* $\mathsf{KEM} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps})$ *described in Sect. 3 provides* IND-CCA *security (Definition 7, Sect. 4.1) when the hash functions $\mathcal{G}, \mathcal{H}$ and $\mathcal{H}'$ are modeled as quantum random oracles.*

Note that, proof of Theorem 5 follows from Theorems 4, 6 and 7 along with the fact that IND-CPA security implies OW-CPA security.

Theorem 6. *If the public key encryption scheme* $\mathsf{PKE}_2 = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *described in Fig. 6 is* OW-CPA *secure (Definition 6, Sect. 4.1), then the public key encryption scheme* $\mathsf{PKE}_1 = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *as described in Fig. 1 provides* OW-PCA *security (Definition 6, Sect. 4.1) when the hash function $\mathcal{G}$ is modeled as a quantum random oracle.*

Theorem 7. *If the public key encryption scheme* $\mathsf{PKE}_1 = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *described in Fig. 1 is* OW-PCA *secure (Definition 6, Sect. 4.1) and there exist cryptographically secure hash functions, then the key encapsulation mechanism* $\mathsf{KEM} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps})$ *as described in Sect. 3 achieves* IND-CCA *security (Definition 7, Sect. 4.1) when the hash functions $\mathcal{H}$ and $\mathcal{H}'$ are modeled as quantum random oracles.*

5 Conclusion

In this work, we give a proposal to design an IND-CCA secure key encapsulation mechanism based on Generalized Srivastava codes. In terms of storage, our work seems well as compared to some other code-based KEM protocols as shown in Table 1. The scheme instantiated with Generalized Srivastava code does not involve any correctness error like some lattice-based schemes which allows achieving a simpler and tighter security bound for the IND-CCA security. In the upcoming days, it would be desirable to devise more efficient and secure constructions using suitable error-correcting codes.

References

1. Aguilar-Melchor, C., Blazy, O., Deneuville, J.C., Gaborit, P., Zémor, G.: Efficient encryption from random quasi-cyclic codes. IEEE Trans. Inf. Theor. **64**(5), 3927–3943 (2018)
2. Albrecht, M., Cid, C., Paterson, K.G., Tjhai, C.J., Tomlinson, M.: NTS-KEM. NIST Submissions (2019)
3. Aragon, N., et al.: BIKE: bit flipping key encapsulation. NIST Submissions (2017)
4. Aragon, N., et al.: BIKE: bit flipping key encapsulation. NIST Submissions (2019)
5. Banegas, G., et al.: DAGS: key encapsulation using dyadic GS codes. J. Math. Cryptol. **12**(4), 221–239 (2018)
6. Bardet, M., et al.: Big quake. NIST Submissions (2017)
7. Barreto, P.S.L.M., Cayrel, P.-L., Misoczki, R., Niebuhr, R.: Quasi-dyadic CFS signatures. In: Lai, X., Yung, M., Lin, D. (eds.) Inscrypt 2010. LNCS, vol. 6584, pp. 336–349. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21518-6_23

8. Berlekamp, E., McEliece, R., Van Tilborg, H.: On the inherent intractability of certain coding problems (corresp.). IEEE Trans. Inf. Theor. **24**(3), 384–386 (1978)
9. Bernstein, D.J., et al.: Classic McEliece: conservative code-based cryptography. NIST Submissions (2017)
10. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G., Van Keer, R., Viguier, B.: KangarooTwelve: fast hashing based on KECCAK-p. In: Preneel, B., Vercauteren, F. (eds.) ACNS 2018. LNCS, vol. 10892, pp. 400–418. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-93387-0_21
11. Fabšič, T., Hromada, V., Stankovski, P., Zajac, P., Guo, Q., Johansson, T.: A reaction attack on the QC-LDPC McEliece cryptosystem. In: Lange, T., Takagi, T. (eds.) PQCrypto 2017. LNCS, vol. 10346, pp. 51–68. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59879-6_4
12. Faugere, J.C., Gauthier-Umana, V., Otmani, A., Perret, L., Tillich, J.P.: A distinguisher for high-rate McEliece cryptosystems. IEEE Trans. Inf. Theor. **59**(10), 6830–6844 (2013)
13. Goldwasser, S., Micali, S.: Probabilistic encryption. J. Comput. Syst. Sci. **28**(2), 270–299 (1984)
14. Guo, Q., Johansson, T., Stankovski, P.: A key recovery attack on MDPC with CCA security using decoding errors. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 789–815. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_29
15. Hofheinz, D., Hövelmanns, K., Kiltz, E.: A modular analysis of the Fujisaki-Okamoto transformation. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017. LNCS, vol. 10677, pp. 341–371. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70500-2_12
16. MacWilliams, F.J., Sloane, N.J.A.: The theory of error-correcting codes, vol. 16. Elsevier (1977)
17. McEliece, R.J.: A public-key cryptosystem based on algebraic. Coding Thv **4244**, 114–116 (1978)
18. Nojima, R., Imai, H., Kobara, K., Morozov, K.: Semantic security for the McEliece cryptosystem without random oracles. Des. Codes Crypt. **49**(1–3), 289–305 (2008)
19. Rackoff, C., Simon, D.R.: Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 433–444. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-46766-1_35
20. Sarwate, D.V.: On the complexity of decoding Goppa codes (corresp.). IEEE Trans. Inf. Theor. **23**(4), 515–516 (1977)
21. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings 35th Annual Symposium on Foundations of Computer Science, pp. 124–134. IEEE (1994)