SCIS 2023 2023 Symposium on
Cryptography and Information Security
Fukuoka, Japan, Jan. 24 – 27, 2023
The Institute of Electronics,
Information and Communication Engineers

Efficient Software Implementation of Signature Scheme QR-UOV

Fumitaka Hoshino* Hiroki Furue† Yasuhiko Ikematsu‡ Tsunekazu Saito§
Yutaro Kiyomura§ Tsuyoshi Takagi†

Abstract: In ASIACRYPT 2021, Furue et al. proposed a new post-quantum signature scheme called QR-UOV, which is a variant of the multivariate signature schemes. A provable version of QR-UOV is proposed by the authors in SCIS 2023. In this work, we present an efficient software implementation of it and evaluate its performance on some x86 environments.

Keywords: PQC, MPKCs, quotient ring UOV (QR-UOV), software implementation

1 Introduction

Multivariate public key cryptosystems (MPKCs) are regarded as a strong candidate for post-quantum cryptography (PQC) which has been widely studied around the world. The security of MPKCs is based on the difficulty of the multivariate quadratic ($\mathcal{MQ}$) problem i.e. solving a system of multivariate quadratic polynomial equations over a finite field. It is known that the general $\mathcal{MQ}$ problem is NP-complete [5], therefore the security of MPKCs is expected to resist against attacks using quantum computers.

In EUROCRYPT 1999, Kipnis et al. proposed a signature scheme called the unbalanced oil and vinegar (UOV) signature, which is based on a special type of $\mathcal{MQ}$ problem [7]. The UOV signature scheme has withstood various types of attacks for approximately 20 years. Although UOV is a well-established due to its short signature size and execution time, its public key has much larger size than those of other PQC candidates. Therefore developing a UOV variant with a small public-key size is an important task. Rainbow [2] was one of such UOV variants, which was proposed by Ding and Schmidt. It was selected as a third-round finalist in the NIST PQC project [8], however Beullens found a serious flaw in this scheme [1].

In ASIACRYPT 2021, Furue et al. proposed another variant of the UOV called QR-UOV [4]. A modified version of it is proposed by the authors in SCIS 2023, whose unforgeability can be proven under some assumptions [3]. In this work, we investigate this version of QR-UOV to implement an efficient software on the x86 environments.

* Faculty of Information Systems, University of Nagasaki, 1-1-1, Manabino, Nagayo-cho, Nishisonogi-gun, Nagasaki, 851-2195, Japan. (hoshino@sun.ac.jp)
† Department of Mathematical Informatics, The University of Tokyo, 7-3-1, Hongo, Bunkyo-ku, Tokyo, 113-8656, Japan. ({furue-hiroki261,takagi}@g.ecc.u-tokyo.ac.jp)
‡ Institute of Mathematics for Industry, Kyushu University, 744, Motooka, Nishi-ku, Fukuoka, 819-0395, Japan. (ikematsu@imi.kyushu-u.ac.jp)
§ NTT Social Informatics Laboratories, 3-9-11, Midori-cho, Musashino-shi, Tokyo, 180-8585, Japan. (tsunekazu.saito@ntt.com, yutaro.kiyomura.vs@hco.ntt.co.jp)

2 Preliminary

2.1 Basic Notions and Notations

The following list is an overview of typical arrow expressions which we use to describe algorithms.

- $X \leftarrow Y$ The value of the expression Y is assigned to the variable X.
- $X \xleftarrow{\circ} Y$ A new value $X \circ Y$ is assigned to the variable X, for any binary operator $\circ$, e.g. $X \xleftarrow{\cup} Y$ means $X \leftarrow X \cup Y$.
- $X \xleftarrow{\$} Y$ X is uniformly selected from Y, assuming that Y is a set.
- $X \xleftarrow{\$} Y()$ X is randomly selected from the output space of the probabilistic Turing machine Y according to the distribution of Y's output when Y's random tape is uniformly selected.
- $X \xmapsto{\$} Y$ A probabilistic Turing machine which takes X as an input tape and outputs Y.
- $X \to Y$ All of maps from X to Y, assuming that X and Y are sets, which is identical with Y^X.
- $X \mapsto Y$ the map which returns Y for X.

In this work, we identify the set of integers $\{0, 1\}$ with the set of truth values. Namely 0 is interpreted as false and 1 as truth. For a probabilistic Turing machine Y whose input is X and output is in $\{0, 1\}$, we say Y accepts X if $Y(X)$ returns 1, or Y rejects X if $Y(X)$ returns 0.

2.2 Signature Scheme

A digital signature scheme Σ consists of three probabilistic polynomial-time algorithms (`KeyGen`, `Sign`, `Verify`), which are defined as follows:

Key generation algorithm $\texttt{KeyGen} : 1^\lambda \xmapsto{\$} (\texttt{pk}, \texttt{sk}) \in (\{0,1\}^*)^2$, returns a pair of public key $\texttt{pk}$ and secret key $\texttt{sk}$ for given security parameter 1^λ.

Signing algorithm $\texttt{Sign} : \mathbf{M}, \texttt{pk}, \texttt{sk} \xmapsto{\$} \sigma \in \{0,1\}^*$, takes a message $\mathbf{M} \in \{0,1\}^*$ and the signer's key pair $(\texttt{pk}, \texttt{sk})$, and then generates a digital signature σ of the signer for the message $\mathbf{M}$.

Verification algorithm $\texttt{Verify} : \mathbf{M}, \texttt{pk}, \sigma \xmapsto{\$} \beta \in \{0, 1\}$, takes a message $\mathbf{M} \in \{0,1\}^*$, a public key $\texttt{pk}$ and a signature σ, and accepts or rejects the message $\mathbf{M}$.

