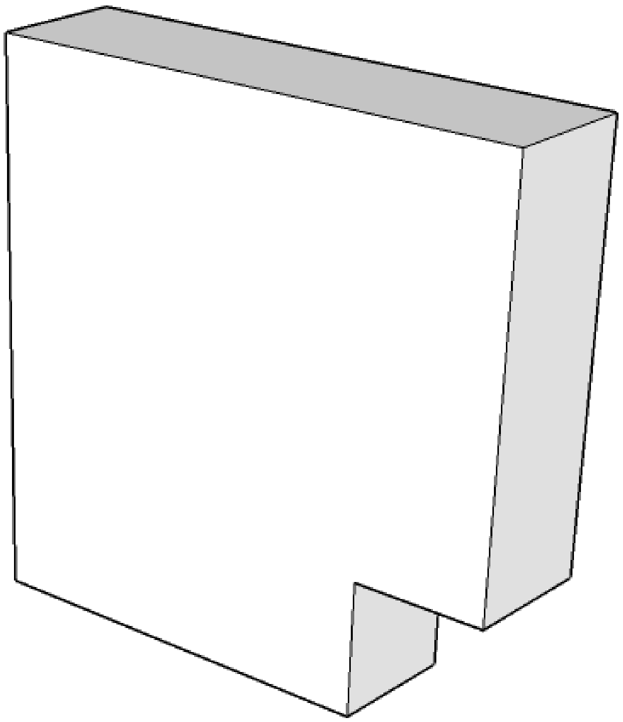
LUOV

Signature Scheme proposal for NIST PQC Project
(Round 2 version)



Principal submitter Ward Beullens, imec-COSIC KU Leuven
`ward.beullens@esat.kuleuven.be`
+32471 12 64 57
Afdeling ESAT - COSIC,
Kasteelpark Arenberg 10 - bus 2452,
3001 Heverlee, Belgium

Auxiliary submitters Bart Preneel, imec-COSIC KU Leuven
Alan Szepieniec, imec-COSIC KU Leuven
Frederik Vercauteren, imec-COSIC KU Leuven

Inventors/developers The same as the principal submitter. Relevant prior work is credited below where appropriate.

Owner Same as submitter

Signature

Contents

1 Introduction

One of the major candidates for providing secure cryptographic primitives in a post-quantum world is Multivariate Cryptography. Multivariate Cryptography is based on the hardness of problems related to multivariate polynomials over finite fields, such as solving systems of multivariate polynomial equations. In general, Multivariate Cryptography is very fast and requires only moderate computational resources, which makes it attractive for applications in low-cost devices. In the field of Multivariate Cryptography, the Unbalanced Oil and Vinegar signature scheme (UOV) is one of the oldest and best studied cryptosystems. Since the proposal of the Oil and Vinegar scheme in 1997 by Patarin [17], UOV has successfully withstood almost two decades of cryptanalysis. The UOV scheme is very simple, has small signatures and is fast. The main disadvantage of UOV is arguably that its public keys are quite large. This document presents the Lifted Unbalanced Oil and Vinegar signature scheme (LUOV), which is a simple improvement of the UOV scheme that greatly reduces the size of the public keys.

2 Algorithm specification (part of 2.B.1)

2.1 Overview of the scheme

The LUOV signature scheme uses a one-way function $\mathcal{P} : \mathbb{F}_{2^r}^n \rightarrow \mathbb{F}_{2^r}^m$, which is a multivariate quadratic polynomial map in $n = m + v$ variables with coefficients in the binary subfield $\mathbb{F}_2 \subset \mathbb{F}_{2^r}$. The trapdoor is a factorization $\mathcal{P} = \mathcal{F} \circ \mathcal{T}$, where $\mathcal{T} : \mathbb{F}_{2^r}^n \rightarrow \mathbb{F}_{2^r}^n$ is an invertible linear map, and $\mathcal{F} : \mathbb{F}_{2^r}^n \rightarrow \mathbb{F}_{2^r}^m$ is a quadratic map whose components $f_1, \cdots, f_m$ are of the form

$$f_k(\mathbf{x}) = \sum_{i=1}^{v} \sum_{j=i}^{n} \alpha_{i,j,k} x_i x_j + \sum_{i=1}^{n} \beta_{i,k} x_i + \gamma_k \,,$$

where the $\alpha_{i,j,k}, \beta_{i,k}$ and γ are chosen randomly from $\mathbb{F}_2$ and $v = n - m$. We say that the first v variables $x_1, \cdots, x_v$ are the *vinegar* variables, whereas the remaining m variables are the *oil* variables. Equivalently, the components of $\mathcal{F}$ are quadratic polynomials with random binary coefficients in the variables x_i such that there are no quadratic terms which contain two oil variables. One could say that the vinegar variables and the oil variables are not fully mixed, which is where their names come from.

How does the trapdoor $\mathcal{P} = \mathcal{F} \circ \mathcal{T}$ help to invert the function $\mathcal{P}$? Given a target $\mathbf{x} \in \mathbb{F}_{2^r}^m$ a solution $\mathbf{y}$ for $\mathcal{P}(\mathbf{y}) = \mathbf{x}$ can be found by first solving $\mathcal{F}(\mathbf{y}') = \mathbf{x}$ for $\mathbf{y}'$ and then computing $\mathbf{y} = \mathcal{T}^{-1}(\mathbf{y}')$. The system $\mathcal{F}(\mathbf{y}') = \mathbf{x}$ can be solved efficiently by fixing the vinegar variables to some randomly chosen values. If we substitute these values in the equations the remaining system only contains linear equations, because every quadratic term contains at least one vinegar variable and thus turns into a linear or constant term after substitution. The remaining linear system can be solved using linear algebra. In the event that there are no solutions we can simply try again with a different assignment to the vinegar variables.

The trapdoor function is then combined with a collision resistant hash function $\mathcal{H} : \{0,1\}^* \to \mathbb{F}_{2^r}^m$ into a signature scheme using the standard hash-and-sign paradigm. The resulting key generation, signature generation and verification algorithms are described in the next few sections.

A large part of the coefficients of $\mathcal{P}$ is generated from a seed. This seed is included in the public key and replaces all the generated coefficients to make the public key much smaller. In order to reduce the size of the secret key we do not store $\mathcal{F}$ nor $\mathcal{T}$. Instead, we only store a private seed that was used to generate the public seed and $\mathcal{T}$.

The LUOV scheme can be used in two modes. One option is the usual appended signature mode where a message is authenticated by appending a signature. A different option is the message recovery mode, which can be used to reduce the size of a message-signature pair. In message recovery mode (part of) the message is not transmitted but recovered from the signature.

2.2 Relation to the UOV scheme

The LUOV scheme is an adaptation of the Unbalanced Oil and Vinegar signature scheme. It differs from the original UOV scheme in a number of ways. The first modification, due to Petzoldt [18], changes the key generation algorithm to make it possible to choose a large part of the public key. One can then choose this part to correspond with the output of a pseudo-random number generator and replace a large part of the public key by a seed. This modified key generation algorithm still produces the same distribution of key pairs (and hence this modification does not affect the security of the scheme), assuming that the output of the PRNG (we have used SHAKE128 or Chacha8) is indistinguishable from true randomness.

A second modification is that a public key $\mathcal{P} : \mathbb{F}_2^n \to \mathbb{F}_2^m$ for the UOV scheme over $\mathbb{F}_2$ is used as a public key for the UOV scheme over a large extension field $\mathbb{F}_{2^r}$. The public key is 'lifted' to the extension field by just extending the polynomial map $\mathcal{P}$ to a map from $\mathbb{F}_{2^r}^n$ to $\mathbb{F}_{2^r}^m$. This is were the Lifted UOV scheme gets its name from. The advantage of this approach is that the public key remains small (since the coefficients of the public key are 0 or 1), while solving the system $\mathcal{P}(x) = y$ for some y in $\mathbb{F}_{2^r}^m$ becomes more difficult compared to the case where y is in $\mathbb{F}_2^m$. This adaptation is due to Beullens and Preneel. [5].

Thirdly, the linear map $\mathcal{T}$ is chosen to have a matrix representation of the form

$$\begin{pmatrix} \mathbf{1}_v & \mathbf{T} \\ 0 & \mathbf{1}_m \end{pmatrix},$$

where $\mathbf{T}$ is a v-by-m matrix. This choice makes the key generation algorithm and the signing algorithm much faster, but does not affect the security of the scheme because for a random public key there exists an equivalent private key with $\mathcal{T}$ of this form with high probability [21]. This implies that if there is an attack against the modified signature scheme, the same attack would work on nearly all public keys of the original UOV scheme. This choice of $\mathcal{T}$ was first proposed by Czypek [7], where it was used to speed up the key generation algorithm. LUOV makes the same choice of $\mathcal{T}$, but uses different key generation and signature generation algorithms that are even faster.

2.3 Parameter space

The parameters for the LUOV algorithm are :

- r — The degree of the field extension $\mathbb{F}_2 \subset \mathbb{F}_{2^r}$.
- m — The number of polynomials in the public key, also the number of oil variables.
- v — The number of vinegar variables.
- $n = m + v$ — The total number of variables
- $\mathcal{H}$ — The extendable output function that is used for hashing the message and deriving the public key, either SHAKE128 or SHAKE256.
- $\mathcal{G}$ — The extendable output function that is used for generating the public map from a public seed, either SHAKE128 or ChaCha8.

2.4 Key Generation Algorithm

The key generation algorithm (Alg. 4) first uses a private seed to pseudo-randomly generate a seed that will be published, as well as the v-by-m matrix that determines the linear map $\mathcal{T}$. Then, the public seed is used to generate $\mathbf{C} \in \mathbb{F}_2^m$, the constant part of the public map $\mathcal{P}$, $\mathbf{L} \in \mathbb{F}_2^{m \times n}$, the linear part of $\mathcal{P}$ and $\mathbf{Q}_1 \in \mathbb{F}_2^{m \times \left(\frac{v(v+1)}{2} + vm\right)}$, the first $\frac{v(v+1)}{2} + vm$ columns of the Macaulay matrix of the quadratic part of $\mathcal{P}$ in the lexicographic ordering. Then $\mathbf{Q}_2 \in \mathbb{F}_2^{m \times \frac{m(m+1)}{2}}$, the remaining part of the Macaulay matrix of the quadratic part of $\mathcal{P}$ is calculated (see Sect. 2.4.1). The public key consists of the public seed and $\mathbf{Q}_2$. The private key is simply the seed that was used as input for the key generation algorithm. The details of how the different objects are sampled from the XOF are described in Sect. 2.9.

