Fast Implementation of Simple Matrix Encryption Scheme on Modern x64 CPU

Zhiniang Peng, Shaohua Tang(✉), Ju Chen, Chen Wu, and Xinglin Zhang

School of Computer Science & Engineering, South China University of Technology, Guangzhou 510006, China
shtang@IEEE.org, csshtang@scut.edu.cn

Abstract. The simple matrix encryption scheme (SMES) is one of the very few existing multivariate public key encryption schemes. However, it is considered impractical because of high decryption failure probability. There exist some ways to reduce the decryption failure probability, but all of them will result in serious performance degradation. In this paper, we solve this dilemma by exploiting the power of modern x64 CPU. SIMD and several software optimization techniques are used to improve the efficiency. The experimental results show that our implementation is three orders of magnitude faster than the existing Rectangular SMES implementation under a similar decryption failure probability and it's comparable to the fastest Ring-LWE and RSA implementations.

Keywords: AVX2 · Simple matrix encryption · Post-quantum cryptosystem · Implementation · MPKC

1 Introduction

In [24,25], Shor proposed some polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. It posed a serious threat to the existing cryptographic schemes such as RSA and ECC, which are based on those problems. After that, Post-Quantum Cryptography [4,9], which is secure against attacks by quantum computer, became a very important research area. Multivariate Public Key Cryptography (MPKC) is one of the most promising candidates in Post-Quantum Cryptography.

Since the first MPKC scheme: MI [18] was proposed in 1988, this area has undergone a rapid development in last two or three decades. A lot of MPKC encryption and signature schemes have been proposed (e.g., TTS [32], MQQ [15], SMES [28], HFE [19], ZHFE [22]). However, most of them were broken by various attacks, such as MinRank [32], High Rank attack [13,32], Direct attack, Differential attack [13] and Rainbow Band Separation attack [13,30]. SMES [28] is one of the very few existing multivariate public key encryption scheme. None of the existing attacks can cause severe security threats to it.

However, SMES is not yet widely used, mainly because its high decryption failure probability. Its decryption failure probability is inversely proportional

© Springer International Publishing AG 2016
F. Bao et al. (Eds.): ISPEC 2016, LNCS 10060, pp. 151–166, 2016.
DOI: 10.1007/978-3-319-49151-6_11

to the order of its basic field. In order to get a reasonable decryption failure probability, we must choose a very large finite field. But this always results in serious performance degradation because operations in large finite field are very inefficient.

An improved SMES called Rectangular SMES (RSMES) was proposed in [29] to reduce the decryption failure probability. But RSMES will increase the computational complexity of basic SMES. Another variant of SMES called Tensor SMES (TSMES) to eliminate the decryption failure was proposed in [21]. However, the security of TSMES is weaker than that of the basic SMES.

Our Results: In this paper, we exploit the power of modern x64 CPU to give a high performance SMES implementation with low decryption failure probability. Here are the main contributions of this paper:

(1) We choose the large prime field $GF(2^{31}-1)$ as our base field to reduce decryption failure probability of SMES, and carefully analyse its behavior against Direct attack.
(2) We give fast SIMD arithmetic operations over $GF(2^{31}-1)$ by using AVX2 instruction set in modern CPU. This is the first time a large prime field is used to implement MPKC schemes.
(3) Our experiments show that the memory latency problem is a main bottleneck of MPKC schemes in modern CPU. We propose several software optimization techniques to break through this bottleneck. Our techniques can also be applied to other MPKC schemes.
(4) Our implementation is three orders of magnitude faster than the existing RSMES implementation. It is comparable to RSA implemented in OpenSSL [1] and the fastest Ring-LWE implementation. This shows that MPKC encryption schemes are still promising candidates for Post-Quantum Cryptography.

2 Simple Matrix Encryption Scheme

In this section, we give a description of SMES and it's variants.

2.1 Basic SMES

We first give a general description of basic SMES.

Key Generation: According to the required security level, we choose the appropriate set of parameters including finite field $K = GF(q)$, $s \in N$. We set $n = s^2$ and $m = 2n$, define three $s \times s$ matrices A, B and C of the form:

$$A = \begin{pmatrix} x_1 & \cdots & x_s \\ \vdots & & \vdots \\ x_{(s-1)s+1} & \cdots & x_n \end{pmatrix}, B = \begin{pmatrix} b_1 & \cdots & b_s \\ \vdots & & \vdots \\ b_{(s-1)s+1} & \cdots & b_n \end{pmatrix}, C = \begin{pmatrix} c_1 & \cdots & c_s \\ \vdots & & \vdots \\ c_{(s-1)s+1} & \cdots & c_n \end{pmatrix}.$$

Here $x_1, \cdots, x_n$ are linear monomials of multivariate polynomial ring $F[x_1, \cdots, x_n]$. $b_1, \cdots, b_n$ and $c_1, \cdots, c_n$ are random linear combinations of $x_1, \cdots, x_n$. Let $E_1 = AB$ and $E_2 = AC$. The central map F of the scheme consists of the m components of E_1 and E_2.

We then choose two random invertible linear maps $S : K^m \to K^m$ and $T : K^n \to K^n$, and compute public key of the scheme $P = S \circ F \circ T$: $K^n \to K^m$. The private key consists of the matrices B and C and the linear maps S and T.

