Flexible Quasi-Dyadic Code-Based Public-Key Encryption and Signature

Kazukuni Kobara

kobara_conf@m.aist.go.jp

Abstract. Drawback of code-based public-key cryptosystems is that their public-key size is large. It takes some hundreds KB to some MB for typical parameters. While several attempts have been conducted to reduce it, most of them have failed except one, which is Quasi-Dyadic (QD) public-key (for large extension degrees). While an attack has been proposed on QD public-key (for small extension degrees), it can be prevented by making the extension degree \( m \) larger, specifically by making \( q^{\log_2(m+1)} \) large enough where \( q \) is the base field and for a binary code, \( q = 2 \). The drawback of QD is, however, it must hold \( n \ll 2^m - t \) (at least \( n \leq 2^m - 1 \)) where \( n \) and \( t \) are the code length and the error correction capability of the underlying code. If it is not satisfied, its key generation fails since it is performed by trial and error. This condition also prevents QD from generating parameters for code-based digital signatures since without making \( n \) close to \( 2^m - t \), \( 2^m t/n \) cannot be small. To overcome these problems, we propose “Flexible” Quasi-Dyadic (FQD) public-key that can even achieve \( n = 2^m - t \) with one shot. Advantages of FQD include 1) it can reduce the public-key size further, 2) it can be applied to code-based digital signatures, too.

Keywords public-key, digital signature, linear code, dyadic

1 Introduction

Public-key cryptosystems (PKCs) can be divided into the categories shown in Fig. 1 and 2, respectively. Almost all of the currently deployed ones are based only on a small class of hard problems, namely Integer Factoring Problem (IFP) or Discrete Logarithm Problem (DLP). They are referred to as number theoretic problems. The number theoretic problem based PKCs have the following disadvantages that should be solved in short term and long term, respectively. The long term problem is the lack of quantum tolerance. The number theoretic problems are closely related to a problem to determine the cycle (hence they may be referred to as a cyclic problem) and they will be solved in (probabilistic) polynomial-time after the emergence of quantum computers [24] though

\footnote{Multivariate polynomial based ones may be included, but all of them have been broken and no relief method is known so far.}
Integer Factoring Based:
- RSA
- Rabin
- Okamoto-Uchiyama
- Paillier

Discrete Logarithm Based:
- Diffie-Hellman
- ElGamal
- ECC
- XTR
- Cramer-Shoup
- Kurosawa-Desmedt

Code Based:
- McEliece
- Niederreiter

Lattice Based:
- NTRU
- Ajtai-Dwork
- Goldreich-Goldwasser-Halevi
- Ajtai
- Regev
- Peikert

Subset Sum Based:
- Okamoto-Tanaka-Uchiyama

Fig. 1. Examples of PKCs Based on Number Theoretic (Cyclic) Problem

Fig. 2. Examples of PKCs Based on Combinatorial Problem

several breakthroughs are needed to realize quantum computers. The short term problem is the requirement of heavy multiple precision modular exponentiations that are not easy to deploy with low cost on low-computational power devices, such as RFID (Radio Frequency Identity), sensors and SCADA (Supervisory Control And Data Acquisition) devices.

On the other hand, combinatorial-problems are quantum tolerant and only small arithmetic units, e.g. addition in a small field or ring, are required for encryption and signature verification. Furthermore, among the combinatorial-problem based PKCs, code-based PKCs are advantageous in redundancy, i.e. (Plaintext Size) – (Ciphertext Size), and in the arithmetic unit, i.e. encryption and signature verification consists mostly on exclusive-ors that are highly parallelizable. Hence, code-based PKCs are suitable for heterogeneous applications where one side may have a reasonable computational power, but that of the other side is limited such as privacy-preserving RFID [8] and lightweight broadcast authentication for emergency. Other than them, code-based primitives can be utilized to construct ZKIP (Zero Knowledge Interactive Proof) [26], hash functions [2], OT (Oblivious Transfer) [17, 11] and so on.