2.4.1 Finding the remaining coefficients of $\mathcal{P}$

For each polynomial p_k in the public map $\mathcal{P}$ there is a uniquely determined upper triangular matrix $\mathbf{P}_k \in \mathbb{F}_2^{n \times n}$, such that $\mathbf{x}^\top \mathbf{P}_k \mathbf{x}$ is equal to the evaluation of the quadratic part of p_k at $\mathbf{x}$. The matrix corresponding to the polynomial f_k in the secret map $\mathcal{F}$ is then, up to the addition of a skew-symmetric matrix, equal to

$$\begin{pmatrix} \mathbf{1}_v & \mathbf{0} \\ -\mathbf{T}^\top & \mathbf{1}_m \end{pmatrix} \begin{pmatrix} \mathbf{P}_{k,1} & \mathbf{P}_{k,2} \\ \mathbf{0} & \mathbf{P}_{k,3} \end{pmatrix} \begin{pmatrix} \mathbf{1}_v & -\mathbf{T} \\ \mathbf{0} & \mathbf{1}_m \end{pmatrix} = \begin{pmatrix} \mathbf{P}_{k,1} & -\mathbf{P}_{k,1}\mathbf{T} + \mathbf{P}_{k,2} \\ -\mathbf{T}^\top \mathbf{P}_{k,1} & \mathbf{T}^\top \mathbf{P}_{k,1} \mathbf{T} - \mathbf{T}^\top \mathbf{P}_{k,2} + \mathbf{P}_{k,3} \end{pmatrix},$$

where we have split up the matrix $\mathbf{P}_k$, into $\mathbf{P}_{k,1} \in \mathbb{F}_2^{v\times v}$, $\mathbf{P}_{k,2} \in \mathbb{F}_2^{v\times m}$ and $\mathbf{P}_{k,3} \in \mathbb{F}_2^{m\times m}$. The terms of f_k that are quadratic in the oil variables have to vanish, so

$$\mathbf{P}_{k,3} = -\mathbf{T}^\top \mathbf{P}_{k,1}\mathbf{T} + \mathbf{T}^\top \mathbf{P}_{k,2}\,,$$

up to the addition of a skew-symmetric matrix. This formula completely determines the upper triangular matrix $\mathbf{P}_{k,3}$. The entries of the $\mathbf{P}_{k,1}$ and $\mathbf{P}_{k,2}$ are generated from the public seed and the matrix $\mathbf{T}$ is known, so the matrices $\mathbf{P}_{k,3}$ can easily be computed. The entries of the matrices $\mathbf{P}_{k,3}$ are then arranged in the Macaulay matrix $\mathbf{Q}_2$. A detailed implementation of this procedure is shown in Alg. 3.

Algorithm findPk1

input: k — An integer between 1 and m.
$\mathbf{Q}_1$ — First part of Macaulay matrix of the quadratic part of $\mathcal{P}$

output: $\mathbf{P}_{k,1}$ — The v-by-v matrix representing the part of p_k that is quadratic in the vinegar variables.

```
$\mathbf{P}_{k,1} \leftarrow \mathbf{0}_v$
column $\leftarrow 1$
for $i$ from 1 to $v$ do
    for $j$ from $i$ to $v$ do
        $\mathbf{P}_{k,1}$[i,j] $\leftarrow \mathbf{Q}_1[k,$ column$]$
        column $\leftarrow$ column $+ 1$                  ▷ move to the next term
    end for
    column $\leftarrow$ column $+ m$          ▷ Skip the terms $x_i x_{v+1}$ up to $x_i x_{v+m}$
end for
return $\mathbf{P}_{k,1}$
```

Alg. 1: Algorithm for reading $\mathbf{P}_{k,1}$ from $\mathbf{Q}_1$.

2.5 Signature Generation Algorithm

The signature generation algorithm first generates $\mathbf{C}, \mathbf{L}, \mathbf{Q}_1$ and $\mathbf{T}$ from the private seed in the same way as the key generation algorithm. Then, it calculates $\mathbf{h}$, the hash digest of the message that will be signed, concatenated with a zero byte and with a random 16-byte salt. Concatenating the message with zero is done to make signatures generated in appended signature mode unrelated to signatures generated in message recovery mode (see Sect. 2.7). Then, the algorithm produces a signature in two steps. First, the special structure of $\mathcal{F}$ is exploited to produce a solution $\mathbf{s}'$ to the equation $\mathcal{F}(\mathbf{s}') = \mathbf{h}$. Then, the signature $\mathbf{s}$ is calculated as

$$\mathbf{s} = \begin{pmatrix} \mathbf{1}_v & -\mathbf{T} \\ \mathbf{0} & \mathbf{1}_m \end{pmatrix} \mathbf{s}'\,.$$

Solving $\mathcal{F}(\mathbf{s}') = \mathbf{h}$ is done by repeatedly substituting randomly generated values into the vinegar variables and trying to solve the resulting linear system until a unique solution

```
Algorithm findPk2
input: $k$ — An integer between 1 and $m$.
       $\mathbf{Q}_1$ — First part of Macaulay matrix of quadratic part of $\mathcal{P}$
output: $\mathbf{P}_{k,2}$ — The $v$-by-$m$ matrix representing the part of $p_k$ that is bilinear in
        the vinegar variables and the oil variables.

$\mathbf{P}_{k,2} \leftarrow \mathbf{0}_{v\times m}$
column $\leftarrow 1$
for $i$ from 1 to $v$ do
    column $\leftarrow$ column $+ v - i + 1$                ▷ Skip terms $x_i^2$ up to $x_i x_v$
    for $j$ from 1 to $m$ do
        $\mathbf{P}_{k,2}$[i,j] $\leftarrow \mathbf{Q}_1[k,$ column$]$
        column $\leftarrow$ column $+ 1$                     ▷ Move to the next term
    end for
end for
return $\mathbf{P}_{k,2}$
```

Alg. 2: Algorithm for reading $\mathbf{P}_{k,2}$ from $\mathbf{Q}_1$.

```
Algorithm findQ2
input: $\mathbf{Q}_1$ — First part of Macaulay matrix of quadratic part of $\mathcal{P}$
       $\mathbf{T}$ — A $v$-by-$m$ matrix
output: $\mathbf{Q}_2$ — The second part of Macaulay matrix for quadratic part of $\mathcal{P}$

$\mathbf{Q}_2 \leftarrow 0_{m\times D_2}$
for $k$ from 1 to $m$ do
    $\mathbf{P}_{k,1} \leftarrow$findPk1$(k, \mathbf{Q}_1)$
    $\mathbf{P}_{k,2} \leftarrow$findPk2$(k, \mathbf{Q}_1)$
    $\mathbf{P}_{k,3} \leftarrow -\mathbf{T}^\top \mathbf{P}_{k,1} \mathbf{T} + \mathbf{T}^\top \mathbf{P}_{k,2}$    ▷ Compute $\mathbf{P}_{k,3}$ up to skew-symmetric matrix
    column $\leftarrow 1$
    for $i$ from 1 to $m$ do                                  ▷ Read off $\mathbf{Q}_2$
        $\mathbf{Q}_2[k,$ column$] \leftarrow \mathbf{P}_{k,3}[i,i]$
        column $\leftarrow$ column $+ 1$
        for $j$ from $i+1$ to $m$ do
            $\mathbf{Q}_2[k,$ column$] \leftarrow \mathbf{P}_{k,3}[i,j] + \mathbf{P}_{k,3}[j,i]$
            column $\leftarrow$ column $+ 1$
        end for
    end for
end for
return $\mathbf{Q}_2$
```

Alg. 3: Algorithm for determining $\mathbf{Q}_2$ from $\mathbf{Q}_1$ and $\mathbf{T}$.

```
Algorithm KeyGen
input: private_seed — seed to generate a key-pair
output: (public_seed, $\mathbf{Q}_2$) — A public key
        private_seed — A corresponding private key

private_sponge ← InitializeAndAbsorb(private_seed)
public_seed ← SqueezePublicSeed(private_sponge)
$\mathbf{T}$ ← SqueezeT(private_sponge)
public_sponge ← InitializeAndAbsorb(public_seed)
$\mathbf{C}, \mathbf{L}, \mathbf{Q}_1$ ← SqueezePublicMap (public_sponge)
$\mathbf{Q}_2$ ← FindQ2($\mathbf{Q}_1, \mathbf{T}$)
return (public_seed, $\mathbf{Q}_2$) and private_seed
```

Alg. 4: The key generation algorithm

is found. A unique solution is almost always found on the first try, the probability of failing being roughly 2^{-r}. For a particular assignment to the vinegar variables $\mathbf{v} \in \mathbb{F}_{2^r}^v$, the augmented matrix for the linear system $\mathcal{F}((\mathbf{v}||\mathbf{o})^\top) = \mathbf{h}$ can be derived as in Alg. 5. This algorithm relies on the fact that after fixing the vinegar variables to $\mathbf{v}$, the map $\mathcal{F}$ is a linear map with constant part

$$\mathbf{C} + \mathbf{L}\begin{pmatrix}\mathbf{v}\\ \mathbf{0}\end{pmatrix} + \begin{pmatrix}\mathbf{v}^\top \mathbf{P}_{1,1}\mathbf{v}\\ \cdots \\ \mathbf{v}^\top \mathbf{P}_{m,1}\mathbf{v}\end{pmatrix},$$

and a linear part with the matrix representation

$$\mathbf{L}\begin{pmatrix}-\mathbf{T}\\ \mathbf{1}_m\end{pmatrix} + \begin{pmatrix}\mathbf{v}^\top[(\mathbf{P}_{1,1} + \mathbf{P}_{1,1}^\top)\mathbf{T} + \mathbf{P}_{1,2}]\\ \cdots \\ \mathbf{v}^\top[(\mathbf{P}_{m,1} + \mathbf{P}_{m,1}^\top)\mathbf{T} + \mathbf{P}_{m,2}]\end{pmatrix}.$$

Pseudocode for the signature generation algorithm is provided in Alg. 6.

2.6 Signature Verification Algorithm

First, the signature verification algorithm uses the public seed to generate $\mathbf{C}, \mathbf{L}$ and $\mathbf{Q}_1$. Together with $\mathbf{Q}_2$, which is included in the public key, this completely determines the public map $\mathcal{P}$. To verify a signature $\mathbf{s}$ for a message M, the verification algorithm simply checks whether $\mathcal{P}(\mathbf{s})$ is equal to the rm-bit long hash digest of the message M, appended with `0x00` and a 128-bit salt. Pseudocode for this algorithm is provided in Alg. 9.