Let $\texttt{Adv}^{\texttt{complete}} : \mathbb{N} \to [0,1]$ be a function as follows.

$$\texttt{Adv}^{\texttt{complete}}(\lambda) := \Pr\left[\beta = 1 \;\middle|\; \begin{array}{l} (\texttt{pk}, \texttt{sk}) \xleftarrow{\$} \texttt{KeyGen}(1^\lambda), \\ \mathbf{M} \xleftarrow{\$} \{0,1\}^{\texttt{poly}(\lambda)}, \\ \sigma \xleftarrow{\$} \texttt{Sign}(\mathbf{M}, \texttt{pk}, \texttt{sk}), \\ \beta \xleftarrow{\$} \texttt{Verify}(\mathbf{M}, \texttt{pk}, \sigma). \end{array}\right].$$

We say a signature scheme $\Sigma := (\texttt{KeyGen}, \texttt{Sign}, \texttt{Verify})$ is complete in λ iff $\texttt{Adv}^{\texttt{complete}}(\lambda)$ is overwhelming in λ. In the following sections, we regard the completeness as a part of the syntax of signature scheme, thus we assume implicitly that any signature schemes are complete.

Let $\texttt{invalid}$ be a set of messages and $\mathcal{O}_{\texttt{pk},\texttt{sk}}$ be a signing oracle defined as follows.

$\mathcal{O}_{\texttt{pk},\texttt{sk}}(\mathbf{M}) :=$

$\sigma \xleftarrow{\$} \texttt{Sign}(\mathbf{M}, \texttt{pk}, \texttt{sk})$,
$\texttt{invalid} \xleftarrow{\cup} \{\mathbf{M}\}$,
return σ.

Let $\mathcal{A}$ be a forger against the signature scheme Σ. We define $\texttt{Adv}^{\texttt{EUF-CMA}}_{\mathcal{A}} : \mathbb{N} \to [0,1]$ as

$$\texttt{Adv}^{\texttt{EUF-CMA}}_{\mathcal{A}}(\lambda) := \Pr\left[\beta = 1 \;\middle|\; \begin{array}{l} (\texttt{pk}, \texttt{sk}) \xleftarrow{\$} \texttt{KeyGen}(1^\lambda), \\ \texttt{invalid} \leftarrow \varnothing, \\ (\mathbf{M}^*, \sigma^*) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\texttt{pk},\texttt{sk}}}(\texttt{pk}), \\ \beta \xleftarrow{\$} \texttt{Verify}(\mathbf{M}^*, \texttt{pk}, \sigma^*) \\ \qquad\qquad \wedge (\mathbf{M}^* \notin \texttt{invalid}). \end{array}\right].$$

We say a signature scheme $\Sigma := (\texttt{KeyGen}, \texttt{Sign}, \texttt{Verify})$ is EUF-CMA in λ iff for any probabilistic polynomial-time algorithm $\mathcal{A}$, $\texttt{Adv}^{\texttt{EUF-CMA}}_{\mathcal{A}}(\lambda)$ is negligible in λ.

In SCIS 2023, Furue et al. proposed a modified version of the QR-UOV signature scheme, which can be proven to be EUF-CMA in the random oracle model under some assumptions. In this work, we will investigate this version of the QR-UOV signature scheme.

2.3 Basic Concept of the QR-UOV

Let $f \in \mathbb{F}_q[x]$ be a polynomial in $\mathbb{F}_q$ of degree ℓ. For each element $g \in \mathbb{F}_q[x]/(f)$, there exists a matrix $\Phi^f_g \in \mathbb{F}_q^{\ell\times\ell}$ s.t.

$$(1, x, \ldots, x^{\ell-1})\Phi^f_g = (g, xg, \ldots, x^{\ell-1}g) \in (\mathbb{F}_q[x]/(f))^\ell.$$

Let $A_f := \{\Phi^f_g \in \mathbb{F}_q^{\ell\times\ell} \mid g \in \mathbb{F}_q[x]/(f)\}$. A_f is a subalgebra of $\mathbb{F}_q^{\ell\times\ell}$ because

$$\Phi^f_{g_1+g_2} = \Phi^f_{g_1} + \Phi^f_{g_2},$$
$$\Phi^f_{g_1 g_2} = \Phi^f_{g_1}\Phi^f_{g_2}.$$

Hence an element in A_f, i.e. $\ell\times\ell$ $\mathbb{F}_q$-matrix can be represented as an element in $\mathbb{F}_q[x]/(f)$, i.e. ℓ elements of $\mathbb{F}_q$. One might think that this property of A_f could be available to construct a UOV variant with short public key. However this technique cannot be applied directly to the UOV variants with another promising technique using symmetric matrices, since the elements of A_f are generally unstable under the transpose operation. Furue et al. solve this problem by introducing an invertible matrix $W \in \mathbb{F}_q^{\ell\times\ell}$ s.t. $W\Phi^f_g$ is transpose for any $\Phi^f_g \in A_f$, thus they propose a UOV variant using $WA_f := \{W\Phi^f_g \in \mathbb{F}_q^{\ell\times\ell} \mid \Phi^f_g \in A_f\}$, that is QR-UOV [4]. Therefore, throughout this work, we will identify the element $g \in \mathbb{F}_q[x]/(f)$ with its matrix form $W\Phi^f_g \in WA_f$. When we need to distinguish them, we write its matrix form $W\Phi^f_g$ as $\tilde{g}$ for $g \in \mathbb{F}_q[x]/(f)$, i.e.

$$\tilde{g} := W\Phi^f_g.$$

Moreover, for $G := (g_{ij})_{i\in[n_1],j\in[n_2]} \in (\mathbb{F}_q[x]/(f))^{n_1\times n_2}$, we define

$$\widetilde{G} := (W\Phi^f_{g_{ij}})_{i\in[n_1],j\in[n_2]} \in (WA_f)^{n_1\times n_2},$$

where $[n] := \{1, \ldots, n\}$. Similarly, for $m \in \mathbb{N}$, we denote the set of integers $\{1, \ldots, m\}$ as $[m]$ in the following sections.