The strongest security notion of a PKC, IND-CCA2 (Indistinguishability against Adaptive Chosen Ciphertext Attack), can be achieved by applying “appropriate” conversion scheme to the primitive code-based PKEs as long as it satisfies OW-CPA (One-Wayness against Chosen Plaintext Attack). For the McEliece primitive PKC, specific conversion scheme [15] makes the redundancy smallest while maintaining provable security in the random oracle model. For the Niederreiter primitive PKC, either OAEP++ [14] for a long plaintext or OAEP+ [25] for a small plaintext can achieve them. Not only in the random oracle model, provable security of IND-CPA and IND-CCA2 have been achieved in the standard model in [23] and [10] respectively even though the constructions in the standard model are less efficient compared to those in the random oracle model. Anyway, secure constructions are available as long as the underlying primitive code-based PKCs satisfy OW-CPA and the parameters meeting OW-CPA are esti-
mated in [12] against the most powerful attacks (Optimized) Information Set Decoding (OISD\(^2\)) and Generalized Birthday Attack (GBA).

The drawback of code-based PKCs is, however, that the public-key size is large, which is \(k(n-k)\) bits if a binary code of length \(n\) with information rate \(k/n\) is used. To overcome this problem, several attempts have been conducted. They are summarized as follows.

(Potential approaches for reducing public-key size for code-based PKCs)

Enhancement of error correction capability:
- Capacity Approaching Codes
  - LDPC codes
  - QC-LDPC codes [3]
- List Decoding
  - Exhaustive search
  - List decoding for Goppa Code [6]
- Error expansion/hold [18]

Compression of public-key:
- Quasi-Cyclic Construction [4]
- Quasi-Dyadic Construction [21]
- Flexible-Quasi-Dyadic Construction (proposal)

Unfortunately, LDPC (Low-Density Parity Check) code approach has been broken in [22, 13] where [22] works if the density of the random nonsingular secret matrix \(S\) is low and [13] works for any \(S\). Error expansion/hold approach has been broken in [16]. Quasi-Cyclic and QC-LDPC approaches have been broken in [1, 28]. Quasi-Dyadic approach has been broken in [28], but only for small extension degrees [20]. Hence the remaining approaches are list decoding and Quasi-Dyadic approach for large extension degrees. While list decoding works, its effect is small since it can correct only a couple of more errors for practical parameters within practical decoding complexity. Hence the last resort is the quasi-dyadic approach with large extension degrees.

2 Quasi-Dyadic Construction

I will skip the preliminary of code-based PKCs, but you can find a lot of contents to explain them, e.g. in the surveys section of [5] or in [9].

Quasi-Dyadic construction was proposed in [21]. It uses the intersection between dyadic matrices and Goppa codes in Cauchy form. A \(2^v \times 2^v\) dyadic matrix \(M\) is in this form:

\[
M = \begin{bmatrix} A & B \\ B & A \end{bmatrix}
\]  \(1\)

where \(A\) and \(B\) are \(2^{v-1} \times 2^{v-1}\) dyadic matrices, respectively. The advantage of a dyadic matrix is that the whole matrix can be constructed from its one row or one column. This is the trick to reduce the public matrix.

\(^2\) In [12], it is referred to as ISD but in this paper we call it OISD to distinguish it from classical ISDs.
Table 1. Sample parameters of plain code-based PKE estimated in [12]

<table>
<thead>
<tr>
<th>m</th>
<th>t</th>
<th>n</th>
<th>BWF OISD (p.l)</th>
<th>Public-key size</th>
<th>Plaintext/Ciphertext</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>32</td>
<td>2,048</td>
<td>2^{56.8} (4, 24)</td>
<td>72.9KB</td>
<td>233/352 [bits]</td>
</tr>
<tr>
<td>12</td>
<td>41</td>
<td>4,098</td>
<td>2^{128.5} (10.54)</td>
<td>216.5KB</td>
<td>327/492 [bits]</td>
</tr>
</tbody>
</table>

Table 2. Sample parameters of Quasi-Dyadic (QD) code-based PKE [21]

<table>
<thead>
<tr>
<th>m</th>
<th>t</th>
<th>n</th>
<th>BWF OISD (p.l)</th>
<th>Public-key size</th>
<th>Plaintext/Ciphertext</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>64</td>
<td>2,560</td>
<td>2^{10.3} (1, 12)</td>
<td>3.0KB</td>
<td>427/1024 [bits]</td>
</tr>
<tr>
<td>16</td>
<td>64</td>
<td>3,072</td>
<td>2^{106.0} (2, 17)</td>
<td>4.0KB</td>
<td>445/1024 [bits]</td>
</tr>
<tr>
<td>16</td>
<td>128</td>
<td>4,096</td>
<td>2^{156.8} (2, 18)</td>
<td>4.0KB</td>
<td>817/1024 [bits]</td>
</tr>
</tbody>
</table>

Due to the following Theorem, it is possible to make a parity check matrix of the Goppa code Cauchy from.

