eMLE-Sig 2.0: A Signature Scheme based on Embedded Multilayer Equations with Heavy Layer Randomization

Dongxi Liu, Raymond K. Zhao

CSIRO Data61, Australia

30 May 2023

1 Introduction

eMLE-Sig 2.0 is a signature scheme based on a new version of Embedded Multilayer Equations (eMLE) problem. This new eMLE improves the security and efficiency of old eMLE [16] as introduced below.

Let d indicate the number of layers in eMLE and $\mathbf{p}$ be a list of d integers as the modulus of each layer. The bottom layer has $\mathbf{p}[0]$ as its modulus, the top layer has modulus $\mathbf{p}[d-1]$, and so on. All integers in $\mathbf{p}$ are co-prime, and $\mathbf{p}[i] < \mathbf{p}[j]$ holds for $0 \le i < j < d$. Let n be the integer indicating the dimension of all vectors in this report.

The following is an example of new eMLE with three layers (i.e., $d = 3$), where only $\mathbf{h} \in \mathbb{Z}^n_{\mathbf{p}[2]}$ and $\mathbf{g}_l \in \mathbb{Z}^n_{\mathbf{p}[l]}$ ($l \in \{0, 1, 2\}$) are public.

$$\begin{aligned}
\mathbf{h} \;&= \mathbf{g}_2 \otimes \mathbf{x} + \mathbf{h}_1 \bmod \mathbf{p}[2] \\
\mathbf{h}_1 &= (\mathbf{g}_1 \otimes \mathbf{x} + \mathbf{h}_0 \bmod \mathbf{p}[1]) + \mathbf{k}_1 * \mathbf{p}[1] \\
\mathbf{h}_0 &= (\mathbf{g}_0 \otimes \mathbf{x} \bmod \mathbf{p}[0]) + \mathbf{k}_0 * \mathbf{p}[0]
\end{aligned}$$

The operator $\otimes$ in the new eMLE above means the convolution product of two vectors [18]. Given two vectors $\mathbf{v}_1$ and $\mathbf{v}_2$, let $\mathbf{v} = \mathbf{v}_1 \otimes \mathbf{v}_2$. Then, $\mathbf{v}[i]$ ($0 \le i \le n-1$) is defined below.

$$\mathbf{v}[i] = \sum_{j=0}^{n-1} \mathbf{v}_1[j] * \mathbf{v}_2[(i-j) \bmod n]$$

The $\otimes$ operation is associative; that is, $(\mathbf{g} \otimes \mathbf{x}) \otimes \mathbf{c} = \mathbf{g} \otimes (\mathbf{x} \otimes \mathbf{c})$. It is also commutative but not realy needed by the signature scheme. Compared with old eMLE [16], new eMLE enhances its security (i.e., hardness of finding secret vector $\mathbf{x}$) and efficiency from the following aspects.

- Randomize internal layers $\mathbf{h}_1$ and $\mathbf{h}_0$ with random noises in $\mathbf{k}_1$ and $\mathbf{k}_0$. Compared with $\mathbf{x}$, the entries in $\mathbf{k}_1$ can contain much bigger random integers. Hence, random and big $\mathbf{k}_1$ makes the expected solution vector not a short one in the solution space;
- Use vector convolution to define values of each layer to allow much higher dimension of vectors (i.e., bigger n), without increasing much of signature sizes;
- Secret vector $\mathbf{x}$ can be configured to contain small values to reduce signature sizes and increase hardness.

1.1 Hardness of New eMLE – Overview

Based on Panny's attack to old eMLE[1], the new eMLE shown above can be flattened into the following form, from which the adversary attempts to find $\mathbf{x}$ by lattice reduction.

$$\mathbf{h} = \sum_{l=0}^{2} \mathbf{g}_l \otimes \mathbf{x} + \mathbf{k'}_0 * \mathbf{p}[0] + \mathbf{k'}_1 * \mathbf{p}[1] \bmod \mathbf{p}[2]$$

In the above form, $\mathbf{k'}_1$ can be as large as required by security requirements by configuring big enough top layer modulus $\mathbf{p}[2]$, while elements of $\mathbf{x}$ are fixed to small integers. As such, $(\mathbf{x}, \mathbf{k'}_0, \mathbf{k'}_1)$ is not a short integer (or shortest) solution to the flattened equations; the norm of $(\mathbf{x}, \mathbf{k'}_0, \mathbf{k'}_1)$ is dominated by random and big $\mathbf{k'}_1$. Lattice reduction

[1] Proposed by Lorenz Panny in NIST's PQC forum on 13 Oct 2021

based attack is then less effective to attack new eMLE. That is, there are solutions that have smaller norm than $(\mathbf{x}, \mathbf{k'}_0, \mathbf{k'}_1)$ but with different values for their $\mathbf{x}$, which is not a valid secret key in the signature scheme.

Moreover, an arbitrary solution to $\mathbf{x}$ for the above flattened equations, irrespective of its size, may not generate correct signatures because such $\mathbf{x}$ may lead to $\mathbf{h}_0$ and $\mathbf{h}_1$ that contain elements too big compared with upper layer modulus. If the adversary adds extra constrains on $\mathbf{x}$ in the above flatten equations to limit the elements in $\mathbf{h}_0$ and $\mathbf{h}_1$, then $\mathbf{x}$ in the solution will become bigger, as to be confirmed later with experiments. In the eMLE algorithm to be defined below, random entries in $\mathbf{h}_1$ could contain random big values that can be close to top modulus $\mathbf{p}[2]$. Hence, it is hard for the adversary to express accurate constrains on the elements in $\mathbf{h}_1$. In old eMLE, all elements in $\mathbf{h}_1$ are bounded by $\mathbf{p}[1]$, so the constraints can be effectively expressed in Panny's attack.

1.2 Notations

A lower-case boldface letter denotes a vector (e.g., $\mathbf{p}$). An upper-case boldface letter indicates a list of vectors (e.g., $\mathbf{G}$) or a matrix (e.g., $\mathbf{A}$). Given two integers a and b with $a < b$, $[a, b]$ means the set of integers $\{a, ..., b\}$. $x \leftarrow [a, b]$ means the uniformly sampling of integer $\mathbf{x}$ from the set $[a, b]$ at random. $\mathbf{0}$ is the zero vector of n-dimension. $\mathbf{1}$ is the vector in which all elements are 1. [] means an empty list.

Algorithm 1: eMLE Algorithm ($\mathbf{eMLE}$)

input : n, d, c_max, $\mathbf{p}$, $\mathbf{G}$, $\mathbf{x}$, $\mathbf{o}$, a
output: $\mathbf{h}$, $\mathbf{F}$, $sumR$

$\mathbf{h} = \mathbf{0}$; $\mathbf{F} = []$;
for $l = 0$ **to** $d - 1$ **do**
    if $l = 0$ **then**
        $\mathbf{h} = \mathbf{h} + \mathbf{G}[l] \otimes (\mathbf{x} + \mathbf{o}) \bmod \mathbf{p}[l]$
    else
        $\mathbf{h} = \mathbf{h} + \mathbf{G}[l] \otimes \mathbf{x} \bmod \mathbf{p}[l]$
    end
    if $l < d - 1$ **then**
        if $l = d - 2$ **then**
            $\mathbf{h}$, $sumR = \mathbf{randomize}(n, d, c\_max, \mathbf{p}, \mathbf{G}, \mathbf{h}, l, a)$
        end
        $\mathbf{F}[l] = \mathbf{h}$
    end
end
return $\mathbf{h}$, $\mathbf{F}$, $sumR$

Algorithm 2: eMLE Layer Randomization (**randomize**)

input : n, d, c_max, $\mathbf{p}$, $\mathbf{G}$, $\mathbf{h}$, l, a
output: $\mathbf{h}$, $sumR$