2.7 Signatures with message recovery

It is possible to use the signature scheme in a message recovery mode. Whether or not message recovery is used does not affect the key generation algorithm. The same key pair

```
Algorithm BuildAugmentedMatrix

input: $\mathbf{C} \in \mathbb{F}_{2^r}^{m}$ — The constant part of the public map $\mathcal{P}$
       $\mathbf{L} \in \mathbb{F}_{2^r}^{m \times n}$ — The linear part of $\mathcal{P}$
       $\mathbf{Q}_1 \in \mathbb{F}_{2^r}^{m \times \frac{v(v+1)}{2} + vm}$ — The first part of quadratic part of $\mathcal{P}$
       $\mathbf{T} \in \mathbb{F}_{2}^{v \times m}$ — The matrix that determines the linear transformation $\mathcal{T}$.
       $\mathbf{h} \in \mathbb{F}_{2^r}^{m}$ — The hash digest to target.
       $\mathbf{v} \in \mathbb{F}_{2^r}^{v}$ — An assignment to the vinegar variables.

output: $\mathbf{LHS}||\mathbf{RHS} \in \mathbb{F}_{2^r}^{m \times m+1}$ — The augmented matrix for $\mathcal{F}(\mathbf{v}||\mathbf{o}) = \mathbf{h}$

$\mathbf{RHS} \leftarrow \mathbf{h} - \mathbf{C} - \mathbf{L}(\mathbf{v}||0)^{\top}$          ▷ Right hand side of linear system
$\mathbf{LHS} \leftarrow \mathbf{L} \begin{pmatrix} -\mathbf{T} \\ \mathbf{1}_m \end{pmatrix}$          ▷ Left hand side of linear system
for $k$ from 1 to $m$ do
    $\mathbf{P}_{k,1} \leftarrow \text{findPk1}(k, \mathbf{Q}_1)$
    $\mathbf{P}_{k,2} \leftarrow \text{findPk2}(k, \mathbf{Q}_1)$
    $\mathbf{RHS}[k] \leftarrow \mathbf{RHS}[k] - \mathbf{v}^{\top}\mathbf{P}_{k,1}\mathbf{v}$          ▷ evaluation of terms of $f_k$ that are
                                                                      quadratic in vinegar variables
    $\mathbf{F}_{k,2} \leftarrow -(\mathbf{P}_{k,1} + \mathbf{P}_{k,1}^{\top})\mathbf{T} + \mathbf{P}_{k,2}$          ▷ Terms of $f_k$ that are bilinear in the
                                                                      vinegar and the oil variables
    $\mathbf{LHS}[k] \leftarrow \mathbf{LHS}[k] + \mathbf{v}\mathbf{F}_{k,2}$          ▷ Insert row in the left hand side
end for
return $\mathbf{LHS}||\mathbf{RHS}$
```

Alg. 5: Builds the augmented matrix for the linear system $\mathcal{P}(\mathbf{v}||\mathbf{o}) = \mathbf{h}$ after fixing the vinegar variables.

```
Algorithm Sign
input: $\mathsf{private\_seed}$ — A private key
       $M$ — A message to sign
output: $(\mathbf{s}, \mathsf{salt})$ — A signature for the message $M$

$\mathsf{public\_seed}||\mathbf{T} \leftarrow \mathcal{H}(\mathsf{private\_seed})$          ▷ See Sect. 2.9.1
$\mathbf{C}||\mathbf{L}||\mathbf{Q}_1 \leftarrow \mathcal{G}(\mathsf{public\_seed})$          ▷ See Sect. 2.9.3
$\mathsf{salt} \leftarrow$RandomBytes(16)
$\mathbf{h} \leftarrow \mathcal{H}(M||\mathsf{0x00}||\mathsf{salt})$          ▷ See Sect. 2.9.2
while No solution $\mathbf{s}'$ to the system $\mathcal{F}(\mathbf{s}') = \mathbf{h}$ is found do
    $\mathbf{v} \leftarrow$ RandomBytes $(rv/8)$
    $\mathbf{A} \leftarrow$ BuildAugmentedMatrix $(\mathbf{C}, \mathbf{L}, \mathbf{Q}_1, \mathbf{T}, \mathbf{h}, \mathbf{v})$
                    ▷ Build the augmented matrix for the linear system $\mathcal{F}(\mathbf{v}||\mathbf{o}) = \mathbf{h}$
    GaussianElimination$(\mathbf{A})$
    if $\mathcal{F}(\mathbf{v}||\mathbf{o}) = \mathbf{h}$ has a unique solution $\mathbf{o}$ then
        $\mathbf{s}' \leftarrow (\mathbf{v}||\mathbf{o})^\top$
    end if
end while
$\mathbf{s} \leftarrow \begin{pmatrix} \mathbf{1}_v & -\mathbf{T} \\ \mathbf{0} & \mathbf{1}_m \end{pmatrix} \mathbf{s}'$
return $(\mathbf{s}, \mathsf{salt})$
```

Alg. 6: The signature generation algorithm

```
Algorithm EvaluatePublicMap
input: $(\mathsf{public\_seed}, \mathbf{Q}_2)$ — A public key
       $\mathbf{s}$ — A candidate–signature
output: The evaluation of $\mathcal{P}$ at $\mathbf{s}$

$\mathbf{C}||\mathbf{L}||\mathbf{Q}_1 \leftarrow \mathcal{G}(\mathsf{public\_seed})$          ▷ See Sect. 2.9.3
$\mathbf{Q} \leftarrow \mathbf{Q}_1||\mathbf{Q}_2$
$\mathbf{e} \leftarrow \mathbf{C} + \mathbf{L}\mathbf{s}$          ▷ Evaluate constant and linear part of $\mathcal{P}$ at $\mathbf{s}$
$\mathsf{column} \leftarrow 1$
for $i$ from 1 to $n$ do          ▷ Evaluate quadratic parts of $\mathcal{P}$ at $\mathbf{s}$
    for $j$ from $i$ to $n$ do
        for $k$ from 1 to $m$ do
            $\mathbf{e}[k] \leftarrow \mathbf{e}[k] + \mathbf{Q}[k, \mathsf{column}]\mathbf{s}[i]\mathbf{s}[j]$          ▷ Evaluate terms in $x_i x_j$
        end for
        $\mathsf{column} \leftarrow \mathsf{column} + 1$
    end for
end for
return $\mathbf{e}$
```

Alg. 7: The algorithm for evaluating the public map at a point

```
Algorithm Verify
input: $(\mathsf{public\_seed}, \mathbf{Q}_2)$ — A public key
       $M$ — A message
       $(\mathbf{s}, \mathsf{salt})$ — A candidate–signature
output: Accept if $\mathbf{s}$ is a valid signature for $M$, Reject otherwise
h ← $\mathcal{H}(M||\mathtt{0x00}||\mathsf{salt})$                ▷ See Sect. 2.9.2
e ←EvaluatePublicMap($(\mathsf{public\_seed}, \mathbf{Q}_2), \mathbf{s}$)
if  e = h  then                                   ▷ Check if $\mathcal{P}(\mathbf{s}) = \mathbf{h}$
    return Accept
else
    return Reject
end if
```

Alg. 8: The signature verification algorithm in appended signature mode

can be used to sign messages in message recovery mode and in appended signature mode, a signature for M in appended signature mode is unrelated to a signature for the same message in message recovery mode, because a different byte is appended to the message in each mode. The signing algorithm in message recovery mode differs from the signing algorithm in appended signature mode (Alg. 6) because the message is padded with `0x01` instead of `0x00` in lines 6 and 8. Furthermore, the procedure to determine the target of the public map is altered to make message recovery possible. In the appended signature mode, the target was determined by interpreting the rm bit long output of a SHAKE function as a vector of m elements of $\mathbb{F}_{2^r}$. In message recovery mode, the target is obtained by interpreting

$$\mathrm{SHAKE}(M||\mathtt{0x01}||\mathsf{salt}, l_1)||\,[\mathrm{SHAKE}(\mathrm{SHAKE}(M||\mathtt{0x01}||\mathsf{salt}, l_1), l_2) \oplus M']$$

as a vector of m elements in $\mathbb{F}_{2^r}$, where l_1 is equal to 256, 384, 512, for security levels 2, 4 and 5 respectively, and l_2 is equal to $rm - l_1$. The part of the message M' is formed by taking the last $\lfloor (l_2 - 1)/8 \rfloor$ bytes of the message M, appending with a `1` and enough `0`'s to make M' l_2 bits long.

The signature verification algorithm evaluates the public map $\mathcal{P}$ at the signature $\mathbf{s}$, and interprets the output as a sequence first_bytes of l_1 bytes, concatenated with a sequence last_bytes of l_2 bytes. The signature verification algorithm recovers up to $\lfloor (l_2 - 1)/8 \rfloor$ bytes of the message M, by calculating

$$M' = \mathsf{last\_bytes} \oplus \mathrm{SHAKE}(\mathsf{first\_bytes}, l_2)$$

and removing the padding. The signature is accepted if first_bytes is equal to $\mathrm{SHAKE}(M||\mathtt{0x01}, l_1)$.

Algorithm Verify

```
input: $(\mathsf{public\_seed}, \mathbf{Q}_2)$ — A public key
       $M$ — The first part of a message (possibly the empty string)
       $(\mathbf{s}, \mathsf{salt})$ — A candidate–signature
output: The full message $M$ if $\mathbf{s}$ is a valid signature, Reject otherwise

e ←EvaluatePublicMap((public_seed, $\mathbf{Q}_2$), s)
first_bytes, last_bytes ←Enc(e)        ▷ Split e into $l_1$ and $l_2$ bytes
padded_message ← last_bytes ⊕ $\mathcal{H}$(first_bytes, $l_2$)
if padded_message is not properly padded then
    return Reject    ▷ Reject if padded_message doesn't end in 0x01 0x00 ··· 0x00
end if
$M \leftarrow M$||RemovePadding(padded_message)
hash_digest ← $\mathcal{H}(M$||0x01||salt, $l_1$)
if first_bytes = hash_digest then
    return $M$
else
    return Reject
end if
```

Alg. 9: The signature verification algorithm in message recovery mode

2.8 Encoding of objects

2.8.1 Encoding of private key

A private key for the LUOV signature scheme is a sequence of 256 random bits (used as input to $\mathcal{H}$) and is simply encoded as a sequence of 32 bytes.

2.8.2 Encoding of public key

A public key of the LUOV signature scheme consists of a sequence of 32 bytes (which are used as input to $\mathcal{G}$) and an m-by-$m(m+1)/2$ matrix with binary entries. The matrix is encoded by concatenating the columns and padding the result with zero bits to get a sequence of bits of length divisible by 8. Then, the sequence is interpreted as a sequence of bytes, where the first bits have the least significant values. The encoding of a public keys is $32 + \lceil \frac{m^2(m+1)}{2} \frac{1}{8} \rceil$ bytes large.