**Theorem 1 (Goppa Codes in Cauchy Form [27, 19])** The Goppa code generated by a monic polynomial \( g(x) = (x - z_0) \cdots (x - z_{t-1}) \) without multiple zeros admits a parity-check matrix \( H \) whose \( i \)-th row and \( j \)-th column is \( H_{ij} = 1/(z_i - L_j) \) for \( 0 \leq i < t \) and \( 0 \leq j < n \).

The Cauchy matrix can be dyadic by choosing distinct \( z_i \) and \( L_j \) meeting the following conditions:

\[
\frac{1}{h_{i+j}} = \frac{1}{h_i} + \frac{1}{h_j} + \frac{1}{h_0} \quad (2)
\]

\[
z_i = \frac{1}{h_i} + \omega \quad (3)
\]

\[
L_j = \frac{1}{h_j} + \frac{1}{h_0} + \omega \quad (4)
\]

The construction algorithm proposed in [21] generates a sequence of \( h_i \) for \( 0 \leq i \leq N \) where \( n < N \) at random meeting (2) to (4). If they are not satisfied, it discards \( h_i \) and regenerates them until the conditions are satisfied. Using the generated \( h_i \), a \( N \times N \) full dyadic matrix can be constructed. It finally picks up a \( t \times n \) sub-matrix from the full \( N \times N \) dyadic matrix.

This algorithm is, however, restrictive on its parameter choice, i.e. \( n \ll 2^m - t \) must hold otherwise it eventually fails to generate a distinct set of \( z_i \) and \( L_j \), or takes a lot of time since it generates them by trial-and-error. This restriction prevents it from generating parameters for digital signatures since in digital signatures \( 2^m / \binom{m}{t} \) must be small enough and without making \( n \) close to \( 2^m - t \), \( 2^m / \binom{m}{t} \) cannot be small.

### 3 Flexible-Quasi-Dyadic Construction

To overcome the problems in QD, we propose a more flexible and efficient construction, which we call Flexible-Quasi-Dyadic (FQD) construction. FQD does not use trial-and-
error approach and generates distinct $z_i$ and $L_j$ with one shot even for $n = 2^n - t$. FQD does not have any restriction such as $n << 2^n - t$.

FQD construction is as follows. It firstly generates one small $u \times u$ dyadic matrix using $\delta_i$ for $0 \leq i < \log_2 u$. We call them “inner delta” since they define the inner structure of the $u \times u$ full dyadic matrix. Then FQD generates the other $u \times u$ full dyadic matrices by duplicating the inner structure of the first $u \times u$ full dyadic matrix but shifting them using both $\Delta_j$ and $\Delta'_j$ for $0 \leq j < [n/u]$ and $1 \leq i < [t/u]$, respectively. We call $\Delta_j$ and $\Delta'_j$ “outer delta” since they define the relationship among the full $u \times u$ dyadic matrices. FQD can also remove the block-wise permutation and removal in the key generation phase of QD since the choice of $\Delta_j$ and $n$ already includes them. This is another advantage of FQD.

Table 3. Sample parameters of Flexible-Quasi-Dyadic (FQD) code-based PKE (proposal)