$num = \lfloor \frac{\mathbf{p}[l+1] - \sum_{i=0}^{n-1} \mathbf{h}[i] * (c\_max - 1)}{c\_max * \mathbf{p}[l]} \rfloor$
if $num < 0$ **then**
  $num = 0$
end
if $a = 1$ **then**
  $num = 2 * num$
end
$t = num$; $\mathbf{w} \leftarrow \mathbb{Z}_n^{\lfloor \frac{n}{2} \rfloor}$
for $j = 0$ **to** $\lfloor \frac{n}{2} \rfloor$-2 **do**
  $i = 0$
  if $\lfloor \frac{num}{\lfloor \frac{n}{2} \rfloor - j} \rfloor > 1$ **then**
    $i \leftarrow \mathbb{Z}_{\lfloor \frac{num}{\lfloor \frac{n}{2} \rfloor - j} \rfloor}$
  end
  $\mathbf{h}[\mathbf{w}[j]] = \mathbf{h}[\mathbf{w}[j]] + i * \mathbf{p}[l]$
  $num = num - i$
end
$\mathbf{h}[\mathbf{w}[\lfloor \frac{n}{2} \rfloor - 1]] = \mathbf{h}[\mathbf{w}[\lfloor \frac{n}{2} \rfloor - 1]] + num * \mathbf{p}[l]$
$w_0 \leftarrow \mathbb{Z}_n; w_1 \leftarrow \mathbb{Z}_n; i \leftarrow \mathbb{Z}_{\lfloor \frac{t}{3} \rfloor}$
for $j = 0$ **to** $n - 1$ **do**
  if $\mathbf{h}[(w_0 + j) \bmod n] < \mathbf{p}[l]$ **then**
    $\mathbf{h}[(w_0 + j) \bmod n] = \mathbf{h}[(w_0 + j) \bmod n] - i * \mathbf{p}[l]$
    break
  end
end
$i = \lfloor \frac{t}{3} \rfloor - i$
for $j = 0$ **to** $n - 1$ **do**
  if $\mathbf{h}[(w_1 + j) \bmod n] < \mathbf{p}[l]$ and $\mathbf{h}[(w_1 + j) \bmod n] \geq 0$ **then**
    $\mathbf{h}[(w_1 + j) \bmod n] = \mathbf{h}[(w_1 + j) \bmod n] - i * \mathbf{p}[l]$
    break
  end
end
$sumR = 0$
for $j = 0$ **to** $n - 1$ **do**
  if $\mathbf{h}[j] < \mathbf{p}[l]$ and $\mathbf{h}[j] \geq 0$ **then**
    if $a = 1$ **then**
      $i \leftarrow [-32 * n, 32 * n]$
    else
      $i \leftarrow [-16 * n, 16 * n]$
      $sumR = sumR + i$
    end
    $\mathbf{h}[j] = \mathbf{h}[j] + i * \mathbf{p}[l]$
  end
end
return $\mathbf{h}$, $sumR$

2 New eMLE Algorithm

New eMLE used by the signature scheme is defined in Algorithm 1. This algorithm will be called by the signature scheme during key generation with $a = 0$, and signing with $a = 1$. The input $\mathbf{G}$ is a list of d vectors, each of which is n-dimensional. With the eMLE example above, we have $\mathbf{G}[l] = \mathbf{g}_l$ for $0 \leq l \leq d-1$. Hence, $\mathbf{G}$ is public. The input c_max is an integer, which is a parameter for the signature scheme to be introduced later. The vector $\mathbf{o}$ is a public value, which is embedded at the bottom layer and can contain values like the hash of public key and message being signed.

The eMLE algorithm calculates the value of each layer, from bottom layer 0 ($l = 0$) to top layer ($l = d-1$), with the value of layer l added into the value of layer $l+1$. The top layer value $\mathbf{h}$ is returned, together with $\mathbf{F}$, which contains the values of $d-1$ lower layers. In the signature algorithm, $\mathbf{h}$ is made public, while $\mathbf{F}$ is kept secret as a part of private key.

As shown by line 10 in the Algorithm 1, layer $d-2$ is randomized (only layer $d-2$ is randomized for the signature scheme). Before randomization, the value of layer $d-2$ has each of its elements bounded by $\mathbf{p}[d-2]$. The randomization algorithm is given in Algorithm 2. The general idea of randomizing layer $d-2$ is to add multiples of $\mathbf{p}[d-2]$ into randomly selected entries in $\mathbf{h}$ at layer $d-2$. The randomisation is carried out in the following three parts.

- Line 1 – Line 17: num multiples are randomly distributed to $\lfloor \frac{n}{2} \rfloor$ random entries of $\mathbf{h}$, permitting repeated selection of entries. The variable num is doubled if eMLE is called from signing.
- Line 18 – Line 31: $\lfloor \frac{num}{3} \rfloor$ multiples of $\mathbf{p}[l]$ are split randomly and subtracted from two random entries of $\mathbf{h}$ (not overlapped with entries with multiples of $\mathbf{p}[l]$ added in the above step).
- Line 32 – Line 43: all other entries not randomized above are randomized. For key generation, random numbers are summarized into $sumR$ and returned.

The value of num and the constants like 16 and 32 in the randomisation algorithm are determined in experiments for the signature scheme by allowing as much as possible noises without sacrificing too much efficiency of key generation and signing.

2.1 Examples of Randomisation

As an example, let $d = 3$, $n = 64$, $\mathbf{p} = [5, 557, 67108864]$, $c\_max = 4$, and $\mathbf{x}$ have integer elements from $[-4, 4]$; this is a parameter set proposed later for Category I security. The following are two examples of the noises for layer 1 (i.e., corresponding to roughly $\mathbf{k}_1$ in the above eMLE example).

$\texttt{noise distribution at layer 1 (num = 30098) :}$
$690, -667, 752, 425, 423, 586, -4, -231, 130, 1963, 834, -692, -406, -77, -236,$
$4448, 1532, -592, 752, 581, 421, 5, 1335, 835, 375, -37, -157, 308, 607, 264, -3679,$
$-1008, 290, 327, -742, -788, -1, 401, -451, 555, -64, 1256, 321, -6353, -267,$
$534, 58, -470, 1797, -294, -934, 562, 1493, 616, 428, 1158, -6, 237, 584, 270, 4385,$
$1841, 515, 2517$

$\texttt{noise distribution at layer 1 (num} = \texttt{30094)}$:
$-177, 1091, 552, 958, 887, 2309, -825, 1124, -189, 1236, 242, 559, 729, -650, -637, -446, 222, -419, 46, 216, 565, 2435, 873, -481, -290, -1139, 1624, -806, 488, 1200, -420, 443, 139, 228, -421, 873, 1720, 1862, -8892, 502, -138, 0, 383, -687, 371, -607, 43, 517, -129, -176, 81, 477, -972, 559, 143, 679, -927, 477, 312, 8625, 412, -560, 992, 404$

Note that the variable num becomes bigger by configuring bigger top layer modulus $\mathbf{p}[d-1]$, with the size of $\mathbf{x}$ fixed. Hence, bigger $\mathbf{p}[d-1]$ means better security of $\mathbf{x}$ if all other parameters are the same, since layer $d-2$ is randomized with more noises and the norm of the solution vector $(\mathbf{x}, \mathbf{k}_0, \mathbf{k}_1)$ is dominated by big random values in $\mathbf{k}_1$. Bigger top modulus increases the size of public keys and signatures.

Continue with the above example, increasing $\mathbf{p}[2]$ from 67108864 to 4294967296 leads to the following noise distribution at layer 1 (In the randomisation algorithm, the constant 16 can be increased too for more noises when $\mathbf{p}[2]$ is increased; this example does not reflect this). The noise vector at layer 1 obviously has bigger norm than the above ones.

$\texttt{noise distribution at layer 1 (num} = \texttt{1927699)}$:
$200106, -407, -417, 700, 829, 975, -444, 908, 77266, -351, 867, -629, -97, 48139, 131889, 596, 761, -93, -138, 53795, -456, 25, 813, 966, 1003, 49361, 617, 776, -624803, -82, 97074, 537, 297245, 91075, 4840, 19058, 61253, -340, 63166, 100, 567, -17763, 735, 901, 43628, 8218, 880, -258, 627, 931, 22283, 107628, 960, -867, -433, 58896, 30767, 240608, 55160, 69351, 13560, -85, -884, 83333$

3 Signature Scheme over eMLE

In this section, we present the signature scheme eMLE-Sig 2.0, constructed over new eMLE. This signature scheme is defined over the following parameters, some of which have been introduced above:

- n : the default dimension of all vectors;
- d : the number of layers in eMLE, fixed to 3 in this report;
- $\mathbf{p}$: a list of d positive co-prime integers, with $\mathbf{p}[l]$ being the modulus for layer l for $0 \leq l \leq d-1$;
- $\mathbf{G}$: a list of d vectors, with $\mathbf{G}[l]$ used to build the value of layer l;
- x_max : an integer indicating the maximum of absolute values of elements in the secret vector $\mathbf{x}$;
- c_max : an integer limiting the elements in a challenge vector used in signing and verification algorithms;
- $\mathbf{vc}$: a list consisting of four integers, used to check the sizes of values in signature verification;
- $\mathcal{H}$: a hash function, such as SHA3-256.

The signature scheme eMLE-Sig 2.0 consists of three algorithms: key generation, signing, and verification.

Algorithm 3: Key Generation ($\mathbf{keyGen}$)

input : n, d, x_max, c_max, $\mathbf{p}$, $\mathbf{G}$
output: $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{F}_1$, $\mathbf{F}_2$, $\mathbf{h}_1$, $\mathbf{h}_2$, pkh

while `true` **do**
    $\mathbf{x}_1 \leftarrow [-x\_max, x\_max]^n$
    $\mathbf{x}_2 \leftarrow [-x\_max, x\_max]^n$
    $sumX = \Sigma_{i=0}^{n-1}(\mathbf{x}_1[i] + \mathbf{x}_2[i]))$
    if $|sumX| < \frac{n}{2}$ **then**
        break
    end
end
while `true` **do**
    $\mathbf{h}_1, \mathbf{F}_1, sumR_1 = \texttt{eMLE}(n, d, c\_max, \mathbf{p}, \mathbf{G}, \mathbf{x}_1, \mathbf{G}[1], 0)$
    $\mathbf{h}_2, \mathbf{F}_2, sumR_2 = \texttt{eMLE}(n, d, c\_max, \mathbf{p}, \mathbf{G}, \mathbf{x}_2, \mathbf{G}[1], 0)$
    if $|sumR_1 + sumR_2| < n * n$ **then**
        break
    end
end
$pkh = \mathcal{H}(\mathbf{h}_1, \mathbf{h}_2)$
return $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{F}_1$, $\mathbf{F}_2$, $\mathbf{h}_1$, $\mathbf{h}_2$, pkh

3.1 Key Generation

The key generation algorithm $\mathbf{keyGen}$ in Algorithm 3 starts by generating two random vectors $\mathbf{x}_1$ and $\mathbf{x}_2$. Each element in $\mathbf{x}_1$ and $\mathbf{x}_2$ is uniformly sampled from the set $[-x\_max, x\_max]$ at random. The absolute value of $\mathbf{x}_1$ and $\mathbf{x}_2$'s sum is required less than half of n, otherwise a resampling is needed.