2.4 LU decomposition

In usual MPKC schemes, some basic operations of linear algebra over a finite field are heavily performed. Of course a UOV variant also uses many operations of matrices over $\mathbb{F}_q$. Especially in its $\texttt{Sign}$ algorithm, one must solve a linear equation system which may be over/underdetermined. It is known that the study of Gaussian elimination has been continued for more than 2,200 years [6], thus a fair amount of studies for this problem likely to be found in the vast amount of literature. However, just for convenience here we describe a variant of it.

The following $\texttt{LUdecompose}$ algorithm is a variant of the LU decomposition (Gaussian elimination) for matrices which is not necessarily invertible. Using this algorithm, a row echelon form can be obtained instead of an upper triangular matrix. This feature of $\texttt{LUdecompose}$ is suitable for the modified version of QR-UOV. On input of a matrix $A \in \mathbb{F}_q^{m\times m}$, $\texttt{LUdecompose}$ outputs a list of matrices $(P, L, U) \in \left(\mathbb{F}_q^{m\times m}\right)^3$ and some additional information, where $A = P^{-1}LU$. Each of (P, L, U) is a matrix of special form as in the following list.

P: a permutation matrix, of course invertible,

L: an invertible lower triangular matrix,

U: a row echelon form where the leading entry in each nonzero row is 1, which is not necessarily invertible.

In the following, we write zero and identity matrices of dimension m as O and I respectively, and $\texttt{row\_swap}(A, P, i, j)$ means the operation to exchange row i and row j for each of A and P.

$\texttt{LUdecompose}(A \in \mathbb{F}_q^{m\times m}) :=$

$P \leftarrow I,\ \ c \leftarrow -1,\ \ \texttt{rank} \leftarrow 0,\ \ \texttt{index} \leftarrow (),$
for $i = 1$ **to** m,
  c++, **if** $c > m$ **break to** label 1,
  $j \leftarrow i$,
  while $A_{j,c} = 0$,
    j++,
    if $j > m$,
      c++, **if** $c > m$ **break to** label 1,
      $j \leftarrow i$,
  $\texttt{row\_swap}(A, P, i, j)$,
  $\texttt{rank}$++,
  $\texttt{index} \leftarrow \texttt{index} \parallel (c)$,
  $\texttt{inv} \leftarrow A_{i,c}^{-1}$,
  $A_{i,i} \leftarrow A_{i,c}$,
  for $k = c+1$ **to** m, $A_{i,k} \xleftarrow{\times} \texttt{inv}$,
  for $j = i+1$ **to** m,
    $\texttt{mul} \leftarrow A_{j,c}$,
    $A_{j,i} \leftarrow \texttt{mul}$,
    for $k = c+1$ **to** m, $A_{j,k} \xleftarrow{-} \texttt{mul} \cdot A_{i,k}$,
1: $(L, U) \leftarrow (I, O)$,
  for $i = 1$ **to** $\texttt{rank}$,
    for $j = i$ **to** m, $L_{j,i} \leftarrow A_{j,i}$,
    $c \leftarrow \texttt{index}_i$,
    $U_{i,c} \leftarrow 1$,
    for $j = c+1$ **to** m, $U_{i,j} \leftarrow A_{i,j}$,
  return $(P, L, U, \texttt{rank}, \texttt{index})$.

Although we define each of (P, L, U) as $\mathbb{F}_q^{m\times m}$, in practice these matrices have some compact representation depending on their form. In fact, L and U are not used until the control reaches label 1, and after label 1, just a portion of updated A is copied to L and U. Moreover, permutation P can be expressed just as an array of row index.

The following `test` algorithm takes $(P, L, U, \texttt{rank}, \texttt{index})$, m-dimensional $\mathbb{F}_q$ column vector $\mathbf{b}$ and some additional information as its arguments, and then decides whether the equation system $A \cdot \mathbf{x} = \mathbf{b}$ is consistent or not, where $(P, L, U, \texttt{rank}, \texttt{index})$ is the return value of the $\texttt{LUdecompose}(A)$. The inverse matrix of L can be relatively easy to derive, because L is an invertible lower triangular matrix, and it can be reused many times once it is obtained.

$\texttt{test}((P, L, U, \texttt{rank}, \texttt{index}), \mathbf{b}, \texttt{cacheR}, R) :=$

if $\texttt{rank} = m$, $\texttt{consistent} \leftarrow 1$,
else,
  if $\texttt{cacheR} = 0$,
    $R \leftarrow L^{-1}$, $\texttt{cacheR} \leftarrow 1$,
  $\mathbf{b}' \leftarrow P \cdot \mathbf{b}$,
  $\texttt{consistent} \leftarrow \bigwedge_{i=\texttt{rank}+1}^{m} (R_i \cdot \mathbf{b}' = 0)$,
return $(\texttt{consistent}, \texttt{cacheR}, R)$.

