PAPER *Special Section on Security, Privacy and Anonymity in Computation, Communication and Storage Systems*

Practical Hybrid Encryption from Simple Matrix Encryption Scheme

Zhiniang PENG[†], ***Student Member***, **Shaohua TANG***[†a)], **Ju CHEN**[†], **Chen WU**[†], ***and*** **Xinglin ZHANG**[†], ***Nonmembers***

SUMMARY The simple matrix encryption scheme (SMES) is one of the very few existing multivariate public key encryption schemes, which are believed to be resistant against quantum computing attacks. However, there are three problems. The first problem is that its decryption failure probability is too large. There exist some ways to reduce the decryption failure probability, but all of them will result in serious performance degradation. The second problem is that its large key size will cause large latency, as CPU spent more time in reading it from memory rather than doing actual computations. The third problem is that so far none of the published SMES implementation provides indistinguishability under chosen ciphertext attacks. In this paper, we solve the dilemma between performance and decryption failure probability by exploiting the power of modern x64 CPU; We propose several software optimization techniques to optimize its memory access pattern; We also present (1) an efficient implementation of basic SMES and (2) the first implementation of IND-CCA hybrid SMES on a modern x64 CPU. The experimental results show that our basic SMES implementation is three orders of magnitude faster than the existing Rectangular SMES implementation and it's comparable to the fastest Ring-LWE and RSA implementations, and our hybrid SMES implementation enjoy a practical performance.

key words: *AVX2; Simple Matrix Encryption; Post-Quantum Cryptosystem; Hybrid Encryption; MPKC.*

1. Introduction

In [1][2], 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 [3][4], 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 [5] 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 [6], MQQ [7], SMES [8], HFE [9], ZHFE [10]). However, most of them were broken by various attacks, such as MinRank [6], High Rank attack [11][6], Direct attack, Differential attack [11] and Rainbow Band Separation attack [12][11]. SMES and ZHFE are the only existing multivariate public key encryption schemes. None of the existing attacks can cause severe security threats to them.

Recently, there was a manuscript on eprint [13] which claims that SMES can be broken in polynomial time by using linearization equation attack. It is incorrect because the author didn't take into account that the equations obtained by using linearization equation attack from SMES are not linear independent. Most of equations obtained by linearization equation attack in SMES are linear dependent. From another point of view, if this attack can break SMES in polynomial time, SMES will also be broken by computing the Gröbner bases. So the experimental results of algebraic attack in [8] demonstrated that SMES is secure against linearization equation attack.

In ZHFE, we need to invert a univariate polynomial on an extended field of the base field to decrypt a ciphertext. This makes it relatively slow. Compared with ZHFE, SMES enjoys a very high efficiency because all of its computation are linear algebras on the base field. However, SMES is not yet widely used because of the following three problems.

The first problem is that its decryption failure probability is too large. Its decryption failure probability is inversely proportional to the order of its base field. In order to get a reasonable decryption failure probability, we have to choose a very large base 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 [14] to reduce the decryption failure probability. But RSMES also results in serious performance degradation because that it increases the computational complexity of basic SMES. Another variant of SMES called Tensor SMES (TSMES) to eliminate the decryption failure was proposed in [15]. However, the security of TSMES is weaker than that of the basic SMES.

The second problem is that SMES suffers from a significant memory latency in CPU implementation. This is a common problem with all MPKC schemes. Although MPKC schemes are considered theoretically faster than traditional asymmetric cryptosystems 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. None of the previous implementation took memory access patterns into account. Lacking of good memory access patterns in MPKC schemes will result in serious speed penalty [16].

Manuscript received December 15, 2016.
Manuscript revised December 15, 2016.

[†]School of Computer Science & Engineering, South China University of Technology, Guangzhou 510006, China.
a) E-mail: csshtang@scut.edu.cn

DOI: 10.1587/transinf.E0.D.1

The third problem is that none of the previous implementation took into account that the plain MPKC encryption schemes do not provide indistinguishability under chosen plaintext (IND-CPA) or indistinguishability under chosen ciphertext attacks (IND-CCA) [17].

There are some general methods to get an IND-CCA secure encryption scheme from OW-CPA secure encryption scheme in random oracle model [18][19][20]. But the performance of MPKC encryption schemes is very sensitive to memory access patterns. It is not clear how efficient such a system is in practice when applied to MPKC encryption schemes.

Contributions: In this paper, we exploit the power of modern x64 CPU to give a high performance SMES implementation with low decryption failure probability. We also give the first IND-CCA secure hybrid encryption based on the basic SMES. 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 basic 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. Our basic implementation is three orders of magnitude faster than the existing RSMES implementation. It is comparable to RSA implemented in OpenSSL [21] and the fastest Ring-LWE implementation.