With $\mathbf{x}_1$ and $\mathbf{x}_2$, the eMLE algorithm is called to generate $\mathbf{h}_1$,$\mathbf{h}_2$, $\mathbf{F}_1$, $\mathbf{F}_2$, $sumR_1$, and $sumR_2$. The absolute value of $sumR_1 + sumR_2$ is required less than $n*n$, otherwise eMLE algorithm is invoked again. The parameter $\mathbf{o}$ in eMLE takes $\mathbf{G}[1]$ as input, and a takes 0, indicating it is called from key generation.

The private key includes four vectors: $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{F}_1$, and $\mathbf{F}_2$. The public key is $\mathbf{h}_1$ and $\mathbf{h}_2$. The hash of $\mathbf{h}_1$ and $\mathbf{h}_2$ is stored into pkh and is also returned.

3.2 Signing

The signing algorithm is given in Algorithm 4. In addition to public parameters, it takes the private key ($\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{F}_1$, and $\mathbf{F}_2$), the hash of public key pkh, the message m, and its length $mlen$. The signature consists of two vectors $\mathbf{s}$ and $\mathbf{u}$.

The algorithm starts with calculating the sum of negative integers $sumXn$ and the sum of the positive integers $sumXp$ in $\mathbf{x}_1$ and $\mathbf{x}_2$, respectively. It then hashes the message m and pkh into two vectors $\mathbf{c}'_1$ and $\mathbf{c}'_2$.

In a while loop, the algorithm samples the random vector $\mathbf{y}$, with which eMLE algorithm called to generate $\mathbf{u}$ and $\mathbf{F}$. Then, $\mathbf{u}$, m, and pkh are hashed into two challenge vectors, $\mathbf{c}_1$ and $\mathbf{c}_2$, each element of which is from 0 to $c\_max - 1$. The hash algorithm

hashVec relies on hash function $\mathcal{H}$ to generate a bit stream and then splits the bit stream into the expected vectors $\mathbf{c}_1$ and $\mathbf{c}_2$. Then, the signature component $\mathbf{s}$ is generated and passed to the **check** algorithm, which is defined in Algorithm 5 and will be explained below. If the check is valid, the signature consisting of two vectors $\mathbf{s}$ and $\mathbf{u}$ is returned.

Each element of vector $\mathbf{y}$ is required to be in a range from y_min to $\lfloor \frac{n*x\_max*c\_max}{2} \rfloor - y\_gap$. This requirement is to reduce the number of loop repetitions in the signing algorithm because the **check** algorithm asks each element of $\mathbf{s}$ is in a particular range, no matter whether $\mathbf{x}_1$ and $\mathbf{x}_2$ contain more positive element or negative elements. Note that y_min and y_gap changes for each iteration in the while loop.

The **check** algorithm uses **checkS** defined in Algorithm 6 to check the validity $\mathbf{s}$ against the following conditions:

– each element of $\mathbf{s}$ must lie in between 0 and $\lfloor \frac{n*c\_max*x\_max}{2} \rfloor - 1$;
– the variance of $\mathbf{s}$ (or variant of variance) is in between $\mathbf{vc}[0]$ and $\mathbf{vc}[1]$.

Algorithm 4: Signing (**sign**)

input : n, d, x_max, c_max, $\mathbf{p}$, $\mathbf{G}$, $\mathbf{vc}$, $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{F}_1$, $\mathbf{F}_2$, pkh, m, $mlen$
output: $\mathbf{u}$, $\mathbf{s}$

Let $sumXn$ be the sum of negative integers in $\mathbf{x}_1$ and $\mathbf{x}_2$
Let $sumXp$ be the sum of positive integers in $\mathbf{x}_1$ and $\mathbf{x}_2$
$\mathbf{c}'_1, \mathbf{c}'_2 = \texttt{hashVec}(n, c\_max, m, mlen, \texttt{null}, pkh)$
while `true` **do**
  if $sumXp > |sumXn|$ **then**
    $y\_min \leftarrow [\lfloor \frac{|sumXn|*c\_max}{10} \rfloor, \lfloor \frac{|sumXn|*c\_max}{8} \rfloor]$
    $y\_gap \leftarrow [\lfloor \frac{sumXp*c\_max}{7} \rfloor, \lfloor \frac{sumXp*c\_max}{5} \rfloor]$
  else
    $y\_min \leftarrow [\lfloor \frac{|sumXn|*c\_max}{7} \rfloor, \lfloor \frac{|sumXn|*c\_max}{5} \rfloor]$
    $y\_gap \leftarrow [\lfloor \frac{sumXp*c\_max}{10} \rfloor, \lfloor \frac{sumXp*c\_max}{8} \rfloor]$
  end
  $\mathbf{y} \leftarrow [y\_min, \lfloor \frac{n*x\_max*c\_max}{2} \rfloor - y\_gap]^n$
  $\mathbf{u}, \mathbf{F}, \_ = \texttt{eMLE}(n, d, c\_max, \mathbf{p}, \mathbf{G}, \mathbf{y}, \mathbf{c}'_1 + \mathbf{c}'_2, 1)$
  $\mathbf{c}_1, \mathbf{c}_2 = \texttt{hashVec}(n, c\_max, m, mlen, \mathbf{u}, pkh)$
  $\mathbf{s} = \mathbf{x}_1 \otimes \mathbf{c}_1 + \mathbf{x}_2 \otimes \mathbf{c}_2 + \mathbf{y}$
  $v = \texttt{check}(n, d, x\_max, c\_max, \mathbf{p}, \mathbf{G}, \mathbf{vc}, \mathbf{F}_1, \mathbf{F}_2, \mathbf{F}, \mathbf{s}, \mathbf{c}_1, \mathbf{c}_2, \mathbf{c}'_1 + \mathbf{c}'_2)$
  if $v =$ `true` **then**
    break
  end
end
return $\mathbf{s}, \mathbf{u}$

After checking $\mathbf{s}$, for each internal layer (i.e., $l \leq d-2$), the **check** algorithm checks at line 8 whether each element of $\mathbf{t}$ is non-negative and does not touch the modulus of upper layer. This check is for ensuring correctness of signatures when layers are removed from top to bottom during verification. At the bottom layer (i.e., $l = 0$), $\mathbf{k}$ is calculated at line 14 by subtracting $\mathbf{t}$ by $\mathbf{G}[0] \otimes (\mathbf{s} + \mathbf{g} + \mathbf{c}') \bmod \mathbf{p}[0]$ and then divided by $\mathbf{p}[0]$; this division is an exact division for a correct signature. The variance of $\mathbf{k}$ is then checked. Note that this $\mathbf{k}$ is not the same as $\mathbf{k}_0$ in the flattened form of eMLE in Section 1.1, though they both contain coefficients of $\mathbf{p}[0]$. At line 13, $\mathbf{g}$ is calculated in that way because $\mathbf{G}[1]$ is embedded in the layer 0 of the public key and during signature verification the embedded $\mathbf{G}[1]$ is convoluted with $\mathbf{c}_1$ and $\mathbf{c}_2$.

Algorithm 5: Signature Validity ($\mathbf{check}$)

input : $n, d, x\_max, c\_max, \mathbf{p}, \mathbf{G}, \mathbf{vc}, \mathbf{F}_1, \mathbf{F}_2, \mathbf{F}, \mathbf{s}, \mathbf{c}_1, \mathbf{c}_2, \mathbf{c}'$
output: `true` or `false`

1 $v = \texttt{checkS}(n, d, x\_max, c\_max, \mathbf{vc}, \mathbf{s})$
2 **if** $v =$ `false` **then**
3   return `false`
4 **end**
5 **for** $l = d - 2$ **to** 0 **do**
6     $\mathbf{t} = \mathbf{F}_1[l] \otimes \mathbf{c}_1 + \mathbf{F}_2[l] \otimes \mathbf{c}_2 + \mathbf{F}[l]$
7   **for** $j = 0$ **to** $n - 1$ **do**
8   **if** $\mathbf{t}[j] < 0$ `or` $\mathbf{t}[j] \geq \mathbf{p}[l+1]$ **then**
9   return `false`
10   **end**
11   **end**
12   **if** $l = 0$ **then**
13         $\mathbf{g} = \mathbf{G}[1] \otimes (\mathbf{c}_1 + \mathbf{c}_2) \bmod \mathbf{p}[0]$
14         $\mathbf{k} = \frac{\mathbf{t} - (\mathbf{G}[0] \otimes (\mathbf{s} + \mathbf{g} + \mathbf{c}') \bmod \mathbf{p}[0])}{\mathbf{p}[0]}$
15         $a = \lfloor \frac{\sum_{i=0}^{n-1} \mathbf{k}[i]}{n} \rfloor$
16         $\mathbf{k} = \mathbf{k} - \mathbf{1} * a$
17   **if** $(\sum_{i=0}^{n-1} (\mathbf{k}[i] * \mathbf{k}[i]) < \mathbf{vc}[2])$ `or` $(\sum_{i=0}^{n-1} (\mathbf{k}[i] * \mathbf{k}[i]) > \mathbf{vc}[3])$ **then**
18   return `false`
19   **end**
20   **end**
21 **end**
22 return `true`

3.3 Verification

The verification algorithm is defined in Algorithm 7. This algorithm returns `true` if the signature $\mathbf{s}$ and $\mathbf{u}$ can be verified against message m with public key $\mathbf{h}_1$ and $\mathbf{h}_2$.