Encryption: To encrypt a message $\mathbf{m} \in K^n$, we simply compute the ciphertext $\mathbf{c} = P(\mathbf{m}) \in K^m$.

Decryption: To decrypt a ciphertext $\mathbf{c} \in K^m$, we have to perform the following three steps.

(1) Compute $\mathbf{y} = S^{-1}(\mathbf{c})$. Write the elements of the vector $\mathbf{y}$ into matrices $\hat{E_1}$ and $\hat{E_2}$ as follows:

$$\hat{E_1} = \begin{pmatrix} y_1 & \cdots & y_s \\ \vdots & & \vdots \\ y_{(s-1)s+1} & \cdots & y_n \end{pmatrix}, \hat{E_2} = \begin{pmatrix} y_{n+1} & \cdots & y_{n+s} \\ \vdots & & \vdots \\ y_{n+(s-1)s+1} & \cdots & y_m \end{pmatrix}.$$

(2) Invert the central map $F(\mathbf{x}) = \mathbf{y}$. To do this, we consider the following four cases:
- If $\hat{E_1}$ is invertible, use the polynomial matrix equation $B \cdot \hat{E_1}^{-1} \cdot \hat{E_2} - C = 0$ to get n linear equations in n variables $x_1, \cdots, x_n$.
- If $\hat{E_1}$ is not invertible, but $\hat{E_2}$ is invertible, use the polynomial matrix equation $C \cdot \hat{E_2}^{-1} \cdot \hat{E_1} - B = 0$ to get n linear equations in the n variables.
- If none of $\hat{E_1}$ or $\hat{E_2}$ is invertible, but $\hat{A} = A(\mathbf{x})$ is invertible, use the relations $\hat{A}^{-1} \cdot \hat{E_1} - B = 0$ and $\hat{A}^{-1} \cdot \hat{E_2} - C = 0$ to get a linear system.
- If none of $\hat{E_1}$, $\hat{E_2}$ and $\hat{A}$ is invertible, there occurs a decryption failure.

(3) Compute the plaintext by $\mathbf{m} = T^{-1}(x_1, \cdots, x_n)$.

2.2 SMES Variants

SMES is one of the very few existing approaches to create secure encryption on the basis of multivariate polynomials. However, to invert correctly, the matrix $\hat{A}$ must be invertible. If $\hat{A}$ is not invertible, there will be a decryption failure. As $\hat{A}$ is a random matrix over $GF(q)$, the probability that $\hat{A}$ is invertible is

$$1 - (1 - \frac{1}{q^s})(1 - \frac{1}{q^{s-1}}) \cdots (1 - \frac{1}{q}) \approx \frac{1}{q}.$$

We can estimate the decryption failure probability by $\frac{1}{q}$. To reduce the decryption failure probability, an improved SMES scheme call RSMES was proposed in [29]. The decryption failure probability of RSMES is reduced to

$$1 - (1 - \frac{1}{q^s})(1 - \frac{1}{q^{s-1}}) \cdots (1 - \frac{1}{q^{s-r+1}}) \approx \frac{1}{q^{s-r+1}}.$$

But RSMES has larger parameters than basic SMES. What's more, we need to solve a system of m quadratic equations during the decryption. This will increase the computational complexity of decryption.

TSMES with no decryption failure was proposed in [21]. The idea is that one uses a tensor product of two small matrices as the affine transformation T. This will enable the sender to check the decryptability of his/her plaintexts without knowing the secret key. However, this scheme is much weaker than the basic SMES. Hashimoto showed that TSMES is equivalent to a weak example of SMES in [17]. It's security may be threatened by UOV reconciliation attack [13].

3 Fast Arithmetic Operations in $GF(2^{31}-1)$

Due to the decryption failure probability, we have to choose a large field as our base field. However, large fields are always considered inefficient. Since lots of additions and multiplications in base field need to be done during encryption and decryption, base field with faster arithmetic operations is of great importance in SMES implementation.

As we know, almost all the existing MPKC implementations [8,11,20,23] use small fields to achieve fast arithmetic operations. In small field, arithmetic operations such as inversion and multiplication can be done by using small look-up tables.

In this paper, we choose the Mersenne prime $2^{31}-1$ as the order of our base field. It is large enough to get a reasonable decryption failure probability. In addition, it will admit faster arithmetic operations by exploiting the power of modern x64 CPU.

3.1 Fast Multiplication

The most important step of multiplication in large prime field is modular arithmetic in integer. Under normal circumstances, one always use $a = a - p * \left\lfloor \frac{a}{p} \right\rfloor$ to reduce a positive integer a into $[0, p-1]$. But integer division is slow in modern CPU. Barrett method [3] and Montgomery method [7] may speed up this procedure, but it's still relatively slow.

In the case of Mersenne prime, we can get faster modulo operation which exploits the special structure of Mersenne prime. Arithmetic operation modulo a Mersenne prime can be done by using well known shift-and-add procedure. For $p = 2^q - 1$, we can do $a = (a\&p) + (a >> q)$ a few times to reduce a positive a into $[0, p-1]$. Compared with regular method, this is much faster in a modern CPU.