The following `sample_a_solution` algorithm takes the return value of $\texttt{LUdecompose}(A)$ and m-dimensional $\mathbb{F}_q$ column vector $\mathbf{b}$ as its arguments, and then samples a solution of the linear equation system $A\mathbf{x} = \mathbf{b}$, if it is consistent.

$\texttt{sample\_a\_solution}((P, L, U, \texttt{rank}, \texttt{index}), \mathbf{b}) :=$

$\mathbf{b}^{(1)} \leftarrow P \cdot \mathbf{b}$,
for $i = 1$ **to** $\texttt{rank}$,
  $\mathbf{b}_i^{(2)} \leftarrow \left(\mathbf{b}_i^{(1)} - \sum_{j=1}^{i-1} L_{i,j} \cdot \mathbf{b}_j^{(2)}\right) / L_{i,i}$,
for $i = \texttt{rank} + 1$ **to** m, $\mathbf{b}_i^{(2)} \leftarrow 0$,
for $i = m$ **downto** 1,
  if $\exists j$ s.t. $i = \texttt{index}_j$,
    $\mathbf{x}_i \leftarrow \mathbf{b}_j^{(2)} - \sum_{k=i+1}^{m} U_{j,k} \cdot \mathbf{x}_k$,
  else, $\mathbf{x}_i \xleftarrow{\$} \mathbb{F}_q$,
return $\mathbf{x}$.

3 the QR-UOV signature scheme

3.1 Algorithms

In the following, we denote the random oracle of input Y whose output set is X as $\texttt{Hash}_X(Y)$. The QR-UOV signature scheme Σ consists of the following three probabilistic polynomial-time algorithms (`KeyGen`, `Sign`, `Verify`), which are defined as follows:

$\texttt{KeyGen}(1^\lambda) :=$

$(\texttt{sk}, \texttt{seed}_{\texttt{pk}}) \xleftarrow{\$} \{0,1\}^{2\lambda}$,
$S' \leftarrow \texttt{Hash}_{\mathbb{F}_{q^\ell}^{V\times M}}(\texttt{sk})$,
$(P_{i,1}, P_{i,2})_{i\in[m]} \leftarrow \texttt{Hash}_{\mathbb{F}_{q^\ell}^{V\times(V+M)\times m}}(\texttt{seed}_{\texttt{pk}})$,
2: $(P_{i,3})_{i\in[m]} \leftarrow (-S'^{\top} P_{i,1} S + P_{i,2}^{\top} S' + S'^{\top} P_{i,2})_{i\in[m]}$,
$\texttt{pk} \leftarrow (\texttt{seed}_{\texttt{pk}}, (P_{i,3})_{i\in[m]})$,
return $(\texttt{pk}, \texttt{sk})$.

Note that $P_{i,2}^{\top} S'$ and $S'^{\top} P_{i,2}$ in the formula of label 2 are just transpose of each other.

$\texttt{Sign}(\mathbf{M}, \texttt{pk}, \texttt{sk}) :=$

$(\texttt{seed}_{\texttt{pk}}, (P_{i,3})_{i\in[m]}) \leftarrow \texttt{pk}$,
$(P_{i,1}, P_{i,2})_{i\in[m]} \leftarrow \texttt{Hash}_{\mathbb{F}_{q^\ell}^{V\times(V+M)\times m}}(\texttt{seed}_{\texttt{pk}})$,
$S' \leftarrow \texttt{Hash}_{\mathbb{F}_{q^\ell}^{V\times M}}(\texttt{sk})$,
$(F_{i,1})_{i\in[m]} \leftarrow (P_{i,1})_{i\in[m]}$,

$(F_{i,2})_{i\in[m]} \leftarrow (-P_{i,1}S' + P_{i,2})_{i\in[m]}$,

$Y^\top := (y_1, \ldots, y_v) \xleftarrow{\$} \mathbb{F}_q^v$,

$A \leftarrow (2Y^\top \widetilde{F}_{i,2})_{i\in[m]} \in \mathbb{F}_q^{m\times m}$,

$\mathtt{LUresult} \leftarrow \mathtt{LUdecompose}(A)$,

$\mathbf{c} \leftarrow (Y^\top \widetilde{F}_{i,1} Y)_{i\in[m]} \in \mathbb{F}_q^m$,

$R \leftarrow I$,

3: **repeat**

$r \xleftarrow{\$} \{0,1\}^\lambda$,

$\mathbf{t} \leftarrow \mathtt{Hash}_{\mathbb{F}_q^m}(\mathbf{M}\|r) \in \mathbb{F}_q^m$,

$\mathbf{b} \leftarrow \mathbf{t} - \mathbf{c} \in \mathbb{F}_q^m$,

$\mathtt{cacheR} \leftarrow 0$,

$(\mathtt{consistent}, \mathtt{cacheR}, R)$

$\xleftarrow{\$} \mathtt{test}(\mathtt{LUresult}, \mathbf{b}, \mathtt{cacheR}, R)$,

until consistent,

$X^\top \xleftarrow{\$} \mathtt{sample\_a\_solution}(\mathtt{LUresult}, \mathbf{b})$,

$\mathbf{s} := \begin{pmatrix} Y - \widetilde{S}'X \\ X \end{pmatrix}^\top \in \mathbb{F}_q^n$,

return $\sigma := (r, \mathbf{s})$.

It is known that the expected number of loops in the **repeat** statement of label 3 is only 2.0 [9]. However this expected complexity is an amortized value and in the actual computation, exponential number of loops are required in the rank shortage. This is why we need to tune the LU decomposition in this work. Note that this Sign algorithm is far from the constant-time implementation.