The algorithm starts by generating the hash values pkh, $\mathbf{c}'_1$, $\mathbf{c}'_2$, $\mathbf{c}_1$ and $\mathbf{c}_2$ with the same method and parameters as done in key generation and signing. The validity of $\mathbf{s}$ is checked with the algorithm `checkS`. And then $\mathbf{t}$ is initialized to $\mathbf{h}_1 \otimes \mathbf{c}_1 + \mathbf{h}_2 \otimes \mathbf{c}_2 + \mathbf{u} \bmod \mathbf{p}[d-1]$. In a loop, each layer of $\mathbf{t}$ is removed from top to bottom. Layer 2 and

Algorithm 6: Validity of s in Signature ($\mathbf{checkS}$)

input : $n, d, x\_max, c\_max, \mathbf{vc}, \mathbf{s}$
output: `true` or `false`

for $j = 0$ **to** $n - 1$ **do**
  if $\mathbf{s}[j] < 0$ `or` $\mathbf{s}[j] > \lfloor \frac{n*c\_max*x\_max}{2} \rfloor - 1$ **then**
    return `false`
  end
end
$a = \lfloor \frac{\sum_{i=0}^{n-1} \mathbf{s}[i]}{n} \rfloor$
$\mathbf{s}' = \mathbf{s} - \mathbf{1} * a$
if $(\sum_{i=0}^{n-1}(\mathbf{s}'[i] * \mathbf{s}'[i]) < \mathbf{vc}[0])$ `or` $(\sum_{i=0}^{n-1}(\mathbf{s}'[i] * \mathbf{s}'[i]) > \mathbf{vc}[1])$ **then**
  return `false`
end
return `true`

layer 1 are removed at line 15, respectively. Layer 0 is removed at line 13. Moreover for layer 0, $\mathbf{k}$ is calculated and checked in the same way as done in the signing algorithm.

If all conditions on $\mathbf{s}$ and $\mathbf{k}$ are satisfied, and $\mathbf{t}$ is a zero vector after all layers are removed, then the verification algorithm returns true.

Note that $\mathbf{t}$ at line 5, which is $\mathbf{h}_1 \otimes \mathbf{c}_1 + \mathbf{h}_2 \otimes \mathbf{c}_2 + \mathbf{u} \bmod \mathbf{p}[d-1]$, could have a flattened expression as that in Section 1.1. However, the vector $\mathbf{k}$ at line 9 of the verification algorithm is not the same as $\mathbf{k}_0$ in the flattened expression. Hence, in the flattened format of $\mathbf{t}$, $\mathbf{k}$ does not appear, making it hard to express the variance condition on $\mathbf{k}$ in lattice-based attack.

3.4 Correctness

The signature scheme eMLE-Sig 2.0 is correct in terms that for any key generated by Algorithm 3, and for any message m and its signature $\mathbf{s}, \mathbf{u}$ generated by Algorithm 4, the verification Algorithm 7 should return true.

The signing algorithm and the verification algorithm have the same way to check $\mathbf{s}$ and the value $\mathbf{k}$ at layer 0. So if the conditions hold in the signing algorithm (it should because of the application of `check` algorithm in signing), then conditions are also satisfied in the verification algorithm. Moreover, the conditions from line 6 to line 10 in the $\mathbf{check}$ algorithm ensures the values of lower layers in $\mathbf{h}_1 \otimes \mathbf{c}_1 + \mathbf{h}_2 \otimes \mathbf{c}_2 + \mathbf{u} \bmod \mathbf{p}[d-1]$ are not affected by removing the upper layer. Hence, after all layers are removed, a zero vector is returned.

4 Parameter Configurations

Three sets of parameters are provided for three security levels: 128-bit security level (`Security Level I`), 192-bit security level (`Security Level III`), and 256-bit security level (`Security Level V`). In the configuration, $\mathbf{p}$ is prepared in the following way,

Algorithm 7: Verification (**verify**)

input : n, d, x_max, c_max, $\mathbf{p}$, $\mathbf{G}$, $\mathbf{vc}$, $\mathbf{h}_1$, $\mathbf{h}_2$, $\mathbf{s}$, $\mathbf{u}$, m, $mlen$
output: $\mathbf{true}$ or $\mathbf{false}$

$pkh = \mathcal{H}(\mathbf{h}_1, \mathbf{h}_2)$
$\mathbf{c}_1', \mathbf{c}_2' = \texttt{hashVec}(n, c\_max, m, mlen, \texttt{null}, pkh)$
$\mathbf{c}_1, \mathbf{c}_2 = \texttt{hashVec}(n, c\_max, m, mlen, \mathbf{u}, pkh)$
$v = \texttt{checkS}(n, d, x\_max, c\_max, \mathbf{vc}, \mathbf{s})$
$\mathbf{t} = \mathbf{h}_1 \otimes \mathbf{c}_1 + \mathbf{h}_2 \otimes \mathbf{c}_2 + \mathbf{u} \bmod \mathbf{p}[d-1]$
for $l = d - 1$ **to** 0 **do**
  if $l = 0$ **then**
    $\mathbf{g} = \mathbf{G}[1] \otimes (\mathbf{c}_1 + \mathbf{c}_2) \bmod \mathbf{p}[0]$
    $\mathbf{k} = \frac{\mathbf{t} - (\mathbf{G}[0] \otimes (\mathbf{s} + \mathbf{g} + \mathbf{c}_1' + \mathbf{c}_2') \bmod \mathbf{p}[0])}{\mathbf{p}[0]}$
    $a = \lfloor \frac{\sum_{i=0}^{n-1} \mathbf{k}[i]}{n} \rfloor$
    $\mathbf{k} = \mathbf{k} - \mathbf{1} * a$
    $v = v \texttt{ and } (\sum_{i=0}^{n-1} (\mathbf{k}[i] * \mathbf{k}[i]) \geq \mathbf{vc}[2]) \texttt{ and } (\sum_{i=0}^{n-1} (\mathbf{k}[i] * \mathbf{k}[i]) \leq \mathbf{vc}[3])$
    $\mathbf{t} = \mathbf{t} - \mathbf{G}[l] \otimes (\mathbf{s} + \mathbf{g} + \mathbf{c}_1' + \mathbf{c}_2') \bmod \mathbf{p}[l]$
  else
    $\mathbf{t} = \mathbf{t} - \mathbf{G}[l] \otimes \mathbf{s} \bmod \mathbf{p}[l]$
  end
end
$v = v \texttt{ and } (\mathbf{t} = 0)$
return v

where p_max indicate the number of bits of the top layer modulus $\mathbf{p}[2]$, `next_prime` is a function returning the next prime of its input, and d is 3. p_max takes the value 26, 28, and 30, respectively, for the three security levels.

$$\mathbf{p}[l] = \begin{cases} \texttt{next\_prime}(c\_max), & \text{if } l = 0 \\ 2^{p\_max}, & \text{if } l = d-1 \\ \texttt{next\_prime}(\lfloor \frac{n}{2} \rfloor * (c\_max - 1) * \mathbf{p}[l-1] + \mathbf{p}[l-1] + n), & \text{otherwise} \end{cases}$$

The parameter $\mathbf{G}$ hard-coded in the reference implementation (named as `GG64`, `GG96`, `GG128` for three security categories) is calculated in the following way for the kth element at layer l: **SHA3-256**$(l, k, n, d, c\_max, x\_max, \mathbf{p}) \bmod \mathbf{p}[l]$. The parameters $\mathbf{G}$ and $\mathbf{p}$ are calculated in the accompanying SageMath implementation and then hard-coded in reference implementation.

The parameter $\mathbf{vc}$ is also hard-coded in the reference implementation, taking the following variable names and values: $\texttt{vc64} = [503673, 952989, 557, 1120]$, $\texttt{vc96} = [1756408, 2988441, 1336, 2368]$, and $\texttt{vc128} = [4229853, 6822141, 2507, 4079]$. Briefly, $\mathbf{vc}$ is determined by generating 500 key samples and signing 20 messages for each key, and then selecting the condition values for satisfying majority of signing and verification operations. The SageMath code generating $\mathbf{vc}$ is provided. Note that $\mathbf{vc}$ each time generated in the sage code is similar but not exactly the same due to randomness.