When multiplying two field elements a and b, we first use integer multiplication to get $c = int(a) * int(b)$. Because $int(a)$ and $int(b)$ is less than 2^{31}, c can be represented in $uint64$ without overflow. Integer multiplication instruction in x64 CPU can be used to compute the result. Then we do shift-and-add reduction no more than twice, we can get $a * b$ in the base field.

To fully exploit the power of modern CPU, we adapt shift-and-add modular arithmetic to the SIMD mode by using AVX2 instruction set. We pack 4 64-bit integers into a AVX2 type integer $\_\_m256i$. Then we can do 4 field multiplications in the meantime. Algorithm 1 describes our vectorized field multiplication.

Algorithm 1. Vectorized field multiplication algorithm.

1: **procedure** VMUL($\boldsymbol{a}$,$\boldsymbol{b}$) ▷ $\boldsymbol{a}$ and $\boldsymbol{b}$ are __m256i type
2: $\boldsymbol{c} = \_mm256\_mul\_epu32(\boldsymbol{a}, \boldsymbol{b})$
3: $\boldsymbol{clo} = \_mm256\_and\_si256(\boldsymbol{c}, \boldsymbol{p})$
4: $\boldsymbol{chi} = \_mm256\_srli\_epi64(\boldsymbol{c}, 31)$
5: $\boldsymbol{c} = \_mm256\_add\_epi64(\boldsymbol{alo}, \boldsymbol{ahi})$
6: **return** $\_mm\_min\_epu64(\boldsymbol{c}, \_mm\_sub\_epi32(\boldsymbol{c}, \boldsymbol{p}))$

3.2 Fast Inversion

Finding the multiplicative inverse of a field element is the most costly operation in finite field. Extended Euclidean Algorithm (EEA) and Binary Extended Euclidean Algorithm (BEEA) are always used to compute the inverse in $GF(p)$. But they are not suitable when dealing with Mersenne prime since they don't exploit the structure of special modulus.

In [31], Thomas et al. proposed an efficient algorithm to calculate multiplicative inverse over $GF(p)$ when p is a Mersenne prime. The key idea of their algorithm is that $\lfloor \frac{p}{z} \rfloor$ can be easily computed when p is a Mersenne prime. However, this algorithm can't take advantage of modern x64 instructions, because it needs roughly q iterations to compute $\lfloor \frac{p}{z} \rfloor$. Instead, we find that there exists another interesting inversion algorithm called Relational Reduction Algorithm (RRA) mentioned in [10]. We implement a slightly modified Relational Reduction Algorithm 2 using x64 instructions. This algorithm can exploit the special form of the modulus with some low cost CPU instructions.

One may notice that TZCNT instruction is used in our algorithm. TZCNT instruction is a x64 CPU instruction to count the number of consecutive zero bits on the right of its operand and it takes less than 3 CPU cycles to execute.

3.3 Timing

Mul-and-add operation and inversion operation are the most important operations in SMES. Tables 1 and 2 display timings for them when using different algorithm. Here we do not consider the time to read and write memory. Timings are average number of clock cycles in an Intel Core i7-4790 when applying the arithmetic operations in CPU register.

The column "Naive" in Table 1 corresponds to the scalar approach using the C++ operator % to compute remainder. In fact, the compiler will use method mentioned in [16] to optimize modular reduction.

Algorithm 2. Field inversion algorithm.

```
1: procedure INV(x)                          ▷ The inversion of x
2:     (a, b) = (1, 0) and (y, z) = (x, p)
3:     e = TZCNT(y)                          ▷ x64 instruction
4:     y = y >> e
5:     a = a << (q − e)
6:     a = (a&p) + (a >> q)
7:     if y == 1 then
8:         return a                          ▷ The inversion is a
9:     (a, b) = (a + b, a − b)
10:    (y, z) = (y + z, y − z)
11:    Goto 3
```

Table 1. Mul-and-add in CPU clock cycles.

	Naive	Shift-and-add	AVX2
Mul-and-add	4.49	1.83	0.54

Table 2. Inversion in CPU clock cycles.

	EEA	BEEA	RRA
Inversion	915.2	760.8	501.6

From the timing results, we can see that our basic field operations are much faster than others. Although the inversion operation is still relatively slow, This will not affect our choice for $GF(2^{31}-1)$. Because only a few inversions are needed in SMES and they don't need to be vectorized.

4 SIMD Algorithms for SMES

The standard SMES encryption and decryption algorithms involve polynomial matrix multiplication as well as polynomial evaluation, which seem inappropriate for SIMD computing. In this section, we give alternative encryption and decryption algorithms which can benefit from SIMD computing.

4.1 Decryption

As described in Sect. 2.1, we need to compute two affine transformations S^{-1} and T^{-1}, and invert the central map F to decrypt a ciphertext. Affine transformation is just matrix vector multiplication which is suitable for SIMD computing. However, to invert the central map F, we need to perform polynomial matrix multiplication to set up linear equations. This may involve complex computations.