$\mathtt{Verify}(\mathtt{pk}, \mathbf{M}, \sigma) :=$

$(\mathtt{seed}_{\mathtt{pk}}, (P_{i,3})_{i\in[m]}) \leftarrow \mathtt{pk}$,

$(P_{i,1}, P_{i,2})_{i\in[m]} \leftarrow \mathtt{Hash}_{\mathbb{F}_{q^\ell}^{V\times(V+M)\times m}}(\mathtt{seed}_{\mathtt{pk}})$,

$(r, \mathbf{s}) \leftarrow \sigma$,

$\mathbf{t} \leftarrow \mathtt{Hash}_{\mathbb{F}_q^m}(\mathbf{M}\|r) \in \mathbb{F}_q^m$,

$\mathbf{t}' \leftarrow \left(\mathbf{s}\begin{pmatrix} P_{i,1} & P_{i,2} \\ P_{i,2}^\top & P_{i,3} \end{pmatrix}\mathbf{s}^\top\right)_{i\in[m]}$,

return $\mathbf{t} = \mathbf{t}'$.

Because the following Verify consists of relatively simple operations, it is much faster than KeyGen or Sign.

3.2 Parameter Sets

Here we quote the tables of various parameter sets for QR-UOV given by Furue et al. in [3]. The definition of each parameter is given as follows:

q: the order of the finite field,

v: the number of vinegar variables,

m: the number of oil variables and equations,

ℓ: block size.

Table 1: Parameters for QR-UOV satisfying the security level I for the NIST PQC standardization project

(v, m, ℓ)	(a)	(b)	(c)
$q = 7$	$(189, 72, 3)$	$(2409, 99, 33)$	$(650, 80, 10)$
$q = 31$	$(165, 60, 3)$	$(1664, 78, 26)$	$(600, 70, 10)$
$q = 127$	$(156, 54, 3)$	$(1440, 72, 24)$	$(550, 60, 10)$

Table 2: Parameters for QR-UOV satisfying the security level III for the NIST PQC standardization project

(v, m, ℓ)	(a)	(b)	(c)
$q = 7$	$(291, 111, 3)$	$(4640, 160, 40)$	$(1050, 130, 10)$
$q = 31$	$(246, 87, 3)$	$(3783, 117, 39)$	$(890, 100, 10)$
$q = 127$	$(228, 78, 3)$	$(3348, 108, 36)$	$(830, 90, 10)$

Table 3: Parameters for QR-UOV satisfying the security level V for the NIST PQC standardization project

(v, m, ℓ)	(a)	(b)	(c)
$q = 7$	$(411, 162, 3)$	$(7335, 225, 45)$	$(1450, 180, 10)$
$q = 31$	$(324, 114, 3)$	$(4699, 148, 37)$	$(1120, 120, 10)$
$q = 127$	$(306, 105, 3)$	$(6000, 144, 48)$	$(1120, 120, 10)$

4 Implementation

4.1 Parallel computing in x86

Although there are so many environments of parallel computing in x86 that it is difficult to cover them all, we list some typical ones.

- Open Multi-Processing (OpenMP) is an application programming interface which enables parallel programming on some Multiple Instruction, Multiple Data (MIMD) environment. It is available from some programming languages such as C, C++ and FORTRAN. By using OpenMP interface, it is relatively easy to create a program with multiple threads which concurrently run on multiple cores.

- Single Instruction, Multiple Data (SIMD) instruction sets are powerful tools for implementing high-performance cryptography, however in x86 environments there are many number of such instruction sets, i.e. MMX (1996), 3DNow! (1998), SSE (1999), SSE2 (2001), SSE3 (2004), SSE4 (2006), AVX (2008), AVX2 (2013), and so on. Some of the recent ones are listed below.

  - The Advanced Vector Extensions (AVX) and AVX2 are expansions of previous SIMD architecture. They supports YMM vector registers of 256 bits.
  - AVX512 is an expansions of AVX2, which supports ZMM vector registers of 512 bits.
  - The Advanced Matrix Extensions (AMX) is a new SIMD-type extension of x86 instruction set which supports a set of 2-dimensional

registers (tiles) representing (sub-)matrices and some operations on them.

- General-Purpose computing on Graphics Processing Units (GPGPUs) is a kind of parallel processing of general-purpose computation which uses Graphics Processing Units (GPUs). GPGPUs are applied to various fields from deep learning to bitcoin mining. CUDA is the most famous platform for GPGPUs, however there are many other platforms.

4.2 Environment

In this work, we choose OpenMP as the first step of parallel computing in x86, because it is significantly easier to use than other technologies. The experimental environment is as follows :

- CPU: AMD EPYC 7763 2.45GHz (Turbo Boost 3.15GHz) (64 cores, 128 threads) x 1
- Memory: 32GB RDIMM,3200MT/s,
- OS: Linux 5.13.0-28-generic #31 20.04.1-Ubuntu SMP x86_64 GNU/Linux
- Compiler: gcc (Ubuntu 9.4.0-1ubuntu1 20.04) 9.4.0
- Compiler Options: -O3 -fomit-frame-pointer -Wno-unused-result -lcrypto -lm -I. -fopenmp
- Libraries: OpenSSL 1.1.1f 31 Mar 2020

4.3 Experiments

The performance of each algorithm is estimated as follows where $N = 100$.

Step.0: The following algorithms are implemented.

(a) : Random Seed/Message Generation etc,

(b) : `KeyGen`,

(c) : `Sign`,

(d) : `Verify`.