Security	n	d	x_max/c_max	**vc**	**p**	**G**
`Level I`	64	3	4	`vc64`	$[5, 557, 67108864]$	`GG64`
`Level III`	96	3	4	`vc96`	$[5, 823, 268435456]$	`GG96`
`Level V`	128	3	4	`vc128`	$[5, 1097, 1073741824]$	`GG128`

Table 1: Parameter Configurations

Suppose the security level is 128 bits. The parameters are required to satisfy the following basic conditions.

- $n * \log_2(\mathbf{p}[0]) \geq 128$, such that more than 128 bits of $\mathbf{t}$ are checked at line 18 of Algorithm 7.
- $n * \log_2(2 * x\_max + 1) \geq 128$, such that $\mathbf{x}_1$ or $\mathbf{x}_2$ has enough bits.
- $2 * n * \log_2(c\_max) \geq 256$, that is, a stream of 256 bits should be generated from hash function $\mathcal{H}$, with 128 bits taken by vector $\mathbf{c}_1$, and the other 128 bits assigned to $\mathbf{c}_2$.
- $d \geq 3$, such that there are at least two internal layers, layer $d-2$ for containing big random numbers and layer 0 for checking the variance of $\mathbf{k}$ at line 12 of the verification algorithm. Layer 0 is not randomized in eMLE, so $\mathbf{k}$ has small variances as reflected by the last two parameters in $\mathbf{vc}$.

The next section will give more analysis and evaluation on the proposed parameters.

5 Security Analysis and Evaluation

eMLE is defined with vector convolution. To compare with Short Integer Solution (SIS) problem and the application of Panny's attack, we need to replace vector convolution in eMLE with matrix and vector multiplication.

Given a n-dimensional vector $\mathbf{v} = (v_1, v_2, ..., v_n)$, let $\overleftrightarrow{\mathbf{v}}$ denote the following matrix.

$$\begin{bmatrix} v_1 & v_n & v_{n-1} \dots v_2 \\ v_2 & v_1 & v_n \dots v_3 \\ \dots & & \\ v_{n-1} & v_{n-2} & v_{n-3} \dots v_n \\ v_n & v_{n-1} & v_{n-2} \dots v_1 \end{bmatrix}$$

Then, we have $\mathbf{v} \otimes \mathbf{v}' = \overleftrightarrow{\mathbf{v}} * \mathbf{v}'$. Let $\mathbf{g} = \sum_{l=0}^{d-1} \mathbf{G}[l]$ and $\mathbf{A} = \overleftrightarrow{\mathbf{g}} \| (\mathbf{I} * \mathbf{p}[0]) \| ... \| (\mathbf{I} * \mathbf{p}[d-2])$, which means $\mathbf{A}$ is a $n * (d * n)$ matrix obtained by concatenating $\overleftrightarrow{\mathbf{g}}$, and $d-1$ identity matrices $\mathbf{I}$ each multiplied by $\mathbf{p}[l]$ for l from 0 to $d-2$. If $\mathbf{h}, \_, \_ = \texttt{eMLE}(n, d, c\_max, \mathbf{p}, \mathbf{G}, \mathbf{x}, \mathbf{o}, a)$, then the following equation can be obtained by flattening eMLE.

$$\mathbf{A} * \mathbf{s} = (\mathbf{h} - (\mathbf{G}[0] \otimes \mathbf{G}[1] \bmod \mathbf{p}[0]) \bmod \mathbf{p}[d-1],$$

where the first n elements in $\mathbf{s}$ correspond to $\mathbf{x}$. Recall that $\mathbf{G}[1]$ is embedded at the bottom layer of the public key, so $\mathbf{G}[0] \otimes \mathbf{G}[1] \bmod \mathbf{p}[0]$ is removed from the public key.

5.1 Comparison with Short Integer Solution (SIS)

The Short Integer Solution problem is defined over the equation $\mathbf{A}_{\texttt{SIS}} * \mathbf{s}_{\texttt{SIS}} = \mathbf{t} \bmod q$, which is similar in format as the flattened eMLE. In SIS, the solution vector $\mathbf{s}_{\texttt{SIS}}$ is required to contain small integers. Compared with SIS, $\mathbf{s}$ in eMLE contains very big integers in its last n elements; it is not hard to find a solution $\mathbf{s}$ that has first n elements small. However, to make the first n elements a valid private key in eMLE-Sig 2.0, only its small size is not sufficient, and it has to make different layers not interfering with each other for passing signature verification. Hence, with the above flattened format, the first n elements in solution $\mathbf{s}$ must be the original private key.

The hardness of eMLE can be increased by increasing the amount of noise at layer $d-2$, the dimension n, or both, while the hardness of SIS can ony be increased by choosing bigger n, when the range of $\mathbf{s}_{\texttt{SIS}}$ is fixed. Hence, for eMLE-Sig 2.0, we can have n that is small to allow the efficient application of current lattice reduction algorithm for concrete security evaluation, but the big noises to ensure the required security level. This makes cryptanalysis to eMLE-Sig 2.0 accessible to a large community.

In [17], a variant of SIS is defined with $\mathbf{A}_{\texttt{SIS}}$ being a $n * (2 * n)$ matrix. This SIS variant can be *roughly* changed into an eMLE instance by extending $\mathbf{A}_{\texttt{SIS}}$ into a $n * (3 * n)$ matrix after concatenating with $\mathbf{I} * \mathbf{p}[1]$ (with q used as top modulus $\mathbf{p}[2]$) and extending $\mathbf{s}_{\texttt{SIS}}$ with n random big integers selected by the adversary. Hence, if the eMLE problem can be efficiently solved (in terms that original private key can be recovered), then this algorithm could be used to attack that SIS variant.

This comparison does not attempt to be a security reduction, because the parameter set proposed in last section (i.e., $n = 64$, $n = 96$, and $n = 128$) is too small to be secure for SIS problem, and the reduction cannot reflect the feature of eMLE that hardness can be increased by adding more noises to layer $d-2$. However, the structure similarity could mean that since there are no efficient quantum algorithms to attack SIS, there should be no such algorithms to attack eMLE.

5.2 Comparison with Schnorr Signature and Σ Protocol

Schnorr signature is a well-studied signature scheme. The signature scheme eMLE-Sig 2.0 has exactly the same pattern of construction as Schnorr signature except for the difference of underlying hardness problems. In both schemes, the signer selects a random value (i.e., $\mathbf{y}$ in eMLE-Sig 2.0), and then generates a commitment to this value as one signature component (i.e., $\mathbf{u}$ in eMLE-Sig 2.0 defined over eMLE); with the hash of the commitment and the message (and public keys in eMLE-Sig 2.0), the second signature component is defined by multiplying the secret key with the supposedly-random hash and then blinding it with the random number from the first step.

Based on the hardness of discrete log, Schnorr signature is strongly unforgeable under chosen-message attacks (i.e., SUF-CMA secure). eMLE-Sig 2.0 should also be SUF-CMA secure, given the hardness of eMLE to be evaluated more in the next section.

By modeling the hash function as a Random Oracle Model, eMLE-Sig 2.0 can be regarded as an instance of Σ protocol. The knowledge extractor in Σ protocol usually takes the *re-winding* strategy to let the prover reuse the random number in the first message, and then extract the witness or private key from the conversation scripts.

If the prover is in a quantum state, the *re-winding* strategy is not reasonable for a proof of special soundness property [6]. Then, if the prover can be in a quantum state, the security of post-quantum signature schemes based on Σ protocol and Fiat-Shamir transformation needs to be re-analyzed with the Quantum Random Oracle Model (QROM). However, re-winding could be avoided for eMLE-Sig 2.0.

Given the random number $\mathbf{y}$, the prover in eMLE-Sig 2.0 can directly produce multiple (or two) $\mathbf{u}$ because eMLE is probabilistic and for each $\mathbf{u}$ a fresh challenge can be generated (even still with hash function) and the prover responds with the corresponding third message for each challenge. The same random number $\mathbf{y}$ is reused in the conversation scripts, without re-winding the prover.

5.3 Evaluation of eMLE's Concrete Security

The concrete security of eMLE-Sig 2.0 will be evaluated below with the attack proposed by Panny [2]. Panny's attack to eMLE uses lattice reduction. Based on the above comparison with SIS, this should be the most efficient way to attack eMLE. On the other hand, the proposed parameters (i.e., $n = 64$, $n = 96$, and $n = 128$) make the current lattice-reduction algorithms efficient enough to do the concrete evaluation. The SageMath implementation of eMLE-Sig 2.0 is used in the evaluation, with all experiment code provided for repeating and refining the evaluation.

Given the dimension n (no matter whether it is big or small), the principle underling the security of eMLE is that $(\mathbf{x}, \mathbf{k}_0, ..., \mathbf{k}_{d-2})$ is *not* a short integer solution in the solution space. This security requirement is achieved by selecting big enough $\mathbf{p}[d-1]$. In other words, a simple strategy to increase the security of the signature algorithm is to increase $\mathbf{p}[d-1]$. As an example, all three $\mathbf{p}[2]$s in Table 1 can be securely increased to 2^{32} for better performance on a 32-bit platform, at the cost of bigger public keys and signatures.