In fact, polynomial matrix multiplication can be avoided. Here we give an alternative algorithm to form the central linear equations using linear algebra. Instead of viewing polynomial matrix B and C as an matrix over polynomial ring $F[x_1, \cdots, x_n]$, we can write it as the following equations:

$$B = \sum_{0}^{n} B_i \cdot x_i, \quad C = \sum_{0}^{n} C_i \cdot x_i \,.$$

Here B_i and C_i are matrices over the base field. $x_1, \cdots, x_n$ are linear monomials of multivariate polynomial ring $F[x_1, \cdots, x_n]$. $x_0 = 1$, which stands for linear part. To form a linear system, we consider the following cases:

Case 1: If $\hat{E}_1$ is invertible, we compute $\hat{E} = {\hat{E}_1}^{-1} \cdot \hat{E}_2$ and $G_i = B_i \cdot \hat{E} - C_i$. Write elements of matrix $G_i \in F^{s \times s}$ into vector as follows:

$$G_i = \begin{pmatrix} {g_1}^i & \cdots {g_s}^i \\ \vdots & \vdots \\ {g_{(s-1)s+1}}^i \cdots {g_n}^i \end{pmatrix} \rightarrow vector(G_i) = \mathbf{g_i} = ({g_1}^i, {g_2}^i, ..., g_n^i)^T.$$

Then we get a linear system $G \cdot \mathbf{x} = \mathbf{g_0}$:

$$\begin{pmatrix} {g_1}^1 & {g_1}^2 & \cdots & {g_1}^n \\ {g_2}^1 & {g_2}^2 & \cdots & {g_2}^n \\ \vdots & \vdots & \vdots & \vdots \\ {g_n}^1 & {g_n}^2 & \cdots & {g_n}^n \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} {g_1}^0 \\ {g_2}^0 \\ \vdots \\ {g_n}^0 \end{pmatrix}.$$

Case 2: If $\hat{E}_2$ or $\hat{A}$ is invertible, we can use similar method to get a linear system.

Case 3: If none of $\hat{E}_1$, $\hat{E}_2$ or $\hat{A}$ is invertible, there occurs a decryption failure. The decryption failure probability of SMES over our base field is approximately 2^{-31}.

After getting the linear system, we can use Gauss Elimination to solve it. Then all the computations in SMES decryption can be sped up by SIMD computing.

4.2 Encryption

To encrypt a message $\mathbf{m} \in K^n$, we simply evaluate the m public multivariate quadratic polynomials $\mathbf{y} = P(\mathbf{x})$.

In this paper we use matrix vector multiplication to evaluate the polynomials. To compute $\mathbf{y} = P(\mathbf{x})$, we first compute all quadratic monomials $x_i x_j$. Then we compute $\mathbf{y} = \mathbf{P} \cdot \mathbf{X}$ as follows:

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix} = \begin{pmatrix} p_{11}^1 \; p_{12}^1 \cdots p_{nn}^1 \; p_1^1 \; \cdots \; p_n^1 \; p_0^1 \\ p_{11}^2 \; p_{12}^2 \cdots p_{nn}^2 \; p_1^2 \; \cdots \; p_n^2 \; p_0^2 \\ \vdots \\ p_{11}^m \; p_{12}^m \cdots p_{nn}^m \; p_1^m \cdots p_n^m \; p_0^m \end{pmatrix} \cdot \begin{pmatrix} x_1 x_1 \\ \vdots \\ x_1 x_n \\ x_1 x_2 \\ \vdots \\ x_n x_n \\ \vdots \\ x_n \\ x_0 \\ 1 \end{pmatrix}.$$

Let $v = \frac{(n+1)(n+2)}{2}$. Matrix $\mathbf{P}$ is the m-by-v Macaulay matrix of the m quadratic polynomials. Monomials vector $\mathbf{X}$ is a v dimension column vector. To compute $\mathbf{y}$, mv additions and $mv + \frac{(n)(n+1)}{2}$ multiplications are needed. Comparing with regular polynomial evaluation method, this procedure reduces the total arithmetic operations by 3 %. What's more, it can take advantage of SIMD computing.

5 Optimization

In this section, we focus on optimizing Gauss Elimination (GE) and matrix vector multiplication (MVM) for SMES. They are the most expensive parts of SMES. All our codes are compiled by Intel Compiler C++ 2016 using Highest Optimizations (/O3) and Favor fast code (/Ot) options. All our experiments are carried on an Intel Core i7-4790 @3.60 Ghz CPU with TurboBoost disabled.

5.1 CPU Bottlenecks

We implement MVM and GE by using our fast field operations and standard C++ code. However, the speed is not fast enough for cryptography. As the arithmetic operations in our base field are pretty fast, the performance is limited by other bottlenecks in CPU.

Although MPKC schemes are considered theoretically faster than traditional asymmetric cryptosystem such as RSA and ECC, we always get poor performance when we implement them in CPU. This is because MPKC schemes have larger key size. CPU spent more time in reading it from memory rather than doing actual computations. Lacking of good memory access patterns in MPKC schemes will result in serious speed penalty.

For better performance, we have to optimize our codes for our CPU.

5.2 Hybrid Representation

In Sect. 3.1, we use 64-bit integer to represent a field element. One may wonder why not use 32-bit integer. This is because integer overflow will cause wrong result if we use 32-bit integer for field multiplication. In the meantime, 64-bit representation allows us to use lazy modular reduction technique. This will save a lot of computation time.