Step.1: Timing data of the following tasks are measured by using time command of bash.

A : $((a)) * N$ times,

B : $((a) + (b)) * N$ times,

C : $((a) + (b) + (c)) * N$ times,

D : $((a) + (b) + (c) + (d)) * N$ times.

Step.2: The performance of each algorithms are estimated as

$$\texttt{time(KeyGen)} = (\texttt{time}(B) - \texttt{time}(A))/N,$$
$$\texttt{time(Sign)} = (\texttt{time}(C) - \texttt{time}(B))/N,$$
$$\texttt{time(Verify)} = (\texttt{time}(D) - \texttt{time}(C))/N.$$

4.4 Generic Code

We investigate the parameter set to find a promising one in terms of performance. To achieve this goal, we prepared a generic code of the QR-UOV which is almost identical to the pseudo code in the previous section. By applying OpenMP to this code, we confirmed the effect of multiple processing. The results of this code are shown in Fig.1 $\sim$ Fig.3.

4.5 Parameter Specific Code

We implement a parameter specific code for $q = 7$, $\ell = 3$. The results of this code are shown in Table.4 $\sim$ Table.6 and Fig.4 $\sim$ Fig.6.

Table 4: performance of QR-UOV I (sec)

$\log_2(n_{\texttt{threads}})$	`KeyGen`	`Sign`	`Verify`
0	0.05135	0.03598	0.00528
1	0.02832	0.02138	0.00516
2	0.01703	0.01402	0.00532
3	0.01121	0.01029	0.00526
4	0.00860	0.00856	0.00521
5	0.00729	0.00789	0.00513
6	0.00671	0.00746	0.00518
7	0.00778	0.00745	0.00585

Table 5: performance of QR-UOV III (sec)

$\log_2(n_{\texttt{threads}})$	`KeyGen`	`Sign`	`Verify`
0	0.27685	0.19146	0.02412
1	0.15046	0.10933	0.02335
2	0.08711	0.06844	0.02351
3	0.05505	0.04825	0.02265
4	0.03878	0.03800	0.02132
5	0.03197	0.03300	0.02091
6	0.02728	0.03014	0.02053
7	0.02701	0.02951	0.02210

Table 6: performance of QR-UOV V (sec)

$\log_2(n_{\texttt{threads}})$	`KeyGen`	`Sign`	`Verify`
0	1.18275	0.80329	0.03016
1	0.62056	0.43968	0.04575
2	0.34517	0.26047	0.05506
3	0.20680	0.17075	0.05696
4	0.13706	0.12510	0.05804
5	0.10262	0.10243	0.05926
6	0.08213	0.09065	0.05908
7	0.08029	0.08809	0.06004

References

[1] W. Beullens, "Breaking rainbow takes a weekend on a laptop," Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part II, ed. Y. Dodis and T. Shrimpton,

Lecture Notes in Computer Science, vol.13508, pp.464–479, Springer, 2022. doi:10.1007/978-3-031-15979-4_16.

[2] J. Ding and D. Schmidt, "Rainbow, a new multivariable polynomial signature scheme," Applied Cryptography and Network Security, Third International Conference, ACNS 2005, New York, NY, USA, June 7-10, 2005, Proceedings, ed. J. Ioannidis, A.D. Keromytis, and M. Yung, Lecture Notes in Computer Science, vol.3531, pp.164–175, 2005. doi:10.1007/11496137_12.

[3] H. Furue, Y. Ikematsu, F. Hoshino, Y. Kiyomura, T. Saito, and T. Takagi, "Secure Parameters for Multivariate Polynomial Signature Scheme QR-UOV." In *Proc. of SCIS 2023 2023 Symposium on Cryptography and Information Security Fukuoka, Japan, Jan. 24 - 27, 2023.* IEICE, 2023.

[4] H. Furue, Y. Ikematsu, Y. Kiyomura, and T. Takagi, "A new variant of unbalanced oil and vinegar using quotient ring: QR-UOV," Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part IV, ed. M. Tibouchi and H. Wang, Lecture Notes in Computer Science, vol.13093, pp.187–217, Springer, 2021. doi:10.1007/978-3-030-92068-5_7.

[5] M.R. Garey and D.S. Johnson, Computers and Intractability; A Guide to the Theory of NP-Completeness, W. H. Freeman & Co., USA, 1990.

[6] S. Kangshen, J.N. Crossley, and A.W.C. Lun, The Nine Chapters on the Mathematical Art, Oxford University Press, Oxford, 1999.

[7] A. Kipnis, J. Patarin, and L. Goubin, "Unbalanced oil and vinegar signature schemes," Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding, ed. J. Stern, Lecture Notes in Computer Science, vol.1592, pp.206–222, Springer, 1999. doi:10.1007/3-540-48910-X_15.

[8] NIST, "Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process." Cryptology ePrint Archive, Paper 2022/434, 2016. URL: https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf.

[9] K. Sakumoto, T. Shirai, and H. Hiwatari, "On provable security of UOV and HFE signature schemes against chosen-message attack," Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011, Taipei, Taiwan, November 29 - December 2, 2011. Proceedings, ed. B. Yang, Lecture Notes in Computer Science, vol.7071, pp.68–82, Springer, 2011. doi:10.1007/978-3-642-25405-5_5.