<table>
<thead>
<tr>
<th>$m$</th>
<th>$t$</th>
<th>$n$</th>
<th>BWF</th>
<th>OISD ($p,l$)</th>
<th>Public-key size</th>
<th>Plaintext/Ciphertext</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>32</td>
<td>2,016</td>
<td>$2^{35.0}$ (4,24)</td>
<td>-</td>
<td>2.2KB</td>
<td>224/352 [bits]</td>
</tr>
<tr>
<td>11</td>
<td>37</td>
<td>1.984</td>
<td>$2^{90.8}$ (4,24)</td>
<td>-</td>
<td>2.1KB</td>
<td>262/407 [bits]</td>
</tr>
<tr>
<td>11</td>
<td>64</td>
<td>1.984</td>
<td>$2^{105.1}$ (4,25)</td>
<td>-</td>
<td>1.7KB</td>
<td>404/704 [bits]</td>
</tr>
<tr>
<td>11</td>
<td>96</td>
<td>1.920</td>
<td>$2^{71.0}$ (2,16)</td>
<td>-</td>
<td>1.2KB</td>
<td>546/1056 [bits]</td>
</tr>
<tr>
<td>11</td>
<td>112</td>
<td>1.920</td>
<td>$2^{105.3}$ (2,16)</td>
<td>-</td>
<td>0.92KB</td>
<td>546/1056 [bits]</td>
</tr>
<tr>
<td>12</td>
<td>19</td>
<td>4.064</td>
<td>$2^{35.0}$ (8,44)</td>
<td>-</td>
<td>5.6KB</td>
<td>171/228 [bits]</td>
</tr>
<tr>
<td>12</td>
<td>23</td>
<td>4.064</td>
<td>$2^{91.5}$ (8,44)</td>
<td>-</td>
<td>5.5KB</td>
<td>202/276 [bits]</td>
</tr>
<tr>
<td>12</td>
<td>32</td>
<td>4.064</td>
<td>$2^{111.5}$ (10,53)</td>
<td>-</td>
<td>5.4KB</td>
<td>266/384 [bits]</td>
</tr>
<tr>
<td>12</td>
<td>42</td>
<td>4.032</td>
<td>$2^{120.9}$ (9,49)</td>
<td>-</td>
<td>5.2KB</td>
<td>333/504 [bits]</td>
</tr>
<tr>
<td>12</td>
<td>64</td>
<td>4.032</td>
<td>-</td>
<td>$2^{157.4}$</td>
<td>4.8KB</td>
<td>470/768 [bits]</td>
</tr>
<tr>
<td>12</td>
<td>128</td>
<td>3.968</td>
<td>-</td>
<td>$2^{156.2}$</td>
<td>3.6KB</td>
<td>811/1536 [bits]</td>
</tr>
<tr>
<td>12</td>
<td>186</td>
<td>3.840</td>
<td>-</td>
<td>$2^{155.7}$</td>
<td>2.4KB</td>
<td>1069/2232 [bits]</td>
</tr>
<tr>
<td>12</td>
<td>256</td>
<td>3.840</td>
<td>$2^{91.5}$ (1,13)</td>
<td>-</td>
<td>1.1KB</td>
<td>1352/3072 [bits]</td>
</tr>
<tr>
<td>12</td>
<td>256</td>
<td>3.728</td>
<td>$2^{100}$ (1,13)</td>
<td>-</td>
<td>0.96KB</td>
<td>1340/3072 [bits]</td>
</tr>
</tbody>
</table>

Table 4. Sample parameters of plain code-based signature (CFS signature [7])

<table>
<thead>
<tr>
<th>$m$</th>
<th>$t$</th>
<th>$n$</th>
<th>GBA</th>
<th>BWF</th>
<th>OISD ($p,l$)</th>
<th>Public-key size</th>
<th>Iteration</th>
<th>Signature Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>11</td>
<td>524</td>
<td>288</td>
<td>$2^{53.6}$</td>
<td>-</td>
<td>13,370.7KB</td>
<td>$2^{25.4}$</td>
<td>209 (234.3) [bits]</td>
</tr>
<tr>
<td>15</td>
<td>12</td>
<td>376</td>
<td>288</td>
<td>$2^{51.5}$</td>
<td>-</td>
<td>716.0KB</td>
<td>$2^{28.8}$</td>
<td>180 (208.8) [bits]</td>
</tr>
<tr>
<td>15</td>
<td>13</td>
<td>376</td>
<td>$2^{54.8}$</td>
<td>-</td>
<td>775.4KB</td>
<td>$2^{27.5}$</td>
<td>195 (227.5) [bits]</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>168</td>
<td>384</td>
<td>$2^{84.0}$ (11.66)</td>
<td>-</td>
<td>387.3KB</td>
<td>$2^{65.4}$</td>
<td>196 (232.4) [bits]</td>
</tr>
<tr>
<td>14</td>
<td>15</td>
<td>168</td>
<td>$2^{89.7}$ (11.67)</td>
<td>-</td>
<td>414.6KB</td>
<td>$2^{49.3}$</td>
<td>210 (250.3) [bits]</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>16</td>
<td>8,192</td>
<td>$2^{83.5}$ (9.32)</td>
<td>-</td>
<td>202.7KB</td>
<td>$2^{44.3}$</td>
<td>208 (252.3) [bits]</td>
<td></td>
</tr>
</tbody>
</table>
Table 5. Sample parameters of Flexible-Quasi-Dyadic (FQD) code-based digital signature (proposal)