But if we store elements in 64-bit integer, there will be 32 zero in its high 32 bits. This will cost redundant storage. What's more, it enlarges the working set size. Half of the memory reading operations will be done for nothing.

A better idea is to store the elements in 32-bit representation, and convert them to 64-bit representation during computing. We can use AVX2 intrinsic $\_mm256\_unpacklo\_epi32$ and $\_mm256\_unpackhi\_epi32$ to convert 8 packed elements in 32-bit representation to 8 packed elements in 64-bit representation. This simple trick can reduce the working set size as well as the cache miss rate.

5.3 Loop Unrolling

During the SMES computation, a lot of matrix operations need to be done. They are always handled by loop statement in C++. MVM and GE are also handled by loop statement. In every inner iteration of MVM and GE, the computation task contains only one SIMD mul-and-add operation. The loop control overhead including index increment and branch test are relatively large.

In this paper, we use loop unrolling technique to optimize program running speed at the expense of its binary size. This is known as space-time tradeoff. Sometimes compilers will use loop unrolling technique automatically, but in our case it seems to not be optimized even with Highest Optimizations (/O3) option. In this paper, we will not totally unroll the loops. Our experiments show that loop with stripe equal to 4 can get the best space-time tradeoff.

5.4 Lazy Modular Reduction

MVM and GE need a lot of additions. To compute $a+b$ in the base field, we first do integer addition: $int(a)+int(b)$, then reduce the result into $[0, p-1]$. As we use 64-bit representation during computation, $int(a)$ and $int(b)$ is far less than 2^{64}. No overflow will occur during integer addition. Besides, $int(a)*int(b)$ is strictly less than $(2^{31}-1)^2$, we can do only 1 modular reduction for 4 mul-and-add operations, which can save a lot of time.

5.5 Pipeline Optimization

Nowadays CPU use out-of-order execution technique to avoid a class of stalls that occur when data needed to perform an operation are unavailable. It can increase memory latency tolerance of CPU. However, this can only work with nearby operations. If the memory latency exceeds the tolerance of CPU, it will still create CPU stalls.

In this paper we use Intel VTune Amplifier 2016 to analyse our code. It can provide us accurate data in our CPU, such as cache misses, branch mispredictions, CPU stalls for each opcode and other hardware issues. By the help of Intel VTune Amplifier, we reduce the unnecessary memory operations and reorder our code in a larger range than CPU automatic out-of-order execution technique does. This will help us to reduce unnecessary CPU stalls.

5.6 Splitting Technique

After applying all of the above optimization method, we measure the CPO (cycles per mul-and-add operation) of MVM. For $n < 89$, the CPO seems to be a constant number 0.58. This is extremely close to the theoretical optimum. However, the CPO become larger when $n > 89$. This is because our CPU uses the least recently used eviction policy to manage cache. Every element of $\mathbf{X}$ is used for m times. During the computation, CPU read element from $\mathbf{P}$ and $\mathbf{X}$, and fetch it into local L1 cache. When $n > 89$, the size of $\mathbf{X}$ is larger than 16 kB

(half of the local L1 cache size), CPU will evict elements of $\mathbf{X}$ from L1 cache before it is reused. This will enlarge L1 cache miss penalty.

To solve this problem, we use splitting technique to get better cache performance. If $\mathbf{v}$ is larger than 16kB, then we divide $P \cdot \mathbf{X}$ as follows:

$$\left(\mathbf{P_1} \cdots \mathbf{P_n}\right) \cdot \begin{pmatrix} \mathbf{x_1} \\ \vdots \\ \mathbf{x_n} \end{pmatrix} = \sum_{i=1}^{n} \left(\mathbf{P_i}\right) \cdot \left(\mathbf{x_i}\right).$$

Each $\mathbf{x_i}$ is less than 16kB. This splitting technique can reduce the L1 cache miss penalty for $\mathbf{X}$. To get a better cache performance with this technique, entries of $\mathbf{P}$ must be placed in accordance with the $\mathbf{P_i}$ sequence. As $\mathbf{P}$ is fixed in SMES, we can reorder its data layout from the beginning. In our experiments, this divide and conquer technique can reduce L1 cache miss rate and the CPO growth rate when $n > 89$. Besides, we also use software prefetch intrinsic in our code.

5.7 Experiment Results

After applying all the optimization we mentioned, we measure the CPO behavior of MVM and GE (without counting the cycles of inversion operations). The results are shown in Fig. 1.

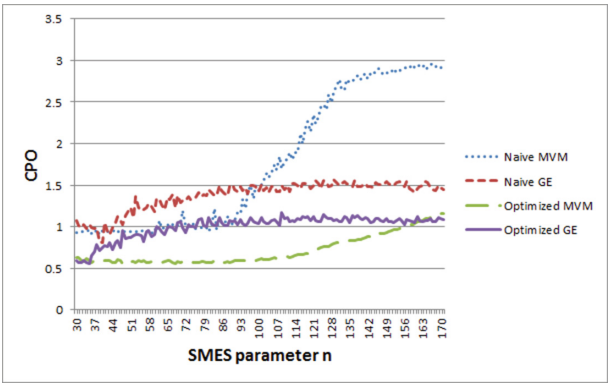

Fig. 1. CPO of our MVM and GE.

As MVM is more suitable for SIMD computing, it has better CPO than GE when n is small. But when n become larger, CPO of MVM increases faster than CPO of GE. This is because the cache miss penalty of MVM grows faster than GE.

Compared with the naive implementation, our optimized one is much better. For MVM, we almost reach the theoretical optimum in low dimension case. For GE, we decrease the CPO by a factor of 1.4. However, memory latency problem still exists in large dimension, but its impact has been reduced a lot.

6 Results for SMES

In this section, we choose security parameters for SMES over our base field and compare our SMES implementation with other encryption schemes.

6.1 Choosing Parameters

To achieve the security requirements, we need to choose security parameters for SMES over $GF(2^{31}-1)$ to block all known attacks. The complexity of Rank attack against SMES is $(n\begin{pmatrix} m+s \\ s \end{pmatrix}+m+1)^3$ and the complexity of High Order Linearization Equation attack against SMES is $O(p^{\lceil \frac{m}{n} \rceil 2s}m^3)$. We can choose our parameters by these complexity formulas.

However, there is no formula for the complexity of Direct attack against SMES. To better estimate the complexity of Direct attack against SMES, we carried out a number of experiments with MAGMA [6], which contains an efficient implementation of F4 algorithm [14] for computing Gröbner bases [27]. As SMES public key system is an overdefined quadratic system with n variables and $2n$ equations, we measure the running time of F4 algorithm for SMES and random overdefined quadratic system over $GF(2^{31}-1)$. The results are presented in Fig. 2.

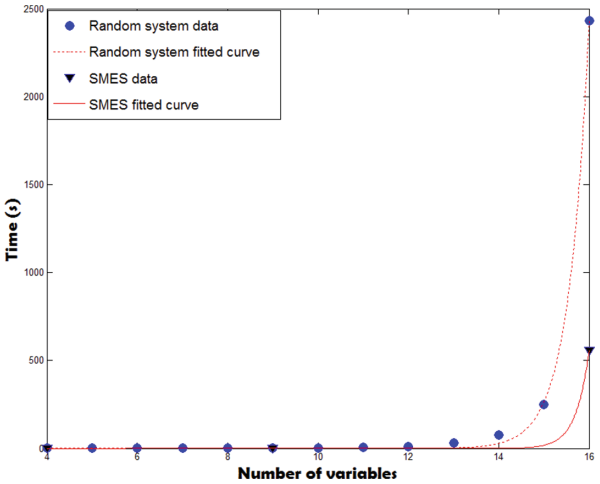

Fig. 2. Fitted curves for computational complexity of F4 algorithm against random system and SMES.

From the experiments, we find that the bit complexity of solving a random overdefined multivariate quadratic system of $2n$ equations in n variables directly is roughly given by:

$$1.60 \cdot n + 14.4$$

for systems over $GF(2^{31}-1)$. The SMES quadratic system is obviously easier to solve than random system, but it's still exponential hard. The bit complexity of solving a SMES overdefined multivariate quadratic system of $2n$ equations in n variables directly is roughly given by:

$$1.52 \cdot n + 14.5$$

for systems over $GF(2^{31}-1)$.

Then we use the previously mentioned formulas to derive security parameters for the SMES to prevent all known attacks. As our base field is bigger, Rank attack and Direct attack have higher complexity against our instances. So we can choose s slightly smaller. We present our parameter sets for SMES over $GF(2^{31}-1)$ in Table 3.

Table 3. Parameters of SMES for different levels of security over $GF(2^{31}-1)$.

Security (bit)	Parameters (s,n,m)	Plaintext size (bit)	Ciphertext size (bit)	Public key size (kB)	Private key size(kB)
80	(7,49,98)	1519	3038	472.8	64.2
112	(8,64,128)	1984	3968	1039.0	109.2
128	(9,81,162)	2511	5022	2086.2	174.7
160	(10,100,200)	3100	6200	3897.5	266.0

6.2 Comparing with the Existing RSMES Implementation

We use the techniques according to Sect. 4 to implement SMES. We first compare our result with RSMES implementation proposed in [29].

Table 4. A comparison with RSMES.

	Parameters (K,n,m)	Security (bit)	Encryption cycles (10^3)	Decryption cycles (10^3)	Probability of decryption failure
RSMES	$(GF(2^8),128,264)$	80	48000	60000	2^{-32}
	$(GF(2^8),364,182)$	100	134000	149000	2^{-32}
SMES	$(GF(2^{31}-1),49,98)$	80	74.7	85.8	$\approx 2^{-31}$
	$(GF(2^{31}-1),64,128)$	112	163.5	140.9	$\approx 2^{-31}$

From Table 4 we can see that our implementation is almost three orders of magnitude faster than the existing RSMES implementation under a similar decryption failure probability. This sounds incredible, but it's really reasonable. There are four reasons:

(1) Our base field $GF(2^{31}-1)$ provides stronger security. This enables us to choose smaller n and m. As the computational complexity of encryption and decryption grow with cube of n, smaller n make our implementation much faster.
(2) We choose $GF(2^{31}-1)$ to reduce the probability of decryption failure, but RSMES use rectangular construction to reduce it. This makes our decryption algorithm faster and our parameters more flexible.
(3) We exploit the power of modern CPU to get faster $GF(2^{31}-1)$ arithmetic operations, which is faster than calling arithmetic functions in some libraries such as NTL [26]. In addition, AVX2 instructions can improve the speed even further.
(4) Our code is optimized for our CPU. We use several techniques to solve the memory latency problem and other bottlenecks in implementation.

The choice of $GF(2^{31}-1)$ actually gives us a win-win situation.

6.3 Comparing with RSA and Ring-LWE

In this section, we compare our implementation with the fastest RSA implementation and Ring-LWE implementation. For RSA, we choose the RSA parameters according to the latest NIST key management recommendation [2]. To get a fair comparison, we get the benchmark results of RSA implemented in OpenSSL from eBATS (ECRYPT Benchmarking of Asymmetric Systems)[5]. For Ring-LWE encryption, we choose the fastest known result in [12]. Table 5 shows an overall comparison.

Table 5. Comparing with RSA and Ring-LWE.

	Parameters	Security (bit)	Encryption cycles(10^3)	Decryption cycles(10^3)	Ciphertext expansion
Ring-LWE	Ring-LWE256	80	121.2	43.3	26
	Ring-LWE512	112	261.9	96.5	28
RSA	Ronald1024	80	73.4	1434.7	1
	Ronald2048	112	140.1	5588.7	1
	Ronald3072	128	209.5	14958.3	1
	Ronald4096	160	300.2	31820.3	1
SMES	SMES49	80	74.7	85.8	2
	SMES64	112	163.5	140.9	2
	SMES81	128	319.9	241.6	2
	SMES100	160	624.3	424.7	2

Compared with Ring-LWE, our SMES implementation is better at encryption time and ciphertext expansion, but the Ring-LWE is better at decryption time.

Compared with RSA, our SMES implementation is better at decryption time, but RSA is better at encryption time and ciphertext expansion.

We admit that SMES have larger key size than RSA and Ring-LWE, but this is not a bottleneck for modern computer. Besides, we can see from Fig. 3 that SMES outperforms RSA and Ring-LWE in throughput, which is of great importance in some applications.

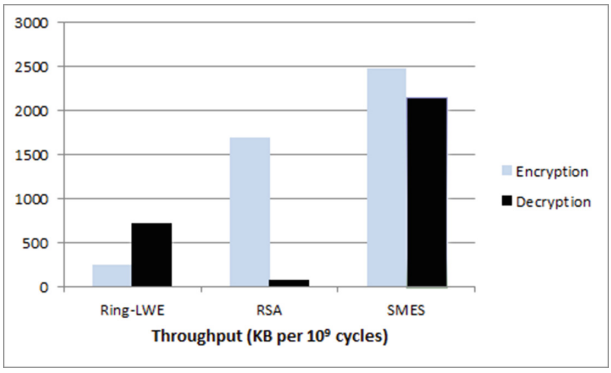

Fig. 3. A comparison of throughput under 80 bits security.

In short, we get enough results to show that SMES is very practical. It is a promising candidate for Post-Quantum Cryptography.

7 Conclusions

In this paper, we exploit the power of modern x64 CPU to give a high performance SMES implementation with low decryption failure probability. The experimental results show that SMES is a promising candidate for Post-Quantum Cryptography.

Here we list some directions for future work.

(1) Use multithreading technique to further improve our implementation.
(2) Improve arithmetic operations in $GF(2^k)$ using SIMD technique.
(3) Extend our techniques to other MPKC schemes.

Acknowledgments. This work was supported by 973 Program (No. 2014CB360501), the National Natural Science Foundation of China (Nos. 61632013, U1135004 and 61170080), Guangdong Provincial Natural Science Foundation (No. 2014A030308006), Guangdong Province Universities and Colleges Pearl River Scholar Funded Scheme (2011), and China Postdoctoral Science Foundation under Grant No. 2015M572318.

References

1. OpenSSL. https://www.openssl.org/
2. Barker, E., Barker, W., Burr, W., Polk, W., Smid, M., Gallagher, P.D., et al.: NIST special publication 800-57 recommendation for key management–part 1: General (2012)
3. Barrett, P.: Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 311–323. Springer, Heidelberg (1987). doi:10.1007/3-540-47721-7_24
4. Bernstein, D.J., Buchmann, J., Dahmen, E.: Post-Quantum Cryptography. Springer Science & Business Media, Heidelberg (2009)
5. Bernstein, D.J., Lange, T., Page, D.: eBATS. ECRYPT benchmarking of asymmetric systems: Performing benchmarks (report) (2008)
6. Bosma, W., Cannon, J., Playoust, C.: The Magma algebra system I: the user language. J. Symb. Comput. **24**(3), 235–265 (1997)
7. Bosselaers, A., Govaerts, R., Vandewalle, J.: Comparison of three modular reduction functions. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 175–186. Springer, Heidelberg (1994). doi:10.1007/3-540-48329-2_16
8. Chen, A.I.-T., Chen, M.-S., Chen, T.-R., Cheng, C.-M., Ding, J., Kuo, E.L.-H., Lee, F.Y.-S., Yang, B.-Y.: SSE implementation of multivariate PKCs on modern x86 CPUs. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 33–48. Springer, Heidelberg (2009). doi:10.1007/978-3-642-04138-9_3
9. Chen, L., Jordan, S., Liu, Y.K., Moody, D., Peralta, R., Perlner, R., Smith-Tone, D.: Report on post-quantum cryptography. National Institute of Standards and Technology Internal Report 8105 (2016)
10. Crandall, R., Pomerance, C.: Prime Numbers: A Computational Perspective, vol. 182. Springer Science & Business Media, Heidelberg (2006)
11. Czypek, P., Heyse, S., Thomae, E.: Efficient implementations of MQPKS on constrained devices. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 374–389. Springer, Heidelberg (2012). doi:10.1007/978-3-642-33027-8_22
12. De Clercq, R., Roy, S.S., Vercauteren, F., Verbauwhede, I.: Efficient software implementation of ring-LWE encryption. In: Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, pp. 339–344. EDA Consortium (2015)
13. Ding, J., Yang, B.-Y., Chen, C.-H.O., Chen, M.-S., Cheng, C.-M.: New differential-algebraic attacks and reparametrization of rainbow. In: Bellovin, S.M., Gennaro, R., Keromytis, A., Yung, M. (eds.) ACNS 2008. LNCS, vol. 5037, pp. 242–257. Springer, Heidelberg (2008). doi:10.1007/978-3-540-68914-0_15
14. Faugere, J.: A new efficient algorithm for computing Gröbner bases (F4). J. Pure Appl. Algebra **139**(1–3), 61–88 (1999)
15. Gligoroski, D., Markovski, S., Knapskog, S.J.: Multivariate quadratic trapdoor functions based on multivariate quadratic quasigroups. In: Proceedings of the American Conference on Applied Mathematics, Stevens Point, Wisconsin, USA, World Scientific and Engineering Academy and Society (WSEAS), pp. 44–49 (2008)
16. Granlund, T., Montgomery, P.L.: Division by invariant integers using multiplication. In: ACM SIGPLAN Notices, vol. 29, pp. 61–72. ACM (1994)
17. Hashimoto, Y.: A note on tensor simple matrix encryption scheme. http://eprint.iacr.org/2016/065.pdf

18. Imai, H., Matsumoto, T.: Algebraic methods for constructing asymmetric cryptosystems. In: Calmet, J. (ed.) AAECC 1985. LNCS, vol. 229, pp. 108–119. Springer, Heidelberg (1986). doi:10.1007/3-540-16776-5_713
19. Patarin, J.: Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): two new families of asymmetric algorithms. In: Maurer, U. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 33–48. Springer, Heidelberg (1996). doi:10.1007/3-540-68339-9_4
20. Petzoldt, A., Chen, M.-S., Yang, B.-Y., Tao, C., Ding, J.: Design principles for HFEv- based multivariate signature schemes. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015. LNCS, vol. 9452, pp. 311–334. Springer, Heidelberg (2015). doi:10.1007/978-3-662-48797-6_14
21. Petzoldt, A., Ding, J., Wang, L.C.: Eliminating decryption failures from the simple matrix encryption scheme. http://eprint.iacr.org/2016/010.pdf
22. Porras, J., Baena, J., Ding, J.: ZHFE, a new multivariate public key encryption scheme. In: Mosca, M. (ed.) PQCrypto 2014. LNCS, vol. 8772, pp. 229–245. Springer, Heidelberg (2014). doi:10.1007/978-3-319-11659-4_14
23. Seo, H., Kim, J., Choi, J., Park, T., Liu, Z., Kim, H.: Small private key MQPKS on an embedded microprocessor. Sensors **14**(3), 5441–5458 (2014)
24. Shor, P.: Algorithms for quantum computation: discrete logarithms and factoring. In: 35th Annual Symposium on Foundations of Computer Science, 1994 Proceedings, pp. 124–134. IEEE (1994)
25. Shor, P.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. **26**, 1484–1509 (1996)
26. Shoup, V.: NTL: A library for doing number theory (2001)
27. Sturmfels, B.: What is a Gröbner basis. Notices Amer. Math. Soc. **52**(10), 1199–1200 (2005)
28. Tao, C., Diene, A., Tang, S., Ding, J.: Simple matrix scheme for encryption. In: Takagi, T. (ed.) PQCrypto 2016. LNCS, vol. 9606, pp. 231–242. Springer, Heidelberg (2013). doi:10.1007/978-3-642-38616-9_16
29. Tao, C., Xiang, H., Petzoldt, A., Ding, J.: Simple matrix-a multivariate public key cryptosystem (MPKC) for encryption. Finite Fields Appl. **35**, 352–368 (2015)
30. Thomae, E.: A generalization of the Rainbow Band Separation attack and its applications to multivariate schemes. IACR Cryptology ePrint Archive 2012, 223 (2012)
31. Thomas, J., Keller, J., et al.: The calcualtion of multiplicative inverses over GF(p) efficiently where p is a Mersenne prime. IEEE Trans. Comput. **100**(5), 478–482 (1986)
32. Yang, B.-Y., Chen, J.-M.: Building secure tame-like multivariate public-key cryptosystems: the new TTS. In: Boyd, C., González Nieto, J.M. (eds.) ACISP 2005. LNCS, vol. 3574, pp. 518–531. Springer, Heidelberg (2005). doi:10.1007/11506157_43