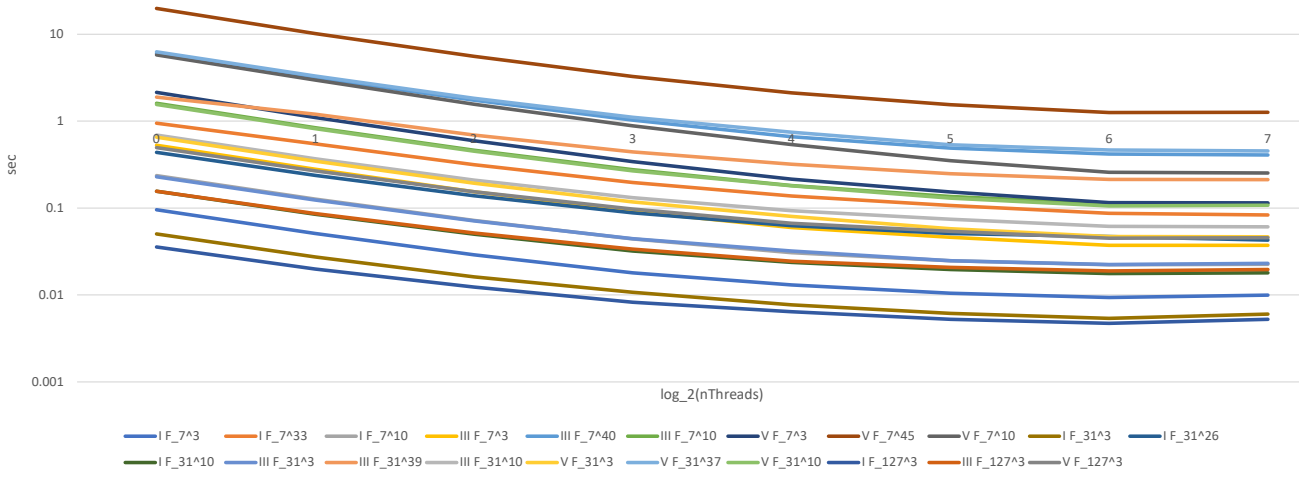

Fig 1: Performance of `KeyGen` (generic code)

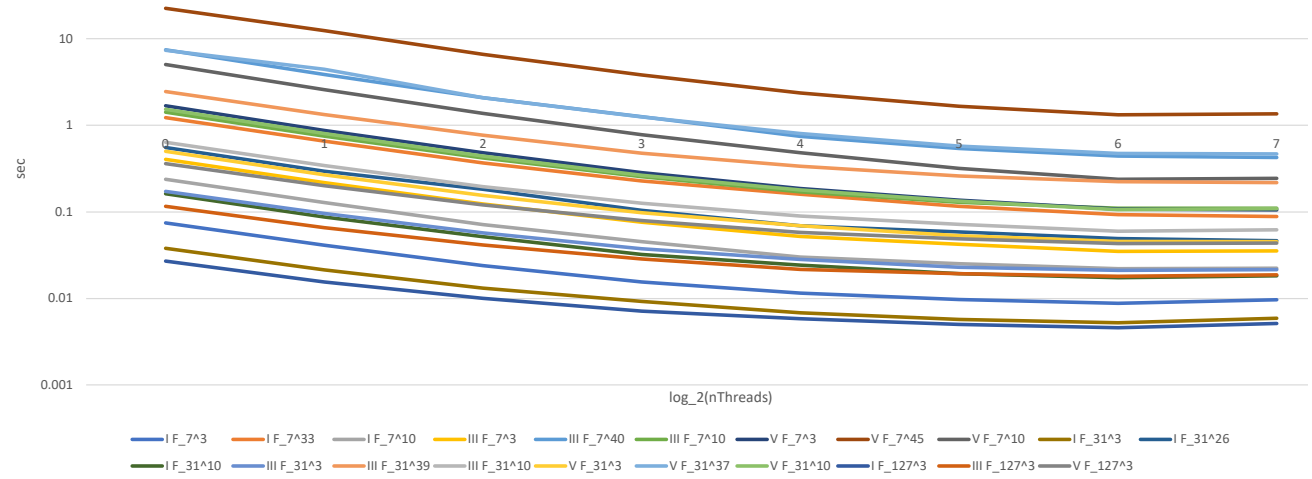

Fig 2: Performance of `Sign` (generic code)

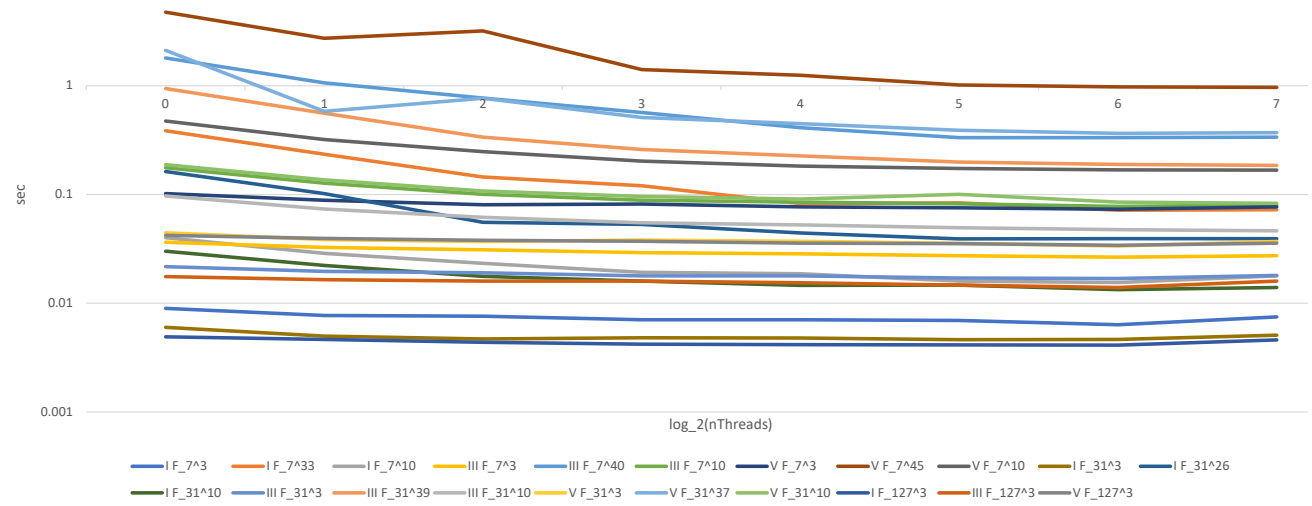

Fig 3: Performance of `Verify` (generic code)