<table>
<thead>
<tr>
<th>$m$</th>
<th>$t$</th>
<th>$n$</th>
<th>$\text{BWF}$</th>
<th>$\text{GBA}$</th>
<th>$\text{OISD} (p.l.)$</th>
<th>Public-key size</th>
<th>Iteration</th>
<th>Signature Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>11</td>
<td>524, 272</td>
<td>$2^{35.0}$</td>
<td>-</td>
<td>1,215.5KB</td>
<td>1,223.3</td>
<td>209 (234.3)</td>
<td>[bits]</td>
</tr>
<tr>
<td>15</td>
<td>12</td>
<td>32, 752</td>
<td>$2^{31.3}$</td>
<td>-</td>
<td>59.6KB</td>
<td>2,28.8</td>
<td>180 (208.8)</td>
<td>[bits]</td>
</tr>
<tr>
<td>15</td>
<td>13</td>
<td>32, 752</td>
<td>$2^{38.8}$</td>
<td>-</td>
<td>59.6KB</td>
<td>2,28.5</td>
<td>195 (227.5)</td>
<td>[bits]</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>16, 368</td>
<td>$2^{38.1}$</td>
<td>-</td>
<td>27.6KB</td>
<td>2,28.9</td>
<td>196 (232.4)</td>
<td>[bits]</td>
</tr>
<tr>
<td>14</td>
<td>15</td>
<td>16, 368</td>
<td>-</td>
<td>$2^{37.2}$ (11, 67)</td>
<td>27.6KB</td>
<td>2,29.5</td>
<td>210 (250.3)</td>
<td>[bits]</td>
</tr>
<tr>
<td>13</td>
<td>16</td>
<td>8, 176</td>
<td>-</td>
<td>$2^{87.4}$ (9, 52)</td>
<td>12.6KB</td>
<td>2,24.3</td>
<td>208 (252.3)</td>
<td>[bits]</td>
</tr>
</tbody>
</table>

I will explain how to choose $\delta_i$, $\Delta_j$, and $\Delta'_j$ later on, but once they are determined, $z_i$ and $L_j$ are given as follows:

\[
\begin{align*}
  z_{i0} &= \sum_{b=0}^{\log_2 u-1} i_0[b] \cdot \delta_b & \text{for } 0 \leq i_0 < u \\
  z_{i1} &= z_{i0} \oplus \Delta'_1 & \text{for } 1 \leq i_1 < \lceil t/u \rceil \\
  L_{j1} &= z_{j0} \oplus \Delta_1 & \text{for } 0 \leq j_1 < \lceil n/u \rceil 
\end{align*}
\]

where $\oplus$ denotes exclusive-or, $i[b]$ and $j[b]$ denote $(b+1)$-th bit of $i$ and $j$ in the binary form, respectively. One can easily verify that $h_i = 1/(z_i \oplus L_j)$ makes a quasi- dyadic matrix. When $t \leq u$, $z_{i1}u+b0$ can be ignored. When $\lceil t/u \rceil \cdot u > t$ and/or $\lceil n/u \rceil \cdot u > n$, by removing $\lceil t/u \rceil \cdot u - t$ rows and $\lceil n/u \rceil \cdot u - n$ columns respectively, the size can be $t \times n$.

Another option is to add removed $z_i$ as $L_j$. This is useful to achieve $n = 2^m - t$ when $t \neq 2^k$ for any positive integer $x$.