5.3.1 Effectiveness of the adapted attack method To apply Panny's attack, the vector convolution in new eMLE needs to be replaced by matrix and vector multiplication as illustrated above. The evaluation method is to limit the noises added to layer $d-2$ of the public key and then recover the private key by solving the equations of the flattened eMLE.

The noises added to the public key at layer $d-2$ in Algorithm 2 are limited by replacing i at lines 14, 21, 28, and 41 with $(i \bmod q)$, and replace num at line 17 with $(num \bmod q)$, where the value of q varies.

When q is small (e.g., $q = 256$ for $n = 64$ and $n = 96$), the private key can be recovered certainly. This experiment confirms the attack method is adapted correctly to new eMLE. When q is big enough (e.g., $q = 712$), the private key cannot be found in our experiment. The bigger n also makes it harder to find the private key. For example, when $n = 64$ and $q = 512$, the private key can be found, white it is not the case for $n = 96$.

This experiment is needed later when determining the concrete security level of each parameter set.

[2] Panny's code available at https://yx7.cc/files/emle-attack.tar.gz, announced in PQC Forum

5.3.2 Resilience to key recovery attack from public key

This experiment is to check whether the proposed parameters can ensure $(\mathbf{x}, \mathbf{k}_0, \mathbf{k}_1)$ used in the definition of the public key is not a short integer solution to the flattened eMLE equation, or to check whether the attack method can return a solution $\mathbf{s}$ that is shorter than $(\mathbf{x}, \mathbf{k}_0, \mathbf{k}_1)$.

For the convenience of experiments, the norm of $(\mathbf{x}, \mathbf{k}_0, \mathbf{k}_1)$ is calculated with the norm of $\mathbf{k}_1$ because $\mathbf{k}_1$ dominates the norm. Moreover, $\mathbf{k}_1$ itself is approximated by considering only i at lines 14, 21, 28, 41, and num at line 16 of Algorithm 2 as entry values. The SageMath code gives the explicit calculation of the approximated $\mathbf{k}_1$ and its norm. Table 2 listed the Euclidean norms obtained in the experiments. The norm of $\mathbf{k}_1$ is rounded to the smallest norm found in the experiments, while the norm $\mathbf{s}$ takes the biggest one. For the proposed parameters, the norm of $\mathbf{k}_1$ is bigger than that of $\mathbf{s}$. Hence, the security requirement that $(\mathbf{x}, \mathbf{k}_0, \mathbf{k}_1)$ is not a short integer solution is satisfied by the parameters.

n	Norm($\mathbf{k}_1$)	Norm($\mathbf{s}$)
64	11000	2900
96	27000	11000
128	76000	45000

Table 2: Comparisons of Norms

Moreover, only the equation $\mathbf{A} * \mathbf{s} = \mathbf{h}$ cannot ensure that each element in $\sum_{l=0}^{1} \mathbf{G}[l] \otimes \mathbf{s}[0:n] + \mathbf{k}_0 * \mathbf{p}[0] + \mathbf{k}_1 * \mathbf{p}[1]$ is non-negative and less than $\mathbf{p}[2]$, and each element in $\mathbf{G}[0] \otimes \mathbf{s}[0:n] + \mathbf{k}_0 * \mathbf{p}[0]$ is non-negative and less than $\mathbf{p}[1]$. Thus, if this $\mathbf{s}[0:n]$ is used to generate a signature, it fails to satisfy the condition $\mathbf{t} = 0$ at line 18 of Algorithm 7, let along other conditions in the verification algorithm, as shown in our experiment.

If extra constrains are added to consider the above two conditions (these extra constraints have been supported in Panny's attack code), then $\mathbf{t} = 0$ at line 18 of Algorithm 7 can be satisfied, but elements of $\mathbf{s}[0:n]$ become much bigger (extra constraints can only increase the size of the existing solutions) and hence cannot satisfy other checks in Algorithm 7.

5.3.3 Resilience to key recovery attack via signatures

Given a valid signature, the experiment is to check whether original $\mathbf{y}$ (the secret vector randomly sampled during signing) can be recovered from $\mathbf{u}$ in the signature by solving $\mathbf{A} * \mathbf{v} = \mathbf{u} - (\mathbf{G}[0] \otimes (\mathbf{c}'_1 + \mathbf{c}'_2) \bmod \mathbf{p}[0]) \bmod \mathbf{p}[2]$, where $\mathbf{c}'_1$ and $\mathbf{c}'_2$ are defined as at line 2 of Algorithm 7. If original $\mathbf{y}$ can be found from two signatures, then $\mathbf{x}_1$ and $\mathbf{x}_2$ in the secret key can be simply recovered from $\mathbf{s}$ in these signatures.

Note that during signing $\mathbf{y}$ is allowed to contain big elements, leading to bigger norm of $(\mathbf{y}, \mathbf{k}_0, \mathbf{k}_1)$. Our experiment shows that original $\mathbf{y}$ cannot be recovered from $\mathbf{u}$ because $\mathbf{k}_1$ is bigger enough for the proposed parameter sets and $\mathbf{y}$ itself is also big.

In addition, the value of each $\mathbf{y}$'s element is sampled in a range that is not known to the adversary and is bigger than the element of $\mathbf{x}_1 \otimes \mathbf{c}_1 + \mathbf{x}_2 \otimes \mathbf{c}_2$. Hence, $\mathbf{x}_1 \otimes \mathbf{c}_1 + \mathbf{x}_2 \otimes \mathbf{c}_2$ is statistically hiding in $\mathbf{s}$.

5.3.4 Strong Unforgeability under Chosen Message Attacks A signature in eMLE-Sig 2.0 consists of two vectors $\mathbf{s}$ and $\mathbf{u}$. If the hash function $\mathcal{H}$ used to generate $\mathbf{c}_1$ and $\mathbf{c}_2$ is collision-resistant and is modeled as a random oracle, then a new message (different from messages that have signatures available) leads to new random $\mathbf{c}_1$ and $\mathbf{c}_2$, and hence $\mathbf{s}$ in existing signatures cannot be reused. If a different signature is expected for an existing message, $\mathbf{u}$ must be different, and it thus leads to new random $\mathbf{c}_1$ and $\mathbf{c}_2$, causing a different $\mathbf{s}$ needed (i.e., $\mathbf{s}$ from the existing signatures not useful).

In addition, given $\mathbf{u}$, public key $\mathbf{h}_1$ and $\mathbf{h}_2$, let the hash values of $\mathbf{u}$ and message m and other parameters is $\mathbf{c}_1$ and $\mathbf{c}_2$. An experiment is carried out to check whether a valid signature component $\mathbf{s}$ (i.e., $\mathbf{v}[0 : n]$) can be recovered by solving the following equation.

$$\mathbf{A} * \mathbf{v} = \mathbf{h}_1 \otimes \mathbf{c}_1 + \mathbf{h}_2 \otimes \mathbf{c}_2 + \mathbf{u} - \mathbf{w} \bmod \mathbf{p}[d-1],$$

where $\mathbf{w} = \mathbf{G}[0] \oplus (\mathbf{G}[1] \oplus (\mathbf{c}_1 + \mathbf{c}_2) + \mathbf{c}_1' + \mathbf{c}_2') \bmod \mathbf{p}[0]$ and $\mathbf{c}_1', \mathbf{c}_2'$ are defined as at line 2 of Algorithm 7.

Similar to the second evaluation case above, when $\mathbf{v}[0 : n]$ and $\mathbf{u}$ is used as the fake signature, the condition $\mathbf{t} = 0$ at line 18 of Algorithm 7 does not hold, let along other verification conditions. This is because the lower layers have values exceeding the modulus of upper layer. If extra constrains are added, $\mathbf{v}[0 : n]$ will become much bigger and cannot pass other conditions in the verification algorithm.

The experiment is also carried out by limiting the noises in the public key as in the first experiment. Even with $q = 128$, a valid $\mathbf{s}$ cannot be generated. Recall that when $q = 128$, the first experiment can recover the private key. The condition value at line 12 of the Algorithm 7 cannot be linearly expressed, making it harder for attacks based on lattice-based reduction to forge a valid $\mathbf{v}[0 : n]$. Hence, the most efficient way for the adversary to forge a signature is to recover the private key from the pubic key.

5.4 Analysis of Security Levels

The parameters given in Table 1 are analyzed for their security levels in this section. The analysis is from two aspects: guessing the noises distributed at layer $d-2$ and guessing $\mathbf{x}_1$ and $\mathbf{x}_2$ directly. The first aspect is hinted by the first experiment above because it shows when the noises in layer $d-2$ become small enough (e.g., after some guessing and reducing), the original $\mathbf{x}$ can be recovered.

Let $\mathbf{k}_1$ contains noises added to the corresponding entries to $\mathbf{h}$. Then, Algorithm 8 is used to estimate security level from the first aspect. In the first experiment above, we have discussed when $q = 712$, the attack method cannot recover the private key. When calculating security level with this algorithm, 912 is used as the threshold. That is, a noise is counted when its absolute value is bigger than 912. Note that the attacker needs to guess the positions of noises bigger than 912 in order to reduce them and also guess how much of noises needs to remove.