Example. *For a parameter set with $m = 3$, the public key could contain the matrix*

$$\mathbf{Q}_2 = \begin{pmatrix} 010111 \\ 111001 \\ 000101 \end{pmatrix} .$$

Concatenating its columns gives 010110010101100111, which results in the 3 bytes (01011001) (01011001) (11000000), so

$$\mathsf{Enc}(\texttt{0x36}\ \cdots\ \texttt{0x5d}, \mathbf{Q}_2) = \underbrace{\texttt{0x36}\ \cdots\ \texttt{0x5d}}_{\textit{32-byte Public seed}}\ \underbrace{\texttt{0x9a}\,\texttt{0x9a}\,\texttt{0x03}}_{\mathbf{Q}_2}\ .$$

2.8.3 Encoding of signature

A signature of the UOV signature scheme consists of a vector $\mathbf{s} \in \mathbb{F}_{2^r}^n$ of $n = v + m$ field elements and a 16-byte random salt. The encoding of the signature consists of the concatenation of the encodings of these n field elements (see below for the details on how field elements are encoded), padded with zeros to fill the last byte if $n * r$ is not divisible by 8. This is then concatenated with the 16-byte random salt. The encoding of a signature is thus $\lceil \frac{nr}{8} \rceil + 16$ bytes large.

$$\mathsf{Enc}(\mathbf{s}) = \mathsf{Enc}(\mathbf{s}[0])||\cdots||\mathsf{Enc}(\mathbf{s}[n-1])||\mathsf{padding\_zeros}||\mathsf{salt}$$

The finite fields that are used by the various instantiations of the LUOV signature scheme are $\mathbb{F}_{2^7}, \mathbb{F}_{2^{47}}, \mathbb{F}_{2^{61}}$ and $\mathbb{F}_{2^{79}}$.

Encoding field elements. Elements in the field $\mathbb{F}_{2^r}$ are represented as binary polynomials modulo an irreducible polynomial $f_r = x^r + \cdots + 1$ (See Table 1). The choice of irreducible polynomial does not affect the security of the scheme, so we have chosen sparse polynomials for efficiency. An element of $\mathbb{F}_{2^r}$ is encoded as the r-bit sequence obtained by concatenating its coefficients, where the least significant bits correspond to the lowest degree terms.

Example.

$$\begin{aligned}
\mathsf{Enc}(1 \in \mathbb{F}_{2^7}) &= \mathsf{0000001} \\
\mathsf{Enc}(x + x^5 + x^6 \in \mathbb{F}_{2^7}) &= \mathsf{1100010} \\
\mathsf{Enc}(x^{46} \in \mathbb{F}_{2^{47}}) &= \mathsf{10000000000000000000000000000000000000000000000}
\end{aligned}$$

Table 1: Irreducible polynomials used for representing finite fields.

Finite Field	Irreducible polynomial in $\mathbb{F}_2[x]$
$\mathbb{F}_{2^7}$	$x^7 + x + 1$
$\mathbb{F}_{2^{47}}$	$x^{47} + x^5 + 1$
$\mathbb{F}_{2^{61}}$	$x^{61} + x^5 + x^2 + x + 1$
$\mathbb{F}_{2^{79}}$	$x^{79} + x^9 + 1$

2.9 Sampling objects from PRNGs

The LUOV signature scheme uses 2 extendable output functions $\mathcal{H}$ and $\mathcal{G}$ to provide pseudorandom bit-streams. The first one, $\mathcal{H}$ is used to generate:

- public_seed — The public seed used to generate a large part of the public map $\mathcal{P}$.
- $\mathbf{T}$ — The matrix that determines the linear transformation that hides the UOV structure of the secret map $\mathcal{F}$.
- $\mathbf{h}$ — The hash digest of a message.

The function $\mathcal{H}$ has to be collision resistant because collisions in $\mathbf{h}$ break the EUF-CMA property of the signature scheme. It is evaluated on secret inputs, so the implementation of this function has to be side-channel secure. For $\mathcal{H}$ we use SHAKE128 or SHAKE256, depending on the security level. In contrast, the only purpose of $\mathcal{G}$ is to expand a public seed to generate a big part of the coefficients of the public map $\mathcal{P}$. The function $\mathcal{G}$ is not required to have strong cryptographic properties such as collision resistance or preimage resistance, it only has to generate public maps that are "random enough" so that there are no special properties of the output that can be used to launch an attack. Moreover since the input and output of this function is public, its implementation does not need to be protected against side-channel attacks.

2.9.1 Generating public seed and T

To generate the public seed one evaluates $\mathcal{H}(\mathsf{private\_seed}, 32 + \lceil \frac{m}{8} \rceil v)$. The public seed then consists of the first 32 bytes of the output. The matrix $\mathbf{T} \in \mathbb{F}_2^{v \times m}$ is represented by the remaining $\lceil \frac{m}{8} \rceil v$ bytes as follows. The bytes $(i-1)\lceil \frac{m}{8} \rceil + 1$ up to $i\lceil \frac{m}{8} \rceil$ are the i-th row of $\mathbf{T}$. If m is not divisible by 8, the most significant bits of the last byte of each row (i.e. $i\lceil \frac{m}{8} \rceil$-th bytes in the sequence) are ignored.

Example. *Suppose $m = 3$, $v = 4$ and the following 32+4 bytes are squeezed from the Keccak sponge :*

$$\underbrace{\texttt{0x36} \cdots \texttt{0x5d}}_{\textit{32-byte Public seed}} \ \texttt{0x49 0xa2 0x86 0x4d} \ .$$

Then, the matrix $\mathbf{T} \in \mathbf{F}_2^{v \times m}$ is equal to

$$\begin{pmatrix} 001 \\ 010 \\ 110 \\ 101 \end{pmatrix} .$$

2.9.2 Generating hash digest

The hash digest is a vector over $\mathbb{F}_{2^r}$ of length m. It is obtained by interpreting the mr bits of output of $\mathcal{H}(M||\texttt{0x00}||\textsf{salt}, mr)$ as the encoding of m elements of $\mathbb{F}_{2^r}$.

2.9.3 Generating most part of the public map

The matrices $\mathbf{C} \in \mathbb{F}_2^{m\times 1}, \mathbf{L} \in \mathbb{F}_2^{m\times n}$ and $\mathbf{Q}_1 \in \mathbb{F}_2^{m\times(\frac{v(v+1)}{2}+vm)}$ are obtained from $\lceil \frac{m}{16} \rceil$ calls to $\mathcal{G}$. The first 16 rows of $\mathbf{C}, \mathbf{L}$ and $\mathbf{Q}_1$ are obtained from $\mathcal{G}(\textsf{public\_seed}||\texttt{0x00}, 2(1+n+\frac{v(v+1)}{2}+vm))$. The first 2 bytes represent the first 16 rows of $\mathbf{C}$, the next $2n$ bytes represent the first 16 rows of $\mathbf{L}$ column by column. Lastly the remaining $2(\frac{v(v+1)}{2} + vm)$ bytes represent the first 16 rows of $\mathbf{Q}_1$ column by column. In the same way the rows with index $16i$ to $16i + 15$ are generated from the output of $\mathcal{G}(\textsf{public\_seed}||\texttt{0x0i}, 2(1 + n + \frac{v(v+1)}{2} + vm))$. If m is not divisible by 16, the most significant bits of each column are ignored.

2.9.4 Option to use ChaCha8

We allow ChaCha8 as an alternative to SHAKE128 for generating the public map. Instead of using the output streams of SHAKE($\textsf{public\_seed}||\texttt{0x0i}$) for i in $\{0, \cdots, \lceil \frac{m}{16} \rceil - 1\}$ we allow to use the keysteam of ChaCha8 with $\textsf{public\_seed}$ used as key and i (padded with zeros to 64 bits) as nonce.

3 List of parameter sets (part of 2.B.1)

We define two sets of parameter choices. The first set aims to provide small signatures, which is suitable for applications where many signatures are communicated. The second set of parameters aims to minimize the combined cost of a signature and a public key and is more suitable when the signatures and the public key are both communicated, such as a chain of signatures anchored to a root certificate authority.

Table 2: Different parameter choices for the LUOV signature scheme. The first 3 choices provide small signatures, the last three choices give small public keys at the cost of larger signatures.

	claimed security level	r	m	v	SHAKE	$\|\mathsf{sig}\|$	$\|\mathsf{pk}\|$	$\|\mathsf{sk}\|$	message recovery (optional)
LUOV-7-57-197	lvl 1	7	57	197	128	239 B	11.5 KB	32B	17 B
LUOV-7-83-283	lvl 3	7	83	283	256	337 B	35.4 KB	32B	24 B
LUOV-7-110-374	lvl 5	7	110	374	256	440 B	82.0 KB	32B	32 B
LUOV-47-42-182	lvl 1	47	42	182	128	1332 B	4.7 KB	32B	214 B
LUOV-61-60-261	lvl 3	61	60	261	256	2464 B	13.4 KB	32B	409 B
LUOV-79-76-341	lvl 5	79	76	341	256	4134 B	27.2 KB	32B	686 B

4 Detailed performance analysis (2.B.2)

4.1 Time

The average number of cycles consumed by the different algorithms and implementations are reported in Table 3 (non-constant time generic optimized implementations) and Table 4 (constant time AVX2 optimized implementation). The measurements are made in appended signature mode, but there is no noticeable difference between the cycle count in appended signature mode and in message recovery mode. The constant timeness of the AVX2 optimized implementation is verified with Valgrind to check that no branching depends on secret data, or that no momory at secret indices is accessed. This test fails at a single point of the implementation, when it is checked that the derived linear system is uniquely solvable or not (which happens at the very end of the constant time Gaussian reduction algorithm). This leaks the number of signing attempts that was made, but this does leak secret data. We also verified the constant timeness with the dudect tool [19], which does not detect any leakage.

Table 3: Cycle counts of generic optimized implementation. The reported values are the average of 1000 executions compiled with "`gcc − O3`" with gcc version 7.3.0. Benchmarking is performed on an ASUS S410 Notebook PC with an Intel Core i5-8250U CPU @ 1.60 GHz. The memory usage is measured with the valgrind tools drd (stack) and massif (heap).

	security level	PRNG	keygen (cycles)	sign (cycles)	verify (cycles)	Memory usage stack	 heap
LUOV-7-57-197	1	Keccak	13 M	4.4 M	3.4 M	12 KB	41 KB
		Chacha8	11 M	3.3 M	2.3 M	12 KB	41 KB
LUOV-7-83-283	3	Keccak	60 M	14 M	10 M	13 KB	151 KB
		Chacha8	56 M	9.5 M	5.9 M	12 KB	151 KB
LUOV-7-110-374	5	Keccak	136 M	28 M	21 M	15 KB	285 KB
		Chacha8	124 M	19 M	12 M	14 KB	285 KB
LUOV-47-42-182	1	Keccak	7.6 M	12 M	9.1 M	12 KB	90 KB
		Chacha8	6.7 M	11 M	8.2 M	12 KB	90 KB
LUOV-61-60-261	3	Keccak	20 M	37 M	26 M	13 KB	177 KB
		Chacha8	18 M	34 M	29 M	12 KB	177 KB
LUOV-79-76-341	5	Keccak	68 M	163 M	83 M	22 KB	551 KB
		Chacha8	63 M	155 M	78 M	21 KB	551 KB

Table 4: Cycle counts of constant time AVX2 optimized implementation. The reported values are the average of 10000 executions compiled with "`gcc − O3`" with gcc version 7.3.0. Benchmarking is performed on an ASUS S410 Notebook PC with an Intel Core i5-8250U CPU @ 1.60 GHz.

	security level	PRNG	keygen (cycles)	sign (cycles)	verify (cycles)
LUOV-7-57-197	1	Keccak	1.9 M	1.4 M	1.0 M
		Chacha8	1.1M	515 K	197 K
LUOV-7-83-283	3	Keccak	6.2 M	3.2 M	2.8 M
		Chacha8	4.6 M	1.6 M	823 K
LUOV-7-110-374	5	Keccak	12.5 M	5.8 M	4.3 M
		Chacha8	9.7 M	3.2 M	1.6 M

4.2 Space

For all parameter choices, the secret key consists of a **32-byte** seed.