The variables $\delta_i$, $\Delta_{j1}$ and $\Delta'_{j1}$ must be chosen at random while making all the $z_i$ for $0 \leq i < t$ and $L_j$ for $0 \leq j < n$ distinct, i.e.

\[
\begin{align*}
  z_i \oplus z'_i &\neq 0 & \text{for } i \neq i' \\
  L_j \oplus L'_j &\neq 0 & \text{for } j \neq j' \\
  z_i \oplus L_j &\neq 0 & \text{for } \forall i
\end{align*}
\]

These conditions are equivalent to the following conditions:

1. $\delta_b$ for $0 \leq b < \log_2 u$ are linearly independent.
2. $\forall r \in \{0, 1\}^{\log_2 u}$,

\[
\Delta'_{i1}, \Delta_{j1}, (\Delta'_{i1} \oplus \Delta_{j1}), (\Delta_{j1} \oplus \Delta'_{i1}), (\Delta_{j1} \oplus \Delta_{j1}) \not\subseteq \oplus_{b=0}^{\log_2 u-1} r[b] \cdot \delta_b
\]

where $r[b]$ denotes the $(b+1)$-th bit of $r$ in the binary form.

$\delta_b$, $\Delta'_j$, and $\Delta_j$ satisfying the above conditions can be generated by the following algorithm:

1. Generate a $m \times m$ random binary nonsingular matrix $M$.
2. Let the $(b+1)$-th row from the top of $M$ denote $\delta_b$ for $0 \leq b \leq (\log_2 u) - 1$.
3. Choose distinct $\Delta'_{j1}$ and $\Delta_{j1}$ from a linear combination of the bottom $m - \log_2 u$ rows of $M$.  

6
The cardinality of a nonsingular matrix $M$ is around $0.289 \cdot 2^m^2$, which is one of the secrets of FQD construction. Other secrets include permutation among $\Delta_j$, random scalar multiplication with each $u \times u$ full dyadic block and multiplication of non-singular random dyadic matrix $S$.

We show some sample parameters for binary codes in Table 1 to 5, but the idea of FQD construction can easily be extended to non-binary codes, too. In these tables, $m$, $t$ and $n$ are parameters of the underlying code. $m$ is the extension degree, $t$ is the error correction capability and $n$ is the code length. In plain (non-quasi-dyadic) schemes, $n = 2^m$ or $n < 2^m - t$ and in FQD, $n = 2^m - t$ (or $n < 2^m - t$). BWF is the minimal binary workfactor to break the system, which is either Optimized Information Set Decoding (OISD), Generalized Birthday Attack (GBA) or the attack in [28] on QD/FQD (we call it UL attack). The values of OISD and GBA follow the estimation in [12]. $p$ and $l$ are optimum parameters for OISD. In [28], the BWF of UL, $BWF_{UL}$ is estimated as $q^m \times (\log_2 q)^3 (v^2 + 3v + b)^2 v(v+b)$ where $v = \log_2 u$ and $b = \lceil n/u \rceil$, but this estimation is for $m = 2$. For $m \geq 2$, it is

$$BWF_{UL} = q^{m(m-1)} \times (\log_2 q)^3 (v^2 + 3v + b)^2 v(v+b)$$

(12)

In the columns of BWF “-” means the corresponding attack is less powerful. In the column of public-key size, $KB = 1024 \times 8$ bits. Plaintext/Ciphertext is the plaintext size and the ciphertext size in bits in the Niederreiter form. Iteration shows the signature generation cost, i.e. the number of trials to decode an error pattern corresponding to given syndromes. The signature size in ( ) is when the error pattern is expressed as the positions of $t$ errors. This increases the signature size but decreases the signature verification cost compared with the case where an error pattern is expressed as an integer between 0 and $\binom{n}{t} - 1$. The signature size can be reduced further by using the same technique in [7], i.e. by removing some error positions in the signature even though this increases the verification cost.

4 Conclusion

We proposed Flexible Quasi-Dyadic (FQD) construction, which can make the Quasi-Dyadic (QD) construction more flexible. FQD can achieve the maximum code length $n = 2^m - t$ with one shot whereas QD must hold $n < < 2^m - t$ and its key generation is performed by trial and error. FQD’s ability to make $n$ close to $2^m - t$ is crucial for code-based digital signatures since without this ability $2^m / \binom{n}{t}$ cannot be small and code-based digital signatures cannot be constructed. FQD can make $n$ close to $2^m - t$ and can even be used to reduce the signature-verification-key size of code-based digital signatures.

Acknowledgment

The author would like to thank Paulo S. L. M. Barreto, Rafael Misoczki and Yang Cui for fruitful discussions on the attacks on Quasi-Dyadic public-keys.
References