Table 3 gives the security levels obtained by guessing noises at layer $d-2$ and guessing the private key ($\mathbf{x}_1$ or $\mathbf{x}_2$) directly. Note that the security level of the first aspect calculated by Algorithm 8 is random because $\mathbf{k}_1$ is random. The security level of guessing noises in Table 3 is the smallest security level observed by running Algorithm 8 for each parameter category a number of times.

Algorithm 8: Security Level Estimation by Guessing Noises

input : n, $\mathbf{k}_1$
output: $\mathtt{SL}$

```
c = 0
SL = 0
for i = 0 to n − 1 do
    if |k_1[i]| > 912 then
        c = c + 1
        SL = SL + log_2(|k_1[i]| − 912)
    end
end
SL = SL + log_2((c+n−1)! / (c! * (n−1)!))
return SL
```

By taking the smaller security levels obtained by guessing noises or by guessing private key, the security level for the three security categories are 145 bits, 304 bits, 405 bits.

	$n = 64$ (Level I)	$n = 96$ (Level III)	$n = 128$ (Level V)
Security Level (bits)	145	530	978
Security Level (bits)	202	304	405

Table 3: Security Levels By Guessing Noises at Layer $d - 2$ or Private Key

5.5 Security beyond Unforgeability

Security properties beyond unforgeability are formalized in [8]. These extra security properties include exclusive ownership, message-bound signatures, and non re-signability.

Briefly, eMLE-Sig 2.0 has all these three properties because both $\mathbf{s}$ and $\mathbf{u}$ are constructed with the hash of public keys and messages.

6 Implementation and Performance Evaluation

6.1 Pseudorandom Generator and Hash Function

We employ the AES-256 CTR mode as the pseudorandom generator (PRG). For the hash function $\mathcal{H}$ in $\mathbf{hashVec}$, we use SHA3-256, SHA3-384, and SHA3-512 for `Level I`, `Level III`, and `Level V`, respectively. Using AES as the PRG also enables the AES-NI hardware instructions on x64 CPUs [10], which significantly accelerates the Key Generation and Signing processes (see Table 4 in Section 6.4).

Let $\mathbf{SHA3}$ be the SHA3 hash function for the corresponding security level as above. Since $\mathbf{SHA3}$ will generate exactly $n/2$ bytes of output, the $\mathbf{hashVec}$ function is implemented as follows. Let $\mathbf{hc}$ be the hash output of $\mathbf{SHA3}(m||pkh||\mathbf{u})$, where the vector $\mathbf{u}$ is packed as bytes of the concatenation of its coordinates i.e. $\mathbf{u}[0]||\mathbf{u}[1]||\ldots||\mathbf{u}[n-1]$, such that each $\mathbf{u}[i]$ is represented as $\log_2(\mathbf{p}[2])$ bits unsigned integer in little endian form without padded 0 in the most significant bits. The output vectors $\mathbf{c}_1, \mathbf{c}_2$ have $\mathbf{c}_1[4i+j] = (\mathbf{hc}[i] >> 2j) \bmod 4, \mathbf{c}_2[4i+j] = (\mathbf{hc}[n/4+i] >> 2j) \bmod 4$, for $i = 0, 1, \ldots, n/4-1$, j from 0 to 3, where $>>$ is the right shift operation. SHA3 can be replaced with SHA2 for the same hash output length in the implementation of $\mathbf{hashVec}$ function.

6.2 Uniform Sampling

To generate uniformly random integers in $[a, b]$ for arbitrary inputs $a, b \in \mathbb{Z}$, we use the following rejection sampling method. Let $m = b - a + 1$, $k = \lceil \log_2(m) \rceil$. Then, let z be a random x bytes integer drawn from the PRG such that $8(x-1) < k \leq 8x$. Let $z' = z \bmod 2^k$. We output $z' + a$ as the result when $z' < m$, or discard z' otherwise.

However, the acceptance rate $m/2^k$ in this sampling process may leak secret information when m is derived from secret e.g. when generating i during the eMLE layer randomization (Algorithm 2) and $\mathbf{y}$ in the signing (Algorithm 4). To mitigate the leakage, we adapt a similar countermeasure to the isochronous sampler used by the Falcon signature [9, 11]. We perform another rejection on each sample with acceptance rate $ccs = 2^{k-1}/m$ i.e. the sampler outputs $z' + a$ when both $z' < m$ and $r < ccs$ hold for random real number $r \in [0, 1)$. The acceptance rate becomes $ccs \cdot m/2^k = 1/2$, which is independent of m.

To avoid generating a uniformly random real r with high absolute precision, we use the comparison technique from the FACCT sampler [20]. Assume an IEEE-754 floating-point value $f \in (0, 1)$ with $(\delta_f + 1)$-bit precision is represented by $f = \left(1 + mantissa \cdot 2^{-\delta_f}\right) \cdot 2^{exponent}$, where integer $mantissa$ has δ_f bits and $exponent \in \mathbb{Z}^-$. To check $r < f$, one can sample $r_m \hookleftarrow \{0,1\}^{\delta_f+1}$, $r_e \hookleftarrow \{0,1\}^l$ uniformly, and check $r_m < mantissa + 2^{\delta_f}$ and $r_e < 2^{l+exponent+1}$ for some l such that $l + exponent + 1 \geq 0$. By $2^{k-1} < m \leq 2^k$, we have $ccs \in [1/2, 1)$, $exponent \geq -2$, and $l \geq 1$ (we assume ccs may contain relative error less than $1/2$).

Additionally, when modifying any coordinate of $\mathbf{h}$ in $\mathbf{randomize}$ (Algorithm 2), since the randomly generated indices are secret, to avoid leakage due to caching, we always access every coordinate in $\mathbf{h}$ and use the constant-time select [2] to set the value. We also adapt the constant-time select [2] when the branch condition depends on the secret.

6.3 Convolution

We use the Karatsuba+schoolbook polynomial multiplication to realize the convolution. This approach is similar to the bottom layers of the polynomial multiplications in the Saber KEM [5]. Because the schoolbook multiplication outperforms Karatsuba when the polynomial degree is less than 20 [12], we use the schoolbook method instead of extra Karatsuba layers at the bottom of the recursion after the polynomial degree is reduced to less than 20. Plantard's modular reduction [19] is used for the `mod` operation. We do not perform the `if` check in the Plantard's reduction, which reduces the output x to 0 when x is equal to the modulus p. We have checked that our modified modular reduction will output the correct reduction result $x \in [0, p-1]$ for every positive integer input less than about 2^{30} when $p = \mathbf{p}[0]$, and this holds for every positive integer input less than 2^{32} when $p = \mathbf{p}[1]$. When `mod` $\mathbf{p}[2]$, since $\mathbf{p}[2]$ is power of 2, we use the bitwise `and` operation with $(\mathbf{p}[2] - 1)$ instead.

Additionally, we adapt the Kronecker substitution [7] to accelerate the polynomial multiplication with modulus $\mathbf{p}[0] = 5$ as follows. Assume the input vectors $\mathbf{a}, \mathbf{b} \in [0, 4]^n$. Let $\mathbf{c}$ be the polynomial multiplication of $\mathbf{a}, \mathbf{b}$ without `mod` 5. We have $\mathbf{c}[i] \leq n * 4^2$. Let $k = \lceil \log_2(16n + 1) \rceil$. Thus, by Kronecker substitution, we can evaluate $\mathbf{a}(2^k), \mathbf{b}(2^k)$, compute the integer multiplication $\mathbf{a}(2^k) * \mathbf{b}(2^k)$, and unpack the result. To implement $\mathbf{a}(2^k) * \mathbf{b}(2^k)$ where both $\mathbf{a}(2^k), \mathbf{b}(2^k)$ have nk bits, we use the aforementioned Karatsuba+schoolbook multiplication, with coefficient size w close to the machine word size and lower input polynomial degree $\lceil nk/w \rceil$ compared to n in the naive multiplication of $\mathbf{a}, \mathbf{b}$. For example, when $n = 64$, we have $k = 11$. Both $\mathbf{a}(2^k), \mathbf{b}(2^k)$ have $nk = 704$ bits. We select $w = 29$ and get the input polynomial degree $\lceil nk/w \rceil = 25$ in the Karatsuba+schoolbook big integer multiplication. Let $\mathbf{c}'$ be the multiplication result before processing the carries. We check $\mathbf{c}'[i] \leq 25 * (2^{29} - 1)^2 < 2^{63}$ i.e. coefficients in the intermediate result of the multiplication will not exceed the machine word size. In this example, the input polynomial degree is reduced to 39% compared to the naive polynomial multiplication.

6.4 Performance Evaluation

We evaluate the speed of our reference implementation (i.e. without hand-optimized CPU-specific instructions such as AVX-2) by measuring the CPU cycles on a laptop running Linux operating system with an Intel i5-11400H CPU at 2.70GHz and 64 gigabytes memory. We use the gcc 12.2.0 with option `-O3` to compile the code.[3] We measure the average number of CPU cycles consumed by 1000 iterations of $\mathbf{keyGen}$, $\mathbf{sign}$, and $\mathbf{verify}$ algorithms, respectively. Hyper-threading and Turbo Boost are disabled during the benchmark. Results are summarized in Table 4. In the KeyGen Speed and Sign Speed columns, the two values are the measured number of CPU cycles without/with using the AES-NI hardware instructions in the PRG, respectively.