The public key consists of a 32 byte public seed, and the remaining $\frac{m^2(m+1)}{2}$ coefficients of the public map $\mathcal{P}$. This makes a total of $\mathbf{32} + \lceil \frac{\mathbf{m^2(m+1)}}{\mathbf{16}} \rceil$ **bytes**. If message recovery is used, the messages can be shortened by roughly 15% of the signature size.

A signature consists of $v+m$ elements of the field $\mathbb{F}_{2^r}$ and a 16 byte salt, good for a total of $\mathbf{16} + \lceil \frac{\mathbf{r(v+m)}}{\mathbf{8}} \rceil$ **bytes**.

The concrete sizes for the proposed parameter choices are displayed in Table 2. Even though the generic implementation did not try to optimize for memory usage we have included the memory usage measurements in Table 3. This includes the space required to store a key pair and a signature (which live on the heap).

4.3 How parameters affect performance

Table 3 shows that key generation is faster for the parameter sets with large extension fields. This is so because key generation benefits from the smaller polynomial systems, without paying the price of more complex field arithmetic, since key generation works in $\mathbb{F}_2$.

In contrast, in our implementation of the signing and verification algorithms, the smaller size of the polynomial systems does not make up for the increased complexity of the field arithmetic. Therefore, signing and verification is faster for the parameter sets with smaller field extensions.

The size of the public key is only impacted by the parameter m, and scales as $O(m^3)$, therefore to keep the public key small m should not be too large. By increasing r, the degree of the field extension $\mathbb{F}_2 \subset \mathbb{F}_{2^r}$, the required value of m to achieve a fixed security level decreases. However, increasing r also increases the size of the signatures. Therefore, it is possible to make a trade-off between small public keys (i.e. large r) or small signatures (i.e. small r). We propose two sets of parameter choices, one aiming at small signatures, the other aiming at small public keys. By varying the parameter r it is possible to interpolate between these parameter sets.

Example. *One might want a signature scheme that attains security level 1 with signatures as small as possible, subject to the condition that the public key is smaller than* $10KB$*. The best option from the proposed parameter sets would be LUOV-47-42-182, having signatures of* $1332B$ *and public keys of* $4.7KB$*. We can do better by adjusting the parameter* r*. For the choice* $r = 11$*, the python script that is included in the submission proposes the parameters* $m = 53, v = 193$*, resulting in signatures of* $338B$ *and public keys of just under* $10KB$*.*

4.4 Optimizations

4.4.1 Bit slicing

The i-th row of $\mathbf{Q}_2$ is calculated using only the data $\mathbf{T}$ and the i-th row of $\mathbf{Q}_1$ and this calculation is exacly the same for each row. This is an ideal situation for using bit slicing. The bits in the columns of $\mathbf{Q}_1$ and $\mathbf{Q}_2$ are packed into words and the computation is performed for all rows simultaneously. This greatly speeds up the key generation algorithm. This optimization is included in the reference implementation, because it does not affect the legibility of the code.

4.4.2 Precomputation on the secret and public keys.

With each signing and verification of a signature a lot of coefficients of the public map $\mathcal{P}$ have to be generated with the SHAKE128 or Chacha8 function. This takes up a majority of the signing and verification time. If enough memory is available (e.g. roughly 250 KB for the first parameter set) these coefficients of $\mathcal{P}$ can be precomputed and stored to speed up the verification of signatures. Similarly, the coefficients of the secret map $\mathcal{F}$ can be precomputed to speed up the signing algorithm. In the case of the first parameter set these precomputations speed up the signing algorithm by a factor 20 and the verification algorithm by a factor 69. In our AVX2 optimized implementation we provide 4 additional functions, besides the usual `keygen`, `sign` and `verify`. These are:

- `precompute_sk`: Which takes a secret key and produces a big secret key.
- `sign_fast`: Which takes a message to sign and a big secret key and produces a signed message.
- `precompute_pk`: Which takes a public key and produces a big public key.
- `verify_fast`: Which verifies a signed message given a big public key.

The performance of these functions is summarized in Table 5.

Table 5: Cycle counts of optimized implementation with precomputation of the keys. The reported values are the average of 10000 executions compiled with "`gcc − O3`" with gcc version 7.3.0. Benchmarking is performed on an ASUS S410 Notebook PC with an Intel Core i5-8250U CPU @ 1.60 GHz, with Turbo Boost disabled.

	security level	PRNG	precompute sk	precompute pk	sign	verify
LUOV-7-57-197	1	Keccak Chacha8	2.5 M 1.8 M	883 K 144 K	224 K	49 K
LUOV-7-83-283	3	Keccak Chacha8	8.7 M 7.3 M	2.1 M 666 K	643 K	152 K
LUOV-7-110-374	5	Keccak Chacha8	18.7 M 16.4 M	3.8 M 1.6 M	1.1 M	331 K

4.5 Precomputation for signing

In short, the signing algorithm consists of fixing random vinegar variables, constructing a linear system $A\mathbf{s} = \mathbf{y}$ and then solving it to find a valid signature $\mathbf{s}$. One observation here is that the matrix A of the linear system depends on the secret key and the choice of vinegar variables, but is independent of the message. This means that we can divide the signing algorithm into an offline phase in which we choose the vinegar variables, construct A and also compute its inverse and an online phase in which we only have to compute the right hand side $\mathbf{y}$ (which amounts to hashing the message), and multiplying this by A^{-1}. We have implemented this approach in the generic optimized implementation. We provide two additional functions:

- `sign_start`: Takes a secret key as input and outputs a partial signature.
- `sign_finish`: Takes a message and a partial signature as input and produces a signed message.

Remark. *Reusing a partial signature to sign multiple messages leaks (part of) the secret key, so the* `sign_finish` *function destroys the partial signature after using it.*

In the case of the first parameter set (7-57-197) and for signing 100 byte messages this approach reduces the online signing time by a factor 400 to just 10K cycles. At the same time the total cycle count of the offline + the online signing phases increases by roughly 60% (there is an increase because we now invert a matrix, rather than only solving a linear system). This is useful for situations where messages have to be signed with very low latencies. This method can also make LUOV signing feasible on very constrained devices, because the bulk of the signing can be done offline when, for example, the CPU is idle or when solar power is available. This offline-online approach was used before in the context of UOV [16], but it can not be used for general MQ schemes. Cycle counts and the sizes of partial signatures are displayed in Table 6. The implementation focuses on minimizing the online signing time, different trade-offs between online signing time and the size of a partial signature are possible.

Table 6: Cycle counts of generic optimized implementation with offline signing phase. The reported values are the average of 1000 executions compiled with "`gcc − O3`" with gcc version 7.3.0. Benchmarking is performed on an ASUS S410 Notebook PC with an Intel Core i5-8250U CPU @ 1.60 GHz.

	security level	PRNG	`sign_start` (cycles)	`sign_finish` (cycles)
LUOV-7-57-197	1	Keccak Chacha8	7.3 M 6.1 M	10 K
LUOV-7-83-283	3	Keccak Chacha8	21 M 17 M	16 K
LUOV-7-110-374	5	Keccak Chacha8	46 M 39 M	26 K
LUOV-47-42-182	1	Keccak Chacha8	17 M 16 M	134 K
LUOV-61-60-261	3	Keccak Chacha8	70 M 65 M	511 K
LUOV-79-76-341	5	Keccak Chacha8	308 M 301 M	2.4 M

5 Expected strength (2.B.4)

The LUOV signature system is designed for EUF-CMA security. The parameters of the LUOV scheme are chosen such that lower bounds to the bit complexity of all the known attacks exceed the required complexity level. The process of choosing the parameters is implemented in a python script which is included in the submission package. The designer specifies the desired security level and chooses the size of the field extension, then the script determines the parameters m and v to reach the required security level. Larger field extensions lead to smaller public keys at the cost of larger signatures. Table 7 summarizes the lower bounds to the complexity of the various attacks. An overview of the known attacks and what the lower bounds to their complexities are is given in section 6.

To reach security level 1 i.e. "Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for key search on a block cipher with a 128-bit key (e.g. AES128)" we assure that all known classical attacks require at least 2^{143} bit operations, which is an estimate for the number of gates required for a key search on AES128. Similarly, to reach security level 3 and 5, we require that all known classical attacks require at least 2^{207} and 2^{272} bit operations respectively. For security level 1,3 and 5, we choose parameters such that any known quantum attack requires at least 2^{106}, 2^{169} and 2^{234} gates respecively. These numbers are the estimated number of quantum gates required for a key search on AES128/AES192/AES256. In all attack scenarios the depth of

Table 7: Summary of attacks against our parameters. The table reports $\log_2$ of a **lower bound** to the number of operations required for each attack. Quantum computations are bounded to a depth of 2^{64} field operations.

		Direct forgery		Subspace forgery			UOV attack		Reconciliation attack	
(r, m, v)	security	k	complexity	d	classical	quantum	classical	quantum	classical	quantum [1]
$(7, 57, 197)$	lvl 1	3	143	2	154	107	170	107	157	133
$(7, 83, 283)$	lvl 3	6	208	2	221	185	233	169	226	219
$(7, 110, 374)$	lvl 5	8	274	2	291	266	299	234	299	310
$(47, 42, 182)$	lvl 1	1	147	9	/	/	170	106	145	118
$(61, 60, 261)$	lvl 3	1	210	12	/	/	233	169	208	197
$(79, 76, 341)$	lvl 5	1	272	15	/	/	298	234	272	277

a quantum computation is assumed to be bounded by 2^{64} quantum gates.

6 Analysis of known attacks (2.B.5)

The signature scheme is an adaptation of Oil and Vinegar [17] scheme that was proposed by Patarin in 1997. The Oil and Vinegar scheme is one of the best studied multivariate signature schemes which has, with the right parameter choices, withstood all cryptanalysis since 1997.