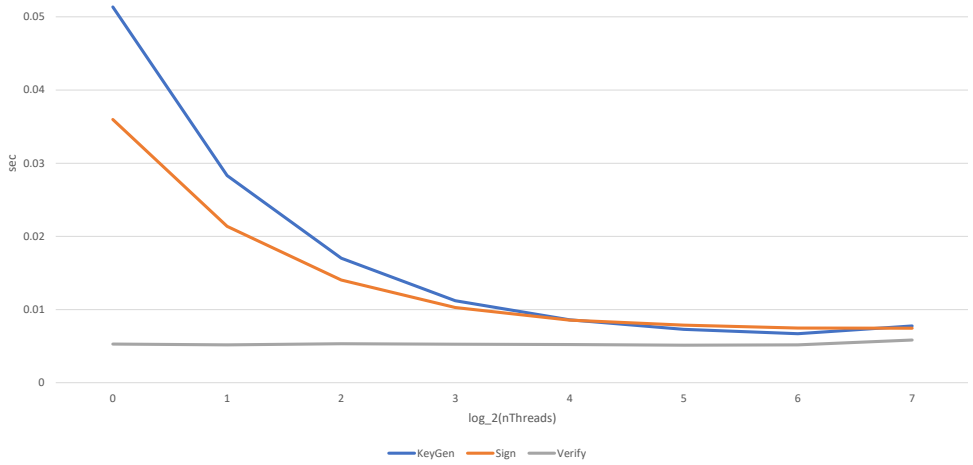

Fig 4: Performance of QR-UOV I (parameter specific code)

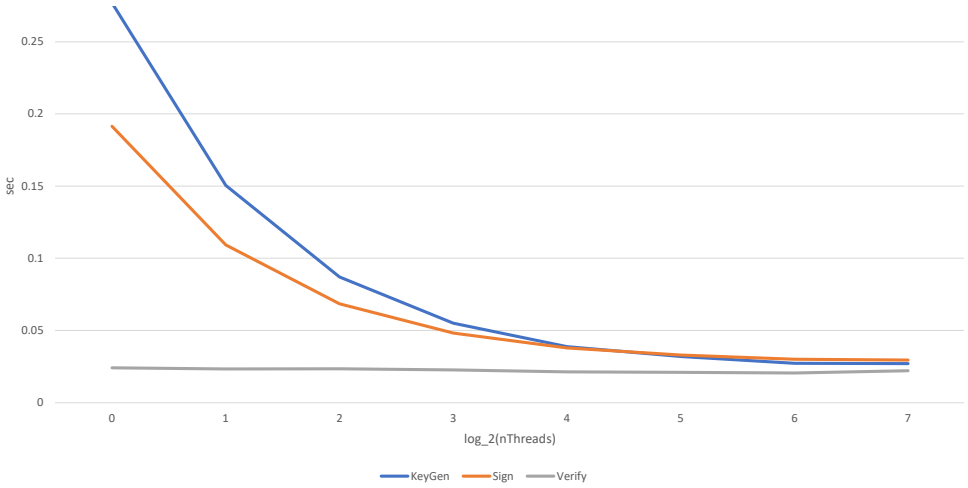

Fig 5: Performance of QR-UOV III (parameter specific code)

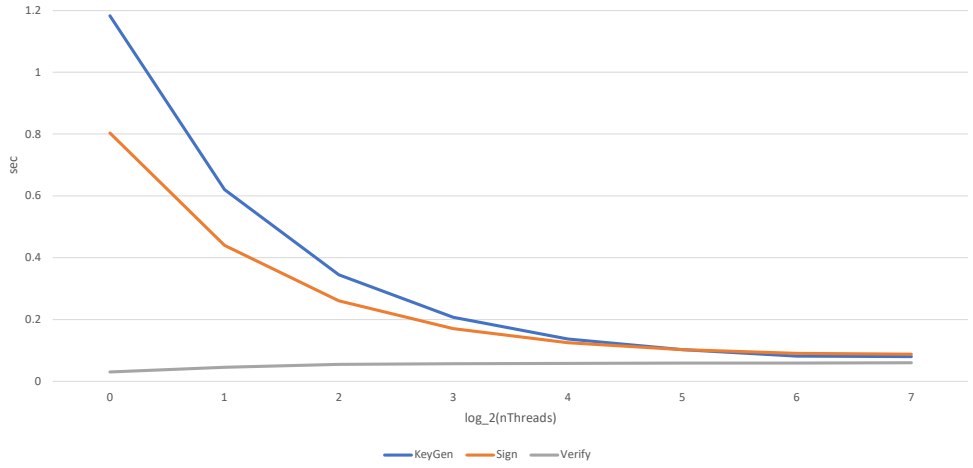

Fig 6: Performance of QR-UOV V (parameter specific code)