Public key and signature sizes (bytes) are summarized in the PK Size and Sig Size columns in Table 4. Coordinates $\mathbf{h}_1[i], \mathbf{h}_2[i]$ of the public key and $\mathbf{u}[i]$ of the signature are $\log_2(\mathbf{p}[2])$ bits unsigned integers. By $\mathbf{checkS}$ (Algorithm 6), coordinates $\mathbf{s}[i]$ of the signature are in $[0, n * c\_max * x\_max/2 - 1]$. For $n = 64$, $\mathbf{s}[i]$ has 9 bits, and for $n = 96, 128$, $\mathbf{s}[i]$ has 10 bits. Similar to the representation of $\mathbf{u}$ in $\mathbf{hashVec}$, we pack these vectors as bytes of the concatenation of their coordinates, such that each coordinate is represented in little endian form without padded 0 in the most significant bits.

From Table 4, for the same security categories, the Sign/Verify speed of our reference implementation is already on par with the hand-optimized AVX-2 implementa-

[3] Run `make benchmark` to compile the `benchmark` program.

Security	n	KeyGen Speed	Sign Speed	Verify Speed	PK Size	Sig Size	SK Size
`Level I`	64	272630/56310	192723/52597	21755	416	280	800
`Level III`	96	375701/88465	272574/80653	38175	672	456	1200
`Level V`	128	455837/112167	343007/114965	63110	960	640	1600

Table 4: Performance Summary

tions of Dilithium [4] and Falcon [9] signatures. The KeyGen speed of our reference implementation is on par with the reference implementation of Dilithium. In addition, our KeyGen/Sign implementations with the AES-NI hardware instructions have the fastest speed compared to the NIST selected signature schemes for standardization implemented with hardware instructions on x64 platforms (Dilithium [4] with AVX-2 and AES-NI, Falcon [9] with AVX-2 and FMA, and SPHINCS+ [3] with Haraka [14] and AES-NI). Note that since the CPU in our benchmark platform supports the AVX-512 instruction set, the compiler may generate AVX-512 or VAES instructions during the benchmark. However, we did not write any hand-optimized AVX-512 assembly codes in our implementation.

For the same security categories, our signature scheme also has smaller signature size and smaller sum of public key and signature sizes compared to the NIST selected signature schemes for standardization (Dilithium [4], Falcon [9], and SPHINCS+ [3]).

7 Advantages and Limitations

7.1 Advantages

– **Better Security Certainty:** The condition underlying the security of eMLE-Sig 2.0 can be verified for the proposed parameters with experiments; if the condition has been verified (i.e., an expected solution is not a short one), then this condition will hold no matter how attack methods will be improved in the future.
– **More accessible cryptanalysis:** The proposed dimension parameter n makes the current lattice reduction algorithms efficient enough to solve flattened eMLE equations.
– **Compactness:** Compared to the NIST selected signature schemes for standardization, on the same security categories, our design has more compact signature and smaller sum of public key and signature sizes.
– **Simple Design:** The main arithmetic component of our design is convolution, which is simple and easy to understand. eMLE-Sig 2.0 is similar to Schnorr signature in structure and conceptually simple. Our design does not require developers knowledgeable in advanced techniques such as Number Theoretic Transform or Merkle Tree. Using a naive implementation or existing arithmetic libraries for the convolution is unlikely to affect the correctness of the implementation. In addition, the convolution is highly parallelizable on platforms such as the GPU [15].
– **Speed:** On the same security categories, the speed of our reference implementation is significantly faster than the reference implementations of the NIST selected

signature schemes for standardization, and competitive compared to their hand-optimized AVX-2 implementations. The Key Generation and Signing speed of our implementation can be accelerated with the AES-NI hardware instructions on x64 CPUs, and the resulted implementation is faster than the NIST selected signature schemes for standardization implemented with hardware instructions on x64 platforms.

7.2 Limitations

- **Necessity of 32-bit Integer Multiplication:** Our design requires 32×32 bits integer multiplication during the convolution with modulus $\mathbf{p}[2]$. This may affect the speed of the implementation on constrained devices (e.g. 16-bit or 8-bit microcontrollers) where such multiplication instructions may not be available on hardware. On the other hand, if allowing a bit increase of the sizes of public keys and signatures, the top layer modulus in all three proposed parameter sets can be increased to 2^{32} to make the shceme more efficient on 32-bit platform, since the reduction $\texttt{mod}\ 2^{32}$ can be done implicitly.
- **Side-channels:** As discussed in Section 6, for timing/cache side-channels, although we have adapted some countermeasures for platform-agnostic leakage, we did not consider the platform or compiler specific leakage due to e.g. arithmetic such as divisions [1] or even multiplications [13] in our current implementation. In addition, protections against other side-channels such as power analysis require future study.

References

1. Andrysco, M., Nötzli, A., Brown, F., Jhala, R., Stefan, D.: Towards verified, constant-time floating point operations. In: CCS. pp. 1369–1382. ACM (2018)
2. Aumasson, J.P.: Guidelines for low-level cryptography software (2019)
3. Aumasson, J.P., Bernstein, D.J., Dobraunig, C., Eichlseder, M., Fluhrer, S., Gazdag, S.L., Hülsing, A., Kampanakis, P., Kölbl, S., Lange, T., et al.: Sphincs+-submission to the 3rd round of the nist post-quantum project (2020)
4. Bai, S., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: Crystals-dilithium: Algorithm specifications and supporting documentation (version 3.1). NIST Post-Quantum Cryptography Standardization Round 3 (2021)
5. Basso, A., Mera, J.M.B., D'Anvers, J.P., Karmakar, A., Roy, S.S., Van Beirendonck, M., Vercauteren, F.: Saber: Mod-lwr based kem (round 3 submission). Online publication (2020)
6. Boneh, D., Dagdelen, Ö., Fischlin, M., Lehmann, A., Schaffner, C., Zhandry, M.: Random oracles in a quantum world. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. pp. 41–69 (2011)
7. Bos, J.W., Renes, J., van Vredendaal, C.: Post-quantum cryptography with contemporary co-processors: Beyond kronecker, schönhage-strassen & nussbaumer. In: USENIX Security Symposium. pp. 3683–3697. USENIX Association (2022)
8. Cremers, C., Düzlü, S., Fiedler, R., Fischlin, M., Janson, C.: Buffing signature schemes beyond unforgeability and the case of post-quantum signatures. IACR Cryptol. ePrint Arch. p. 1525 (2023), https://eprint.iacr.org/2020/1525
9. Fouque, P.A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Prest, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z.: Falcon: Fast-fourier lattice-based compact signatures over ntru, specification v1. 2. NIST Post-Quantum Cryptography Standardization Round 3 (2020)

10. Gueron, S.: Intel's new AES instructions for enhanced performance and security. In: FSE. Lecture Notes in Computer Science, vol. 5665, pp. 51–66. Springer (2009)
11. Howe, J., Prest, T., Ricosset, T., Rossi, M.: Isochronous gaussian sampling: From inception to implementation. In: PQCrypto. Lecture Notes in Computer Science, vol. 12100, pp. 53–71. Springer (2020)
12. Kannwischer, M.J., Rijneveld, J., Schwabe, P.: Faster multiplication in \mathbb z_2ˆm[x] on cortex-m4 to speed up NIST PQC candidates. In: ACNS. Lecture Notes in Computer Science, vol. 11464, pp. 281–301. Springer (2019)
13. Kaufmann, T., Pelletier, H., Vaudenay, S., Villegas, K.: When constant-time source yields variable-time binary: Exploiting curve25519-donna built with MSVC 2015. In: CANS. Lecture Notes in Computer Science, vol. 10052, pp. 573–582 (2016)
14. Kölbl, S., Lauridsen, M.M., Mendel, F., Rechberger, C.: Haraka v2 - efficient short-input hashing for post-quantum applications. IACR Trans. Symmetric Cryptol. 2016(2), 1–29 (2016)
15. Lee, W., Seo, H., Hwang, S.O., Achar, R., Karmakar, A., Mera, J.M.B.: Dpcrypto: Acceleration of post-quantum cryptography using dot-product instructions on gpus. IEEE Trans. Circuits Syst. I Regul. Pap. 69(9), 3591–3604 (2022)
16. Liu, D.: Embedded multilayer equations: a new hard problem for constructing post-quantum signatures smaller than RSA (without hardness assumption). IACR Cryptol. ePrint Arch. (2021), https://eprint.iacr.org/2021/1338
17. Lyubashevsky, V.: Lattice signatures without trapdoors. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. Lecture Notes in Computer Science, vol. 7237, pp. 738–755. Springer (2012)
18. Micciancio, D.: Generalized compact knapsacks, cyclic lattices, and efficient one-way functions. Comput. Complex. 16(4), 365–411 (2007)
19. Plantard, T.: Efficient word size modular arithmetic. IEEE Trans. Emerg. Top. Comput. 9(3), 1506–1518 (2021)
20. Zhao, R.K., Steinfeld, R., Sakzad, A.: FACCT: fast, compact, and constant-time discrete Gaussian sampler over integers. IEEE Trans. Computers 69(1), 126–137 (2020)