All the adaptations that LUOV makes to the Unbalanced Oil and Vinegar scheme (see Sect. 2.2) can be shown not to impact the security of the scheme (assuming the output of the PRNG $\mathcal{G}$ is indistinguishable from random bits), an exception being the adaptation of lifting a public key of UOV over $\mathbb{F}_2$ to a large extension field. It requires some argument to show that a direct signature forgery against the modified scheme is as difficult as a direct signature forgery against UOV over the extension field. However, since the key generation algorithm is not changed by this adaptation, it is clear that a key recovery attack against LUOV is equivalent to a key recovery attack against UOV over $\mathbb{F}_2$.

We now give an overview of known attacks. The overview is based on the overview given in [5]; We have adapted the example to match one of the proposed parameter sets.

6.1 Direct attack

This attack tries to forge a signature for a certain message M by trying to find a solution $\mathbf{s} \in \mathbb{F}_{2^r}^n$ for the system $\mathcal{F}(\mathbf{s}) = \mathcal{H}(M)$. This is an instance of the MQ (Multivariate Quadratic) problem.

[1] In some cases the complexity of a quantum attack is larger than that of a classical attack. This is due to the fact that quantum attacks are bounded to a maximium depth of 2^{64} bit operations, while classical computations are not restricted in depth. (See Sect. 6.3.2 for details.)

MQ Problem. Given a quadratic polynomial map $\mathcal{P} : \mathbb{F}_q^n \to \mathbb{F}_q^m$ over a finite field $\mathbb{F}_q$, find $\mathbf{x} \in \mathbb{F}_q^n$ that satisfies $\mathcal{P}(\mathbf{x}) = \mathbf{0}$.

Thomae and Wolf showed that finding a solution for an underdetermined system with $n = \alpha m$ can be reduced to finding a solution of a determined system with only $m + 1 - \lfloor \alpha \rfloor$ equations [20]. This means that as a system becomes more underdetermined it becomes easier to solve.

For all but very small values of q, (e.g. $q = 2, q = 3$), the best known classical algorithms to solve the MQ-problem for generic determined systems over finite fields use a hybrid approach [3, 4] that combines exhaustive search with Gröbner basis computations. In this approach k variables are fixed to random values and the remaining $n-k$ variables are found with a Gröbner basis algorithm such as F_4, F_5 or XL. If no assignment to the remaining $n-k$ variables exists that solves the system, the procedure starts again with a different guess for the first k variables. We require on average roughly q^k Gröbner basis computations until a solution is found. As a result, the optimal value of k decreases as q increases. The complexity of computing a Gröbner basis for a system of polynomials depends critically on the degree of regularity (d_{reg}) of that system. We refer to Bardet [1] for a precise definition of the degree of regularity.

The most costly part of the F_5 algorithm is doing Gaussian elimination on a large matrix with roughly $\binom{n+d_{reg}}{d_{reg}}$ rows and columns. The complexity of the F_5 algorithm is thus given by

$$C_{F_5}(n, d_{reg}) = O\left(\binom{n + d_{reg}}{d_{reg}}^{\omega}\right) ,$$

where $2 \leq \omega < 3$ is the constant in the complexity of doing Gaussian reduction on the matrices constructed in the Gröbner basis computation. These matrices are structured and sparse, which can be exploited to make Gaussian elimination more efficient [10]. The complexity of the hybrid approach is

$$C_{\text{Hybrid}F_5(n,d_{reg},k)} = O\left(q^k \binom{n - k + d_{reg}(k)}{d_{reg}(k)}^{\omega}\right) , \tag{1}$$

where $d_{reg}(k)$ stand for the degree of regularity of the system after fixing the values of k variables.

Determining the degree of regularity for a specific polynomial system is difficult, but for a certain class of systems, called semi-regular systems, it is known that the degree of regularity can be deduced from the number m of equations and the number n of variables [1, 8]. In particular, for quadratic semi-regular systems the degree of regularity is the degree of the first term in the power series of

$$S_{m,n}(x) = \frac{(1 - x^2)^m}{(1 - x)^n}$$

that has a non-positive coefficient. This gives a practical method to calculate the degree of regularity of any semi-regular system. Empirically, polynomial systems that are randomly chosen have a very large probability of being semi-regular and it is conjectured that most systems are semi-regular systems. For the definition and the theory of semi-regular systems we refer to chapter 3 of the PhD thesis of Bardet [1].

In a direct attack against the LUOV scheme all the coefficients of the system that needs to be solved lie in $\mathbb{F}_2$, except those of the constant terms, because those coefficients come from the message digest. By definition [1], the degree of regularity of a polynomial system does only depend on its quadratic part, and it is apparent that lifting a polynomial system to a field extension does not affect its degree of regularity. Therefore, the degree of regularity of a LUOV public key follows the same distribution of a UOV public key over the field $\mathbb{F}_2$, even after fixing a number of variables. It has been observed by Faugère and Perret [11] that polynomial systems that result from fixing $\approx v$ variables in a UOV system behave like semi-regular systems, whose degree of regularity does not depend on q. Therefore, the degree of regularity of a LUOV public polynomial system is distributed identically to that of a UOV public polynomial system, independently of the size q of the finite field that is used.

Since the degree of regularity, in combination with the number of variables, determines the complexity of a Gröbner basis computation (measured in number of field operations), a Gröbner basis computation on the LUOV polynomial system is not significantly more efficient than a Gröbner basis computation against regular UOV with the same parameters. This argument is confirmed by the experimental data in Table 8. There we see that a direct attack is slightly faster against the modified scheme than against the original UOV scheme, but only by a small constant factor. Even though the Gröbner basis is computed over $\mathbb{F}_{2^r}$, the largest part of the arithmetic only involves the field elements 0 and 1, so the arithmetic is faster than with generic elements of $\mathbb{F}_{2^r}$. This is where the difference observed in Table 8 comes from. If we do the same experiment with a smaller extension field such as $\mathbb{F}_{2^8}$ there is no observed difference between the running time of a direct attack against a regular UOV scheme and our modified scheme.

Remark. *In a direct attack one fixes $\approx v$ variables randomly to make the system a slightly overdetermined system. In our experiments we have fixed these variables to values in $\mathbb{F}_2$ to make sure that we do not introduce linear terms with coefficients in $\mathbb{F}_{2^r}$ instead of $\mathbb{F}_2$ in the case of the modified UOV scheme.*

Table 8: Running time of a direct attack against the regular UOV scheme over $\mathbb{F}_{2^{64}}$ and the modified UOV scheme, with the MAGMA v2.22-10 implementation of the F4 algorithm. We did not implement the method of Thomae and Wolf [20].

(m,v)	Regular UOV (s)	Lifted UOV (s)	difference
(7,35)	0.43	0.21	-52%
(8,40)	1.56	0.76	-51%
(9,45)	7.00	3.21	-54%
(10,50)	33.50	17.44	-48%
(11,55)	132.88	76.60	-42%
(12,60)	828.31	588.33	-29%

To obtain a lower bound to the complexity of a Gröbner basis computation we assume that the parameter ω in the complexity of Gaussian elimination on the matrices constructed in the Gröbner basis algorithm is equal to 2 and that the constant factor hidden by the big O notation is equal to 1. That is, in Eqn. (1) we put $\omega = 2$ and we drop the big O notation to get a concrete lower bound to the number of bit operations of a hybrid attack. Even though this is a generous lower bound, we require that this lower bound exceeds the required bit complexity when choosing parameters.

Example. *We will estimate the complexity of a direct attack against LUOV with the parameter set* $(r = 7, m = 57, v = 197)$*; this set is proposed as a set that achieves security level 1. Using the method of Thomae and Wolf. we can reduce finding a solution to this underdetermined system to finding a solution of a determined system with* $57 - \lfloor (197)/57 \rfloor = 54$ *equations. We assume this system, and the systems that are derived by fixing a number of variables, to be semi-regular. If we fix k extra variables the degree of regularity is equal to the degree of the first term in the power series of*

$$S_{54,54-k}(x) = \frac{(1-x^2)^{54}}{(1-x)^{54-k}}$$

which has a non-positive coefficient. For $k = 0$ *we have* $S_{54,54}(x) = (1+x)^{54}$*, so the degree of regularity is* 55*. For* $k = 1$ *we have*

$$S_{54,53}(x) = 1 + 53x + \cdots + 69533550916004x^{27} - 69533550916004x^{28} + O(x^{29}),$$

where all the omitted terms have positive coefficients, so the degree of regularity is 28*. We can now use (1) to obtain a lower bound to the complexity of the hybrid approach. For k equal to* 0 *and* 1 *this is equal to*

$$\binom{54+55}{55}^2 \approx 2^{210.6} \qquad \textit{and} \qquad 2^7\binom{54-1+28}{28}^2 \approx 2^{150.8}$$

respectively. Repeating this calculation for higher values of k we eventually see that the optimal value of k is 3, the corresponding degree of regularity is 22 *and the complexity of the direct attack is estimated as* $2^{143.3}$*. Thus, this lower bound exceeds* 2^{143}*, as required.*

In theory, a quantum attacker could use Grover search instead of the brute force part of the hybrid approach to speed up a direct attack. The complexity of this attack would be

$$C_{\mathrm{Hybrid}F_5(n,d_{reg},k)} = O\left(q^{k/2}\binom{n-k+d_{reg}(k)}{d_{reg}(k)}^{\omega}\right), \qquad (2)$$

where the only difference with (1) is that the factor q^k is replaced by $q^{k/2}$. However, this attack is not possible if the depth of a quantum computation is limited to, say, 2^{64} operations. For all our parameter choices and all practical values of k, the complexity of even a single Gröbner basis computation is beyond 2^{64}, and the Grover algorithm should do a large number of these computations sequentially in order to enjoy a noticeable speedup over the classical brute force search.

6.2 Subfield and subspace differential attacks

6.2.1 Subfield differential attack

Ding et al. proposed a new method to solve underdetermined quadratic polynomial systems of the form $\mathcal{P}(\mathbf{x}) = \mathbf{y}$, where the coefficients of $\mathcal{P}$ lie in $\mathbb{F}_2$, but the entries of $\mathbf{y}$ lie in an extension field $\mathbb{F}_{2^r}$. This method can be used to forge a LUOV signature. This method requires the existence of a subfield $\mathbb{F}_{2^d} \subset \mathbb{F}_{2^r}$. The method then picks a random vector $\mathbf{x}_0 \in \mathbb{F}_{2^r}^n$, the goal of the attack is to find a "correction vector" $\mathbf{x}'$ that lives in the subfield $\mathbb{F}_{2^d}$, such that $\mathcal{P}(\mathbf{x}_0 + \mathbf{x}') = \mathbf{y}$. Heuristically, since there are 2^{dn} possibilities for $\mathbf{x}'$ and 2^{rm} possibilities for $\mathcal{P}(\mathbf{x}_0 + \mathbf{x}')$, we expect that a correction vector over $\mathbb{F}_{2^d}$ exists if $nd \geq mr$. This implies that the subfield $\mathbb{F}_{2^d}$ should not be too small.