3) We propose several software optimization techniques to break through CPU bottleneck of SMES implementation in modern CPU. Our techniques can also be applied to other MPKC schemes.

4) We give the first implementation of IND-CCA secure hybrid encryption in Multivariate Public Key Cryptography. It can achieve a practical performance on modern x64 CPU.

5) Our work 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.

2.1.1 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 $K[x_1, \cdots, x_n]$. $b_1, \cdots, b_n$ 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.

2.1.2 Encryption

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

2.1.3 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 $F(\mathbf{x}) = \mathbf{y}$. 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 called RSMES was proposed in [14]. The decryption failure probability of RSMES is reduced to $\frac{1}{q^{s-r+1}}$. But RSMES has larger parameters than basic SMES. In addition, 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 [15]. 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 [22]. It's security may be threatened by UOV reconciliation attack [11].

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 [23][24][25][26] 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 ($\approx \frac{1}{2^{31}}$). 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. In general, one always uses $a = a - q * \left\lfloor \frac{a}{q} \right\rfloor$ to reduce a positive integer a into $[0, q-1]$. But integer division is slow in a modern CPU. Barrett method [27] and Montgomery method [28] 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 $q = 2^p - 1$, we can do $a = (a\&q) + (a >> p)$ a few times to reduce a positive a into $[0, q-1]$. 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($\vec{a}$,$\vec{b}$) ▷ $\vec{a}$ and $\vec{b}$ are __m256i type
2: $\vec{c}$ = $\_mm256\_mul\_epu32(\vec{a}, \vec{b})$;
3: $\vec{clo}$ = $\_mm256\_and\_si256(\vec{c}, \vec{q})$;
4: $\vec{chi}$ = $\_mm256\_srli\_epi64(\vec{c}, 31)$;
5: $\vec{c}$ = $\_mm256\_add\_epi64(\vec{alo}, \vec{ahi})$;
6: **return** $\_mm\_min\_epu64(\vec{c}, \_mm\_sub\_epi32(\vec{c}, \vec{q}))$;

3.2 Fast inversion

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

In [29], Thomas et al. proposed an efficient algorithm to calculate the multiplicative inverse over $GF(q)$ when q is a Mersenne prime. The key idea of their algorithm is that $\left\lfloor \frac{q}{z} \right\rfloor$ can be easily computed when q is a Mersenne prime. However, this algorithm can't take advantage of modern x64 instructions, because it needs roughly p iterations to compute $\left\lfloor \frac{q}{z} \right\rfloor$. Instead, we find that there exists another interesting inversion algorithm called Relational Reduction Algorithm (RRA) mentioned in [30]. 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. Table 1 and Table 2 display

Algorithm 2 Field inversion algorithm.

```
1: procedure Inv(x)                                  ▷ The inversion of x
2:     (a, b) = (1, 0) and (y, z) = (x, q);
3:     e = TZCNT(y);                                 ▷ x64 instruction
4:     y = y >> e;
5:     a = a << (p − e);
6:     a = (a&q) + (a >> p);
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 line 3;
```

timings for them when using different algorithms. Here we do not consider the time to read and write memory. Timings are the average number of clock cycles in an Intel Core i7-4790 CPU when applying the arithmetic operations in CPU register.

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

The column "Naive" in Table 1 corresponds to the scalar approach using the C++ operator % to compute remainder. In fact, the compiler will use the method mentioned in [31] to optimize modular reduction.

From the timing results, we can see that our base 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. Improving Basic SMES

In this section, we propose several optimization techniques to improve the performance of basic SMES on modern x64 CPU.

4.1 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 fact, nearly all the computations in SMES encryption and decryption can be sped up by SIMD computing.

For encryption, we can use the Macaulay matrix time variable vector to compute the ciphertext $\mathbf{y} = P(\mathbf{x})$ as in [32]. For decryption, affine transformation, polynomial matrix multiplication and Gauss Elimination are needed. Affine transformation and Gauss Elimination are suitable for SIMD computing [33]. The complex polynomial matrix multiplication can be replaced by linear algebra over the base field.

4.2 Software Optimization

Although MPKC schemes are considered theoretically faster than traditional asymmetric cryptosystems 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 CPU. We focus on optimizing Gauss Elimination (GE) and matrix vector multiplication (MVM), which are the most expensive parts of SMES, for SMES in our base field. All our experiments are carried on an Intel Core i7-4790 @3.60Ghz CPU with TurboBoost disabled.

Hybrid representation: In Section 3.1, we use 64-bit integer to represent a field element. This is because 64-bit representation can prevent integer overflow and allows us to use lazy modular reduction technique. But it will cost redundant storage. 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.

Loop unrolling: Matrix operations are always handled by loop statement in C++. 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. We use loop unrolling technique to optimize program running speed at the expense of its binary size. This is known as space-time trade-off. Our experiments show that loop with stripe equal to 4 can get the best space-time tradeoff.

Lazy modular reduction: 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.

Pipeline optimization: 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.

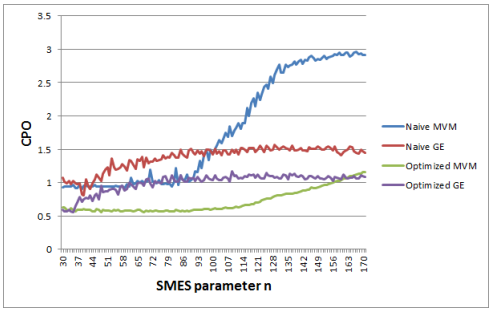


Fig. 1: CPO of our MVM and GE.

This will help us to reduce unnecessary CPU stalls.

Parameters-aware programing: As our codes are very sensitive to the working set size, we have to use different codes to deal with different parameters. We design a number of different codes to deal with the different border situations in each inner iterations of GE and MVM. We test the performance of different codes for different parameters and record the best one. By using this technique, the performance can be improved by 2% for MVM and 5% for GE.

Cache-aware algorithm: When the size of the Macaulay matrix is larger than L1 cache, we use a divide and conquer technique to split the matrix and vector into small ones. This is a cache-aware algorithm [34]. In our experiments, this technique can reduce L1 cache miss rate when $n > 89$.

Results: After applying all the optimizations we mentioned, we measure the CPO (cycles per mul-and-add operation) behaviour of MVM and GE (without counting the cycles of inversion operations). The results are shown in Figure 1.

As MVM is more suitable for SIMD computing, it has better CPO than GE when n is small. But when n becomes 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.

5. Hybrid Encryption from SMES

None of the previous implementation took into account that the plain MPKC encryption schemes do not provide indistinguishability under chosen IND-CPA or IND-CCA. Just like plain RSA encryption scheme, the plain MPKC encryption schemes can be viewed as an one-way trapdoor function. They only provide one-wayness under chosen ciphertext attacks (OW-CPA).

In reality, ciphertext indistinguishability is what really needed. Intuitively, if an encryption scheme can provide indistinguishability, then an adversary will not able to distinguish pairs of ciphertext based on the message they encrypt. IND-CPA or IND-CCA is considered a basic requirement for public key encryption schemes.

In this section, we construct a IND-CCA secure hybrid SMES based on basic SMES.

5.1 Hybrid encryption

A hybrid encryption scheme can be divided into two independent components [35]: (1) a key encapsulation mechanism (KEM), which is a public key cryptosystem that encrypts a randomly generated symmetric session key, and (2) a data encapsulation mechanism (DEM), which is a symmetric key cryptosystem that uses the symmetric session key to encrypt the plaintext.

Symmetric key encryption is widely used in practice, it enjoys high security and efficiency. But the biggest problem with symmetric key encryption is that the sender has to share a secret key with the receiver. This is inconvenient for real world applications. In a public key encryption scheme, anyone can encrypt a message using the public key of the receiver without having a shared secret key with the receiver. But it is not suitable for large plaintext because of its lack of efficiency.

Hybrid encryption scheme combines the convenience of a public key encryption scheme with the efficiency of a symmetric key encryption scheme. It's widely used in many scenarios.

5.2 The KEM-DEM framework

Key Encapsulation Mechanism: A key encapsulation mechanism is essentially a key distribution mechanism aiming to allow both sender and receiver to share a random session key. A key encapsulation mechanism consists of three algorithms (KEM.Kg,KEM,Enc,KEM.Dec) described in Table 3.

Table 3: Key Encapsulation Mechanism.

KEM.Kg	The key generation algorithm takes security parameter as input and outputs the public/private key pair (pk,sk).
KEM.Enc	The encryption algorithm takes the public key pk as input and outputs a key/ciphertext pair (K,C_0).
KEM.Dec	The decryption algorithm takes the private key sk and the ciphertext C_0 as inputs and outputs the session key K or $\perp$.

.

Correctness: A KEM is required to be correct for all but a negligible probability. Which means if (pk,sk)$\leftarrow$ KEM.Kg(), the following probability should be close to 1: Pr[KEM.dec(pk,C_0)=K|(C_0,K)$\leftarrow$ KEM.Enc(pk)].

Data Encapsulation Mechanism: A data encapsulation mechanism is a symmetric encryption scheme which

encrypts the plaintext using the session key outputted by the KEM. A data encapsulation mechanism consists of two algorithms (DEM.Enc,DEM.Dec) described in Table 4.

Table 4: Data Encapsulation Mechanism.

DEM.Enc	The encryption algorithm takes the secret key K and plaintext m as inputs and it outputs the a ciphertext C_1.
DEM.Dec	The decryption algorithm takes the secret key K and the ciphertext C_1 as inputs and it outputs the plaintext m or $\perp$.

A hybrid cryptosystem can be constructed using a KEM and a DEM. It is a public key cryptosystem, whose public and private key spaces are the same as key spaces of KEM. Table 5 shows the constructing of hybrid encryption scheme from KEM and DEM.

Table 5: Hybrid encryption from KEM and DEM.

H.Kg	The key generation algorithm takes security parameter as input and outputs the public/private key pair (pk,sk) using KEM.Kg().
H.Enc	The encryption algorithm takes public key pk and plaintext m as inputs and uses KEM.Enc(pk) to generate a key/ciphertext pair (K,C_0). Then it runs DEM.Enc(K,m) to get ciphertext C_1. In the end, it outputs the ciphertext C=(C_0,C_1).
H.Dec	The decryption algorithm takes the private key sk and the ciphertext C=(C_0,C_1) as inputs and computes the session key K by using DEM.Dec(C_0). Then it runs DEM.Dec(K,C_1) to get the plaintext m. It outputs $\perp$ if any of the above algorithm output $\perp$, otherwise it outputs the plaintext m.

If KEM and DEM are all IND-CCA secure, then the corresponding hybrid encryption scheme is also IND-CCA secure [35].

5.3 KEM from SMES

Before proposing our construction, we first introduce key derivation function (KDF) which is needed for our construction.

Key Derivation Function: A key derivation function is a deterministic function that takes a random string **x** as input and outputs a string **K** with fix length.

In cryptography, a KDF is often modelled as a random oracle and used to generate secret key from a shared secret. In practice, we always instantiate KDF with a cryptographic hash function.

We present our KEM from SMES in Table 6. Our work follows up the suggestion from Alexander [36].

Here we prove that our KEM is IND-CCA secure in random oracle model in the following theorem.

Theorem 1: The advantage of any adversary A in IND-CCA game against our KEM is less than $\epsilon+N/q^n$, where ϵ is the advantage of some efficient algorithm against the one-wayness of SMES and N is the total number of decryption queries A can ask.

Table 6: KEM from SMES.

KeyGen	The key generation algorithm chooses the appropriate set of parameters (K,s,n,m) according to the required security level. It generates the SMES secret key (B,C,S,T) and publishes the public key P.
Enc	The encryption algorithm takes the public key P as input and chooses a random vector x, then it computes the K=KDF(x), t=Hash(x) and c=P(x). Let C=(c,t). Finally, it outputs a key/ciphertext pair (K,C).
Dec	The decryption algorithm takes the private key (B,C,S,T) and ciphertext C=(c,t) as inputs. It runs the decryption algorithm SMES.Dec(c) to get x. If succeeds and hash(x) equals to t, it outputs K=KDF(x). Otherwise, it outputs $\perp$.

Proof. We show how to construct an adversary B using A to break the one-wayness of SMES. The game between the challenger and the adversary B starts with the challenger which first randomly chooses a SMES key pair and randomly chooses a vector $x*$. It computes $y* = P(x*)$ and gives B P and $y*$. Then algorithm B is supposed to output the preimage $x*$.

In our proof, we replace KDF with a random oracle H_1 and hash function with random oracle H_2. B works by interacting with A in an IND-CCA game as follows (B simulates the challenger for A):

Key generation: On input the instance (P,$y*$) of SMES, B chooses P to be the public key of KEM and gives it to A. Then B creates two empty lists L_1 and L_2.

Decryption queries: When A asks decryption oracle for a ciphertext C= (c,t). If c=$y*$, B returns $\perp$. If there exists (x,t) in L_2 such that c=$P(x)$, then B computes K=KDF(x) by asking the KDF oracle H_1, and returns K. Otherwise, B returns $\perp$.

Challenge ciphertext : B randomly chooses K_0 and K_1, and adds ($y*$,K_0) to list L_1. Then B selects a bit σ uniformly from {0,1} and returns (K_σ,$y*$) to A.

Hash queries: When A asks the hash oracle H_2 for some x, B first checks the list L_2. If (x,t) is in L_2, B returns t. Otherwise, B returns a random t and adds (x,t) to L_2.

KDF queries: When A asks the KDF oracle H_1 for some x. B first checks the list L_1. If x is in L_1 for some K, B returns K. Otherwise B returns a random K and adds (x,K) to L_1.

Challenge: After A outputs its answer σ', B check lists to see if there exists some (x,K) in L_1 or some (x,y) in L_2 such that $P(x) = y*$. If so, B outputs the preimage x.

Claim: During the simulation algorithm A's view is identical to that in the real attack unless the challenge ciphertext $y*$ had been submitted to the decryption oracle. This will happen with probability less than N/q^n,

where q^n is the number of valid ciphertexts and N is the total number of decryption queries A can ask. As we are in random oracle model, A has no advantage of guessing the output of random oracle unless A submits $x*$ to random oracle. As long as A submits $x*$, B can break the one-wayness of SMES. So we can conclude that the the advantage of any adversary A in IND-CCA game against our KEM is less than $\epsilon + N/q^n$, where ϵ is the advantage of some efficient algorithm against the one-wayness of SMES and N is the total number of decryption queries A can ask.

5.4 AES-GCM as DEM

In general, we need an IND-CPA encryption scheme combined with a secure MAC to get an IND-CCA secure DEM. For example, AES-CBC-HMAC-SHA1 is an authenticated encryption scheme combined AES-CBC with HMAC. However, this general composition is generally inefficient because there are very few MAC schemes can keep up with cipher in speed. In this paper, we choose AES-GCM as DEM for better performance. AES-GCM is an authenticated encryption aiming to provide high speed and constant-size tag. In AES-GCM, Galois field multiplication which is used for message authentication can be computed in parallel with the block cipher. By using the AES-NI instruction set, AES-GCM outperforms AES-CBC-HMAC-SHA1 by a factor of 5 in throughout.

5.5 Hybrid SMES

Our hybrid SMES is constructed using the KEM-DEM framework described in Table 5. The KEM is instantiated with KEM constructed in Section 5.3 and the DEM is instantiated with AES-GCM.

As both our KEM and DEM are IND-CCA secure, the corresponding hybrid SMES is also IND-CCA secure.

6. Results

In this section, we choose security parameters for basic SMES and hybrid SMES and compare our implementations with other encryption schemes. All our experiments are carried on an Intel Core i7-4790 @3.60Ghz CPU with TurboBoost disabled.

6.1 Choosing parameters for basic SMES

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 $O(q^{\lceil \frac{m}{n} \rceil 2s} m^3)$. We can choose our parameters by these complexity formulas.

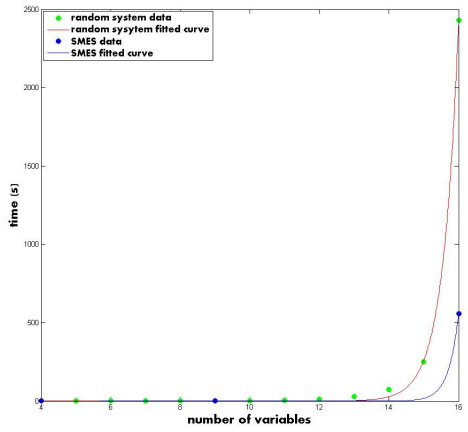


Fig. 2: Fitted curves for computational complexity of F4 algorithm against random system and SMES.

However, there is no known 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 [37], which contains an efficient implementation of F4 algorithm [38] for computing Gröbner bases [39]. 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 Figure 2.

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 7.

Table 7: Parameters of SMES for different levels of security over $GF(2^{31}-1)$.

Security (bit)	80	112	128
Parameters (s,n,m)	(7,49,28)	(8,64,128)	(9,81,162)
Plaintext size (bit)	1,519	1,984	2,511
Ciphertext size (bit)	3,038	3,968	5,022
Public key size (kB)	472.8	1,039.0	2,086.2
Private key size (kB)	64.2	109.2	174.7
Decryption failure rate	$\approx \frac{1}{2^{31}}$	$\approx \frac{1}{2^{31}}$	$\approx \frac{1}{2^{31}}$

6.2 Comparing with other encryption schemes

We use the techniques according to Section 4 to implement basic SMES and compare our result with the fastest RSA, Ring-LWE and RSMES implementations.

For RSMES, we choose the implementation results from [14]. In [14], authors present various RSMES parameters on different fields. But they only implement it in $GF(8)$ for performance reason. In $GF(8)$, the decryption failure probability of RSMES is 2^{-32}. It has a similar decryption failure probability with ours ($\approx \frac{1}{2^{31}}$).

For RSA, we choose the RSA parameters according to the latest NIST key management recommendation [40]. To get a fair comparison, we get the benchmark results of RSA implemented in OpenSSL from eBATS (ECRYPT Benchmarking of Asymmetric Systems) [41]. For Ring-LWE encryption, we choose the fastest known result in [42]. Table 8 shows an overall comparison in encryption and decryption speed.

Table 8: Comparing with RSA and Ring-LWE.

Parameters	Security (bit)	Encryption cycles(10^3)	Decryption cycles(10^3)	Ciphertext expansion
Ring-LWE256	80	121.2	43.3	26
Ring-LWE512	112	261.9	96.5	28
RSA1024	80	73.4	1,434.7	1
RSA2048	112	140.1	5,588.7	1
RSA3072	128	209.5	14,958.3	1
RSMES80	80	48,000	60,000	2
RSMES100	100	134,000	149,000	2
SMES80	80	71.8	83.2	2
SMES112	112	160.9	137.9	2
SMES128	128	313.2	232.4	2

From Table 8, we can see that our implementation is almost three orders of magnitude faster than the existing RSMES implementation. This is because the base field $GF(2^{31}-1)$ provides stronger security. It enables us to choose smaller n and m. Smaller n make our implementation much faster. In addition, we exploit the power of modern CPU to get faster $GF(2^{31}-1)$ arithmetic operations. The choice of $GF(2^{31}-1)$ actually gives us win-win situation.

Compared with Ring-LWE, our basic 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 has larger key size than RSA and Ring-LWE. As storage is not a bottleneck for modern computer now, the only problem is communication overhead for public key. Large key size will make SMES unsuitable for some scenarios in which public key is frequently required to be transferred. Nonetheless, considering its speed and quantum attack-resistance, SMES is useful in many scenarios.

6.3 Hybrid SMES results

Our implementation of hybrid SMES is based on the basic SMES implementation mentioned in Section 5. We instantiate DEM with AES-GCM and hash with SHA-1. As for KDF, we instantiate it using SHA-1 with padding. We choose SMES80-AES-GCM128-SHA1 for 80-bit security and SMES128-AES-GCM128-SHA1 for 128-bit security. AES-NI is used to improve our implementation.

AES-NI is an extension to the x64 instruction set architecture for modern CPU proposed by Intel. It can greatly improve the speed of AES-GCM encryption. We implement AES-GCM according to the method in [43]. We use Karatsuba algorithm to compute GHASH in AES-GCM. The PCMULQDD instruction is used to increase the speed of polynomial multiplication. We generate 1000 random instances of hybrid SMES to encrypt messages with different length and calculate their average performance. The performance in cycles/byte of hybrid SMES is recorded in Table 9.

Table 9: Performance in cycles/byte of hybrid SMES.

Scheme	Security (bit)	Message length (byte)	Cycles/byte for encrypt	Cycles/byte for decrypt
Hybrid SMES80	80	64	1,665.4	1,860.8
		1536	72.2	80.5
		36,864	5.6	6.1
		1,048,576	2.9	3.0
Hybrid SMES128	128	64	6,601.5	5,171.9
		1536	277.7	218.4
		36,864	14.3	11.9
		1,048,576	3.2	3.2

Table 10: Results of hybrid SMES.

Scheme	Hybird SMES80	Hybird SMES128
Encapsulation time(10^3 cycles)	105.2	420.5
Decapsulation time(10^3 cycles)	117.6	329.5
C_0 overhead(bit)	3,038	5,022
Public key size(kB)	472.8	2,086.2
Private key size(kB)	64.2	174.7
Decryption failure rate	$\approx \frac{1}{2^{31}}$	$\approx \frac{1}{2^{31}}$

As is shown in Table 10 and Table 9, the execution time of KEM is independent of message length. When the message length is short, the cycles/bytes of the hybrid SMES is large and the execution time of hybrid SMES mainly depends on the execution time of KEM. As the length of the message becomes longer, cycles/bytes of hybrid SMES converges to a constant number.

The computation time of the hybrid SMES isn't a simple summation of the computation time of basic SMES, KDF, hash function and DEM. Because the combination of KEM and DEM will increase working set size of hybrid SMES, it will introduce some memory latency into our program. The experimental results show that the execution time of

the KEM is only about 1.3 to 1.5 times of that in the basic SMES. This is very fast. The only drawback of hybrid SMES is the key size. Nonetheless, our results are positive. The KEM-DEM framework works well for SMES and our hybrid SMES is practical for applications where key is not frequently required to be transferred.

In short, the results show that SMES can actually be very efficient. We can construct efficient and IND-CCA secure hybrid encryption scheme on it. It is a promising candidate for Post-Quantum Cryptography.

7. Conclusion

In this paper, we exploit the power of modern x64 CPU to give a high performance SMES implementation with low decryption failure probability and propose several software optimization techniques to optimize its memory access patterns. We also construct an efficient IND-CCA secure hybrid SMES based on the basic SMES. The experimental results show that both our basic SMES and hybrid SMES are promising candidates for Post-Quantum Cryptography.

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

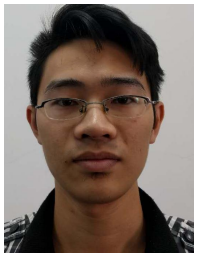
[1] P. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on, pp.124–134, IEEE, 1994.
[2] P. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," SIAM Journal on Computing, vol.26, pp.1484–1509, 1996.
[3] D.J. Bernstein, J. Buchmann, and E. Dahmen, Post-quantum cryptography, Springer Science & Business Media, 2009.
[4] L. Chen, S. Jordan, Y.K. Liu, D. Moody, R. Peralta, R. Perlner, and D. Smith-Tone, "Report on Post-quantum cryptography," National Institute of Standards and Technology Internal Report, vol.8105, 2016.
[5] H. Imai and T. Matsumoto, "Algebraic methods for constructing asymmetric cryptosystems," Algebraic Algorithms and Error-Correcting Codes, pp.108–119, 1986.
[6] B. Yang and J. Chen, "Building secure tame-like multivariate public-key cryptosystems: The new TTS," Information Security and Privacy, pp.518–531, Springer, 2005.
[7] D. Gligoroski, S. Markovski, and S.J. Knapskog, "Multivariate quadratic trapdoor functions based on multivariate quadratic quasigroups," Proceedings of the American Conference on Applied Mathematics, Stevens Point, Wisconsin, USA, World Scientific and Engineering Academy and Society (WSEAS), pp.44–49, 2008.
[8] C. Tao, A. Diene, S. Tang, and J. Ding, "Simple matrix scheme for encryption.," PQCrypto, vol.13, pp.231–242, 2013.
[9] J. Patarin, "Hidden fields equations (HFE) and isomorphisms of polynomials (IP): Two new families of asymmetric algorithms," Advances in Cryptology-EUROCRYPT' 96, pp.33–48, Springer, 1996.
[10] J. Porras, J. Baena, and J. Ding, "ZHFE, a new multivariate public key encryption scheme," in Post-Quantum Cryptography, pp.229–245, Springer, 2014.
[11] J. Ding, B. Yang, C. Chen, M. Chen, and C. Cheng, "New differential-algebraic attacks and reparametrization of Rainbow," Proceedings of the 6th international conference on Applied cryptography and network security, pp.242–257, Springer-Verlag, 2008.
[12] E. Thomae, "A generalization of the Rainbow Band Separation attack and its applications to multivariate schemes.," IACR Cryptology ePrint Archive, vol.2012, p.223, 2012.
[13] C. Gu, "Cryptanalysis of simple matrix scheme for encryption." `http://eprint.iacr.org/2016/1075.pdf`.
[14] C. Tao, H. Xiang, A. Petzoldt, and J. Ding, "Simple matrix–a multivariate public key cryptosystem (MPKC) for encryption," Finite Fields and Their Applications, vol.35, pp.352–368, 2015.
[15] A. Petzoldt, J. Ding, and L.C. Wang, "Eliminating decryption failures from the simple matrix encryption scheme,"
[16] U. Drepper, "What every programmer should know about memory," Red Hat, Inc, vol.11, p.2007, 2007.
[17] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway, "Relations among notions of security for public-key encryption schemes," Annual International Cryptology Conference, pp.26–45, Springer, 1998.
[18] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern, "RSA-OAEP is secure under the RSA assumption," Annual International Cryptology Conference, pp.260–274, Springer, 2001.
[19] T. Okamoto and D. Pointcheval, "RSA-REACT: An alternative to RSA-OAEP," Proc. second open NESSIE workshop, Egham, 2001.
[20] E. Kiltz and J. Malone-Lee, "A general construction of IND-CCA2 secure public key encryption," IMA International Conference on Cryptography and Coding, pp.152–166, Springer, 2003.
[21] "OpenSSL." `https://www.openssl.org/`.
[22] Y. Hashimoto, "A note on tensor simple matrix encryption scheme,"
[23] P. Czypek, S. Heyse, and E. Thomae, "Efficient implementations of MQPKS on constrained devices," in Cryptographic Hardware and Embedded Systems–CHES 2012, pp.374–389, Springer, 2012.
[24] A.I.T. Chen, M.S. Chen, T.R. Chen, C.M. Cheng, J. Ding, E.L.H. Kuo, F.Y.S. Lee, and B.Y. Yang, "SSE implementation of multivariate PKCs on modern x86 CPUs," in Cryptographic Hardware and Embedded Systems-CHES 2009, pp.33–48, Springer, 2009.
[25] H. Seo, J. Kim, J. Choi, T. Park, Z. Liu, and H. Kim, "Small private key MQPKS on an embedded microprocessor," Sensors, vol.14, no.3, pp.5441–5458, 2014.
[26] A. Petzoldt, M.S. Chen, B.Y. Yang, C. Tao, and J. Ding, "Design principles for HFEv- based multivariate signature schemes," in Advances in Cryptology–ASIACRYPT 2015, pp.311–334, Springer, 2015.
[27] P. Barrett, "Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor.," Crypto, pp.311–323, Springer, 1986.
[28] A. Bosselaers, R. Govaerts, and J. Vandewalle, "Comparison of three modular reduction functions," Annual International Cryptology Conference, pp.175–186, Springer, 1993.
[29] J. Thomas, J. Keller, *et al.*, "The calcualtion of multiplicative inverses over GF(p) efficiently where p is a Mersenne prime," Computers, IEEE Transactions on, vol.100, no.5, pp.478–482, 1986.
[30] R. Crandall and C. Pomerance, Prime numbers: a computational perspective, Springer Science & Business Media, 2006.
[31] T. Granlund and P.L. Montgomery, "Division by invariant integers using multiplication," ACM SIGPLAN Notices, pp.61–72, ACM, 1994.
[32] S. Tanaka, T. Nishide, and K. Sakurai, "Efficient implementation of evaluating multivariate quadratic system with GPUs," Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on, pp.660–664, IEEE, 2012.
[33] S. McGinn and R.E. Shaw, "Parallel Gaussian Elimination using OpenMP and MPI.," HPCS, pp.169–176, 2002.

[34] M. Kowarschik and C. Weiß, "An overview of cache optimization techniques and cache-aware numerical algorithms," in Algorithms for Memory Hierarchies, pp.213–232, Springer, 2003.
[35] R. Cramer and V. Shoup, "Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack," SIAM Journal on Computing, vol.33, no.1, pp.167–226, 2003.
[36] A.W. Dent, "A designerâĂŹs guide to KEMs," IMA International Conference on Cryptography and Coding, pp.133–151, Springer, 2003.
[37] W. Bosma, J. Cannon, and C. Playoust, "The Magma algebra system I: The user language," Journal of Symbolic Computation, vol.24, no.3, pp.235–265, 1997.
[38] J. Faugere, "A new efficient algorithm for computing Gröbner bases (F4)," Journal of pure and applied algebra, vol.139, no.1-3, pp.61–88, 1999.
[39] B. Sturmfels, "What is a Gröbner basis," Notices Amer. Math. Soc, vol.52, no.10, pp.1199–1200, 2005.
[40] E. Barker, W. Barker, W. Burr, W. Polk, M. Smid, P.D. Gallagher, *et al.*, "NIST special publication 800-57 recommendation for key management–part 1: General," 2012.
[41] D.J. Bernstein, T. Lange, and D. Page, "eBATS. ECRYPT benchmarking of asymmetric systems: Performing benchmarks (report). 2008."
[42] R. De Clercq, S.S. Roy, F. Vercauteren, and I. Verbauwhede, "Efficient software implementation of ring-LWE encryption," Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, pp.339–344, EDA Consortium, 2015.
[43] V. Gopal, E. Ozturk, W. Feghali, J. Guilford, G. Wolrich, and M. Dixon, "Optimized Galois-Counter-Mode Implementation on Intel® Architecture Processors," Intel White Paper, August, 2010.

Zhiniang Peng received the BE Degree from South China University of Technology in 2013. Now he is a doctoral candidate at the School of Computer Science and Engineering, South China University of Technology. His current research interests include lattice-based cryptography, crypto chip design, and multivariate public key cryptography.

Shaohua Tang received the B.Sc. and M.Sc. Degrees in applied mathematics from South China University of Technology, China, in 1991 and 1994, respectively, and the Ph.D. Degree in communication and information system from South China University of Technology, in 1998. He was a visiting scholar with North Carolina State University, USA, and a visiting professor with University of Cincinnati, USA. He has been a full professor with the School of Computer Science and Engineering, South China University of Technology since 2004. His current research interests include multivariate public key cryptography, crypto chip design, and distributed system security. He has authored or co-authored over 100 technical papers in journals and conference proceedings.

Ju Chen received the BE Degree from Guangdong University of Technology in 2014. Now he is a postgraduate student at the School of Computer Science and Engineering, South China University of Technology. His current research interests include multivariate public key cryptography and crypto chip design.



Chen Wu received the BE Degree from South China University of Technology in 2016. Now he is a postgraduate student at the School of Computer Science and Engineering, South China University of Technology. His research interests include public key cryptography, system security, etc.

Xinglin Zhang received the BE degree in software engineering from Sun Yat-sen University in 2010, and the PhD degree in computer science from Hong Kong University of Science and Technology in 2014. He is now with the South China University of Technology. His main research interests include crowdsensing, wireless ad-hoc/sensor networks, and mobile computing.