It is clear that for every multivariate quadratic map $\mathcal{P} : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$, there exists a bilinear map $\mathcal{G} : \mathbb{F}_2^n \times \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ (called the polar form of $\mathcal{P}$), such that

$$\mathcal{P}(\mathbf{a} + \mathbf{b}) = \mathcal{P}(\mathbf{a}) + \mathcal{P}(\mathbf{b}) + \mathcal{G}(\mathbf{a}, \mathbf{b})\,.$$

Using this identity we get that

$$\mathbf{y} = \mathcal{P}(\mathbf{x}_0) + \mathcal{P}(\mathbf{x}') + \mathcal{G}(\mathbf{x}_0, \mathbf{x}'). \tag{3}$$

For a fixed value of $\mathbf{x}_0$, the only quadratic part is the term $\mathcal{P}(\mathbf{x}')$, which is defined over the subfield $\mathbb{F}_{2^d}$. The constant and linear terms are defined over $\mathbb{F}_{2^r}$. Suppose $\{1, t, t^2, \cdots, t^{\frac{r}{d}-1}\}$ is a $\mathbb{F}_{2^d}$-basis for $\mathbb{F}_2^r$. If we write the equation (3) in terms of this basis, we get m quadratic equations, and $(\frac{r}{d}-1)m$ linear equations over $\mathbb{F}_{2^d}$. After eliminating variables with the linear equations, we end up with m quadratic equations in $n - (\frac{r}{d} - 1)m$ variables.

To summarize, it turns out that solving $\mathcal{P}(\mathbf{x}_0 + \mathbf{x}') = \mathbf{y}$ for $\mathbf{x}'$ boils down to solving a system of m multivariate quadratic equations in $n - (\frac{r}{d} - 1)m$ variables over $\mathbb{F}_{2^d}$. If a good subfield exists (which was is the case for the original LUOV parameter sets), then this approach is more efficient than a direct attack (which amounts to solving m equations in n variables over $\mathbb{F}_{2^r}$), because solving systems over the smaller subfield $\mathbb{F}_{2^d}$ is more efficient than solving systems over $\mathbb{F}_{2^r}$. However, the current proposed parameter sets use a field extension of prime degree, therefore there do not exist nontrivial subfields that can be used to mount the attack.

6.2.2 Subspace differential attack

We can adapt the subfield differential attack to use a *subspace* of $\mathbb{F}_{2^r}$ instead of a subfield. This generalized attack will be applicable even if r is prime. To do this, we replace the role of the subfield $\mathbb{F}_{2^d}$ by a $\mathbb{F}_2$-subspace V of dimension d. We now pick a random $\mathbf{x}_0$ and we try to solve the system $\mathcal{P}(\mathbf{x}_0 + \mathbf{x}') = \mathbf{y}$ for some vector $\mathbf{x}'$ with entries in V. It is clear that if r is prime, and $1 < d < r$ the dimension of the subspace spanned by vectors of the form $\mathbf{v}_1\mathbf{v}_2$, for $\mathbf{v}_1, \mathbf{v}_2 \in \mathbf{V}$ is at least $d + 1$. Otherwise, if $\dim(\langle V^2 \rangle) = d = \dim(V)$, then V must be (a translation of) a subfield with 2^d elements, which is a contradition.

Table 9: Estimate of the complexity of the subspace differential attack against the proposed parameter sets. The number of evaluations in the Grover algorithm assumes a maximal depth of 2^{64} evaluations. For the parameter sets aiming for smaller public keys, the systems are very large, so we do not estimate the complexity of solving them.

	Security				Joux-Vitse			Grover
Parameter set	level	d	n	m	k	D	$\log_2$ cost	$\log_2$ evaluations
LUOV-7-57-197	1	2	171	280	32	14	154	106
LUOV-7-83-283	3	2	249	400	45	22	221	185
LUOV-7-110-374	5	2	330	528	57	28	291	266
LUOV-47-42-182	1	9	714	756	/	/	/	/
LUOV-61-60-261	3	12	1380	1572	/	/	/	/
LUOV-79-76-341	5	15	2204	2455	/	/	/	/

Now we extend a basis of $\langle V^2 \rangle$ to a basis of $\mathbb{F}_2^r$ and write out the coordinates of equation (3) with respect of this basis to get rm equations over $\mathbb{F}_2$. Then, we see that because the only quadratic part of equation 3 is $\mathcal{F}(\mathbf{x}') \in \langle V^2 \rangle$, we see that at most $m * (r - d - 1)$ of these equations are linear. After eliminating $m(r-d-1)$ variables with these linear equations, we end up with a system of at least $m(d+1)$ quadratic equations in $dn - m(r-d-1)$ variables.

For our proposed parameter sets, this attack boils down to solving quadratic systems over $\mathbb{F}$ with sizes given in Table 9. We estimate the complexity of solving those systems with the Joux-Vitse crossbred algorithm [13], which is the best known algorithms for solving boolean quadratic system of those sizes. We conclude that solving those systems classically is not more efficient than a direct attack. However, for the $r = 7$ parameter sets, solving the system with Grover's algorithm gives a competitive quantum attack.

6.3 Key recovery attacks.

Since the key pair generation algorithm used by the LUOV scheme is identical to that of the original UOV scheme over the field $\mathbb{F}_2$ it is clear that a key recovery attack against the Lifted UOV scheme is equivalent to a key recovery attack against a regular UOV scheme over $\mathbb{F}_2$. Key recovery attacks against UOV have been investigated ever since the invention of the Oil and Vinegar scheme in 1997 [17], so it is well understood which attacks are possible and what the complexities of these attacks are. It is also clear that we can make key recovery attacks harder by increasing the number of vinegar variables.

6.3.1 UOV attack

Patarin [17] suggested in the original version of the Oil and Vinegar scheme to choose the same number of vinegar and oil variables, or $v = m$. This choice was cryptanalyzed by Kipnis and Shamir [15]: they showed that an attacker can find the inverse image of the oil variables under the map $\mathcal{T}$. This is enough information to find an equivalent secret key, so this breaks the scheme. This approach generalizes for the case $v > m$; the complexity then increases to $O(q^{v-m}n^4)$ [14] and is thus exponential in $v - m$. Since a UOV attack on the Lifted UOV scheme is equivalent to a UOV attack over $\mathbb{F}_2$, we have that the complexity of a UOV attack against the Lifted UOV scheme is approximately $2^{v-m-1} \cdot n^4$ binary operations.

The generalized UOV attack chooses a random linear combination of the matrices that represent the quadratic parts of the polynomials in the public system and computes the minimal eigenspaces of the matrix. With probability 2^{m-v+1} this computation yields a vector in the oil subspace. This means that a quantum attacker can use the Grover search algorithm [12] to look for a random linear combination that will yield a vector in the oil subspace. Ignoring issues of 'Groverizing' the algorithm such as making the computation reversible and the probabilistic nature of the eigenspace computation, the complexity of a quantum attack becomes $2^{\frac{v-m-1}{2}}n^4$. If we limit the depth of a quantum computation to 2^{depth}, and we ignore the depth of the eigenspace-finding subroutine, the complexity of an attack is at least $\max(2^{\frac{v-m-1}{2}}n^4, 2^{v-m-1}n^4/2^{depth})$.

6.3.2 Reconciliation attack

The reconciliation attack against the lifted UOV scheme is equivalent to the UOV reconciliation attack against UOV over the field $\mathbb{F}_2$. A lower bound on the complexity of this attack is given by the complexity of solving a quadratic system of v variables and v equations over $\mathbb{F}_2$, but the problem is expected to be harder [5]. There exists specialized algorithms for solving polynomial systems over $\mathbb{F}_2$ that are more efficient than the generic hybrid approach. One method is a smart exhaustive search, which requires approximately $log_2(n)2^{n+2}$ bit operations [6]. The BooleanSolve algorithm [2] combines an exhaustive search with sparse linear algebra to achieve a complexity of $O(2^{0.792n})$. However the method only becomes faster than the exhaustive search method when $n > 200$. Recently, Joux and Vitse proposed a new algorithm that was able to solve a Boolean system of 146 quadratic equations in 73 variables in one day [13]. The algorithm beats the exhaustive search algorithm, even for small systems. The complexity of this algorithm is still under investigation, but a rough estimate based on the reported experiments suggests that the number of operations scales like $2^{\alpha n}$ with α between 0.8 and 0.85 and with a constant factor between 2^7 and 2^{10}. For choosing the parameters of the LUOV signature scheme, we have assumed that finding a solution to a determined system of n quadratic Boolean equations requires $2^{0.8n}$ operations in $\mathbb{F}_2$, even though this is likely to seriously overestimate the capabilities of the state of the art algorithms.

Due to the limit on the circuit depth of quantum computations, the Gröbner based methods of solving a Boolean system cannot be 'Groverised'. In contrast, quantum attackers can still use a brute force Grover search to solve systems over $\mathbb{F}_2$ with $2^{n/2}$ sequential evaluations of the polynomials in the system. However, if the depth of a quantum computation is restricted to 2^{depth} evaluations of the polynomials, the required number of polynomial evaluations in a Grover search is at least $\max(2^{n-depth}, 2^{n/2})$. Asymptotically this is worse than the classical

Gröbner basis based methods, which is why the reported hardness of a quantum reconciliation attack in Table 6 is higher than the hardness of the classical reconciliation attack. One would expect quantum attacks to be at least as efficient as classical attacks, because a quantum computer can simulate a classical computer. In our analysis this is not the case, because the depth of a quantum computation is assumed to be limited, which is not the case for a classical computation.

6.4 Hash collision attack

As is the case for all hash-and-sign digital signature algorithms, a hash collision can be exploited to break the EUF-CMA security definition. The SHAKE extendable output functions are used to generate a hash digest of the required length. The parameter sets claiming a security level 2 use SHAKE-128, those claiming security level 4 or 5 use SHAKE-256. In each proposed parameter set the output length (i.e. rm bits) is large enough to reach the required hardness of finding collisions. Therefore, a hash collision attack does not threaten the claimed security levels.

7 Advantages and limitations (2.B.6)

7.1 Advantages

- **Small signatures.** Like many other MQ signature schemes, the signatures of the LUOV scheme are very small. For security level 2 the signatures are only 311 bytes long.

- **Small secret key.** The secret key consists of 32 random bytes.

- **A wide security margin** Instead of trying to estimate the complexity of existing attacks and choosing the parameters such that these estimates match the required security level we have formulated conservative lower bounds to plausible attacks. For example, we have assumed that a classical attacker can solve a determined system of n Boolean quadratic polynomials with only $2^{0.80n}$ bit operations, whereas the best known algorithms perform much worse.

- **Simple arithmetic.** The scheme only uses SHA-3 and simple arithmetic operations over $\mathbb{F}_2$ or over an extension field. Arithmetic over $\mathbb{F}_2$ translates to the operations AND and XOR, while the arithmetic over an extension field can be implemented with XOR, additions and table lookups in small tables. This makes the algorithm very suitable for hardware implementations.

- **Message recovery.** It is possible to use the LUOV scheme in a message recovery mode. In this mode, a part of the message can be recovered from the signature and does not need to be communicated. This can reduce the size of a message-signature pair by up to 15 percent of the signature size.

- **Stateless.** The signing algorithm does not need to maintain a state between signing sessions and can sign an unbounded number of messages. This makes a secure implementation of the algorithm easier.

- **Flexible.** The parameters of the signature are easily adjustable to reach a specific security level. It is also possible to choose parameters to make a trade-off between small signatures and small public keys.

- **Diversity.** Multivariate cryptography relies on a different hard problem than other branches such as lattice cryptography or hash-based cryptography. It is prudent to have cryptographic algorithms that rely on a diverse set of hard problems such that if one hard problem is broken and wipes out a branch of cryptography, there are alternative algorithms available.

7.2 Limitations

- **Public key size.** Even though the public key size of the LUOV scheme is much smaller than the public key size of other MQ signature schemes, it remains larger than the public key size of some other post quantum signature schemes. It is possible to mitigate this problem by making a trade-off for a smaller public key at the cost of larger signatures.

- **No encryption or KEM.** The LUOV scheme is a digital signature scheme. This submission does not include an encryption scheme or a key encapsulation mechanism.

References

[1] Magali Bardet. *Étude des systèmes algébriques surdéterminés. Applications aux codes correcteurs et à la cryptographie.* PhD thesis, Université Pierre et Marie Curie-Paris VI, 2004.

[2] Magali Bardet, Jean-Charles Faugère, Bruno Salvy, and Pierre-Jean Spaenlehauer. On the complexity of solving quadratic Boolean systems. *Journal of Complexity*, 29(1):53–75, 2013.

[3] Luk Bettale, Jean-Charles Faugère, and Ludovic Perret. Hybrid approach for solving multivariate systems over finite fields. *Journal of Mathematical Cryptology*, 3(3):177–197, 2009.

[4] Luk Bettale, Jean-Charles Faugère, and Ludovic Perret. Solving polynomial systems over finite fields: Improved analysis of the hybrid approach. In *Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation*, pages 67–74. ACM, 2012.

[5] Ward Beullens and Bart Preneel. Field lifting for smaller UOV public keys. In *Progress in Cryptology–INDOCRYPT 2017: 18th International Conference on Cryptology in India, Chennai, India, December 10-13, 2016, Proceedings 18*. Springer, 2017.

[6] Charles Bouillaguet, Hsieh-Chung Chen, Chen-Mou Cheng, Tung Chou, Ruben Niederhagen, Adi Shamir, and Bo-Yin Yang. Fast exhaustive search for polynomial systems in $\mathbb{F}_2$. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 203–218. Springer, 2010.

[7] Peter Czypek. *Implementing Multivariate Quadratic Public Key Signature Schemes on Embedded Devices*. PhD thesis, Diploma Thesis, Chair for Embedded Security, Ruhr-Universität Bochum, 2012.

[8] Claus Diem. The XL-algorithm and a conjecture from commutative algebra. In *Asiacrypt*, volume 4, pages 338–353. Springer, 2004.

[9] Jintai Ding, Zheng Zhang, Joshua Deaton, Kurt Schmidt, and FNU Vishakha. New attacks on lifted unbalanced oil vinegar. *The 2nd NIST PQC Standardization Conference.*

[10] Jean-Charles Faugère and Sylvain Lachartre. Parallel Gaussian elimination for Gröbner bases computations in finite fields. In *Proceedings of the 4th International Workshop on Parallel and Symbolic Computation*, pages 89–97. ACM, 2010.

[11] Jean-Charles Faugère and Ludovic Perret. On the security of UOV. *IACR Cryptology ePrint Archive*, 2009:483, 2009.

[12] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219. ACM, 1996.

[13] Antoine Joux and Vanessa Vitse. A crossbred algorithm for solving Boolean polynomial systems. *IACR Cryptology ePrint Archive*, 2017:372, 2017.

[14] Aviad Kipnis, Jacques Patarin, and Louis Goubin. Unbalanced Oil and Vinegar signature schemes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 206–222. Springer, 1999.

[15] Aviad Kipnis and Adi Shamir. Cryptanalysis of the Oil and Vinegar signature scheme. In *Annual International Cryptology Conference*, pages 257–266. Springer, 1998.

[16] Bo Lv, Zhiniang Peng, and Shaohua Tang. Precomputation methods for UOV signature on energy-harvesting sensors. *IEEE Access*, 6:56924–56933, 2018.

[17] Jacques Patarin. The Oil and Vinegar signature scheme. In *Dagstuhl Workshop on Cryptography1997*, 1997.

[18] Albrecht Petzoldt. *Selecting and Reducing Key Sizes for Multivariate Cryptography.* PhD thesis, TU Darmstadt, July 2013. Referenten: Professor Dr. Johannes Buchmann, Professor Jintai Ding, Ph.D.

[19] Oscar Reparaz, Josep Balasch, and Ingrid Verbauwhede. Dude, is my code constant time? In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 1697–1702. IEEE, 2017.

[20] Enrico Thomae and Christopher Wolf. Solving underdetermined systems of multivariate quadratic equations revisited. In *International Workshop on Public Key Cryptography*, pages 156–171. Springer, 2012.

[21] Christopher Wolf and Bart Preneel. Equivalent keys in multivariate quadratic public key systems. *Journal of Mathematical Cryptology*, 4(4):375–415, 2011.

A History of LUOV submission

A.1 Changes made for the Second Round version of LUOV

The changes between the first round version and the second round version of LUOV are the following:

- **Updated parameters.** Compared to other NIST Round 1 MQ schemes it was clear that the parameter choice of LUOV was too conservative. Therefore we dropped the "10% in the exponent" security margin that we had in the Round I version of the document. The updated parameters result in a faster signature scheme with smaller keys and signatures.
- **Add a salt.** The second round version of includes a 16-byte salt to each message. This improves the security of LUOV against side-channel attacks and fault injection attack.
- **Choose vinegar variables randomly.** In a previous version of LUOV the vinegar variables were chosen deterministically. In the second round version they are chosen at random. This improves the security of LUOV against side-channel attacks and fault injection attacks. Moreover this makes it possible to do most of the signing work offline which makes LUOV suitable for applications were a very low signing latency is required.
- **Sampling public map from PRNG.** To improve the efficiency of sampling the public map we have introduced ChaCha8 as an option. We also sample the public map in sets of 16 polynomials. This makes it possible to evaluate multiple instances of the PRNG (Keccak or ChaCha8) in parallel for improved efficiency. Moreover this allows for implementations with a smaller memory footprint.
- **Add an AVX2 optimized implementation.** We report on the cycle count of our AVX2 optimized implementation, as well as the variant that uses precomputation on the secret key and public key.
- **Add a generic C implementation with online and offline signing phases.** See Sect. 4.5

A.2 Changes in response to the subfield differential attack

On the 23th of July 2019 the list of accepted papers of the NIST Second PQC Standardization Conference were made available online. Among these papers was the "New Attacks on Lifted Unbalanced Oil Vinegar" paper authored by Ding et al [9]. This paper describes the subfield differential attack (see Sect. 6.2.1). While we believe there are some mistakes in the estimate of the complexity of the attack, we do agree that the attack breaks the claimed security levels for at least some of the proposed parameter sets. Therefore, we choose to propose new parameter sets where the degree of the field extension is prime, such that there are no intermediate fields that can be used to launch the attack. In Sect. 6.2.2, we study a generalization of the attack, where we exploit the existance of subspaces, rather than subfields. However, this makes the attack less efficient. In particular, the subspace differential attack is slightly less efficient than a direct attack.

B Statements

B.1 Statement by Each Submitter

I, Ward Beullens, of Afdeling ESAT - COSIC, Kasteelpark Arenberg 10 - bus 2452, 3001 Heverlee, Belgium, do hereby declare that the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as LUOV, is my own original work, or if submitted jointly with others, is the original work of the joint submitters. I further declare that (check one):

☑ *I do not hold and do not intend to hold any patent or patent application with a claim which may cover the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as LUOV OR (check one or both of the following):*

- □ *to the best of my knowledge, the practice of the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as LUOV may be covered by the following U.S. and/or foreign patents: None*
- □ *I do hereby declare that, to the best of my knowledge, the following pending U.S. and/or foreign patent applications may cover the practice of my submitted cryptosystem, reference implementation or optimized implementations: None*

I do hereby acknowledge and agree that my submitted cryptosystem will be provided to the public for review and will be evaluated by NIST, and that it might not be selected for standardization by NIST. I further acknowledge that I will not receive financial or other compensation from the U.S. Government for my submission. I certify that, to the best of my knowledge, I have fully disclosed all patents and patent applications which may cover my cryptosystem, reference implementation or optimized implementations. I also acknowledge and agree that the U.S. Government may, during the public review and the evaluation process, and, if my submitted cryptosystem is selected for standardization, during the lifetime of the standard, modify my submitted cryptosystem's specifications (e.g., to protect against a newly discovered vulnerability).

I acknowledge that NIST will announce any selected cryptosystem(s) and proceed to publish the draft standards for public comment.

I do hereby agree to provide the statements required by Sections 2.D.2 and 2.D.3, below, for any patent or patent application identified to cover the practice of my cryptosystem, reference implementation or optimized implementations and the right to use such implementations for the purposes of the public review and evaluation process.

I acknowledge that, during the post-quantum algorithm evaluation process, NIST may remove my cryptosystem from consideration for standardization. If my cryptosystem (or the derived cryptosystem) is removed from consideration for standardization or withdrawn from consideration by all submitter(s) and owner(s), I understand that rights granted and assurances made under Sections 2.D.1, 2.D.2 and 2.D.3, including use rights of the reference and optimized *implementations, may be withdrawn by the submitter(s) and owner(s), as appropriate.*

Signed: Ward Beullens
Title:
Date:
Place:

B.2 Statement by Reference/Optimized Implementations' Owner(s)

I, Ward Beullens, Afdeling ESAT - COSIC, Kasteelpark Arenberg 10 - bus 2452, 3001 Heverlee, Belgium, am the owner or authorized representative of the owner Ward Beullens of the submitted reference implementation and optimized implementations and hereby grant the U.S. Government and any interested party the right to reproduce, prepare derivative works based upon, distribute copies of, and display such implementations for the purposes of the post-quantum algorithm public review and evaluation process, and implementation if the corresponding cryptosystem is selected for standardization and as a standard, notwithstanding that the implementations may be copyrighted or copyrightable.

Signed: Ward Beullens
Title:
Date:
Place: