Future Generation Computer Systems 52 (2015) 116–124

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Efficient hardware implementation of PMI+ for low-resource devices in mobile cloud computing☆

Shaohua Tang [a,*], Bo Lv [a], Guomin Chen [a], Zhiniang Peng [a], Adama Diene [b], Xiaofeng Chen [c]

[a] School of Computer Science and Engineering, South China University of Technology, China
[b] Department of Mathematical Sciences, United Arab Emirates University, United Arab Emirates
[c] School of Telecommunications Engineering, Xidian University, China

HIGHLIGHTS

- We design a hardware to implement MQ asymmetric cipher PMI+ for low-resource devices.
- Basic arithmetic units are implemented in optimized and full parallel.
- Our design can complete a large power operation in 16 clock cycles.
- Our hardware can complete an encryption operation of PMI+ within 497 clock cycles, and a decryption operation within 438 clock cycles.
- Our design has a good performance in cycle-area products.

ARTICLE INFO

Article history:
Received 30 June 2014
Received in revised form
12 October 2014
Accepted 14 November 2014
Available online 24 November 2014

Keywords:
Multivariate Quadratic (MQ) public key algorithm
PMI+ encryption and decryption
Hardware implementation
Mobile cloud computing
Low-resource devices
Optimized large power operation

ABSTRACT

With rapid development of cloud computing, security issues have gained more and more attention, especially in mobile cloud computing environment. Smart phones and other mobile devices provide a lot of convenience to us, but due to its intrinsic low-resource limitation, it also causes many security problems. In this paper, we design a hardware that can efficiently implement PMI+, which is a Multivariate Quadratic (MQ) asymmetric cipher, for low-resource devices in mobile cloud computing. Our main contributions are that, firstly, hardware architectures of encryption and decryption of PMI+ are developed, and descriptions of corresponding hardware algorithm are proposed; secondly, basic arithmetic units are implemented with higher efficiency that multiplication, squaring, vector dot product and power operation are implemented in full parallel; and thirdly, optimized implementations for core modules, including optimized large power operation, are achieved. The encryption and decryption hardware of PMI+ is efficiently realized on FPGA by the above optimization and improvement. It is verified by experiments that the designed hardware can complete an encryption operation within 497 clock cycles, and the clock frequency can be up to 145.60 MHz, and the designed hardware can complete a decryption operation within 438 clock cycles wherein the clock frequency can be up to 132.21 MHz. Our experiment results also confirm that our design can be deployed in low-resource devices as thin client of mobile cloud computing.

1. Introduction

Security is one of the most significant barriers to faster and more widespread adoption of cloud computing, which has become the focus that many researchers care [1–6]. Thin client technology has been widely used in cloud computing. However, due to the limitation of resources, computation and data processing are usually performed in the clouds. It is not hard to predict that connecting mobile devices to the cloud will become mainstream in the coming days. When users use cloud applications or cloud services, data transmission is engaged. In order to preserve the privacy of sensitive data, mobile device itself requires not only basic security handling capacity and defense capabilities, but also public key cryptographic processing capabilities. Public key cryptography has played an important role in modern communication and computer networks. The public key cryptography, which is used widely, mainly includes RSA that is based on integer factorization problem, ElGamal that is based on discrete logarithm problem, and elliptic curve cryptography, etc. In order to adapt various

☆ The material in this paper was presented in part at the 10th International Conference on Information Security Practice and Experience (ISPEC'2014) (Tang et al., (2014) [38]), May 2014.

* Corresponding author.
E-mail addresses: csshtang@scut.edu.cn, shtang@IEEE.org (S. Tang).

http://dx.doi.org/10.1016/j.future.2014.11.014
0167-739X/

occasions, many efficient hardware implementations are proposed by researchers [7–17]. However, traditional public key cryptography, such as RSA, is not suitable for low-resource mobile devices, because of its large amount of calculation and high power requirements.

The quantum algorithm of P. Shor is able to solve the integer factorization and discrete logarithm problem in polynomial time, including a computation problem in elliptic curve field. As a result, it directly threatens classical cryptosystems based on hard problems of number theory, and which helps to drive the development of post quantum cryptography. The post quantum cryptography can be divided into four categories: signature schemes based on hash function [18], lattice-based public key cryptosystem [19], public key cryptosystem based on error correcting code [20] and multivariate public key cryptosystem [21]. The research for post quantum cryptography is growing rapidly and many hardware and embedded system implementations of the post quantum cryptography appear in order to adapt various occasions [22–29].

MPKCs have advantages in low-resource devices with its low-power and high speed computation. PMI+ [30] is one kind of multivariate public key cryptosystem, and is a variant of MI [31]. Ding enhanced the security of MI by adding an internal perturbation to the central map of MI in 2004, to produce a new variant of the MI cryptosystem which is called PMI cryptosystem [32]. However, the PMI cryptosystem has been broken by differential cryptanalysis by Fouque et al. [33] in 2005. Ding introduced a new external perturbation to the central mapping of MI [30] in 2006, to produce PMI+ cryptosystem whose security has been greatly improved. In this paper, we design an efficient PMI+ hardware implementation which works fine for low-resource devices under mobile cloud computing environment.

The material in this paper was presented in part at the 10th International Conference on Information Security Practice and Experience (ISPEC'2014) (Tang et al., (2014) [38]), May 2014.

1.1. Our contribution

In this paper, we design a hardware that can efficiently implement PMI+ for low-resource devices in mobile cloud computing.

Firstly, hardware architectures of encryption and decryption of PMI+ are developed, and descriptions of corresponding hardware algorithm are proposed.

Secondly, basic arithmetic units are implemented with higher efficiency that multiplication, squaring, vector dot product and power operation are implemented in full parallel, wherein compared with a full parallel multiplier, a full parallel squarer takes up about one-twentieth of the logical units and has shorter latency.

Thirdly, we implement an optimized large power operation which, compared with the general power operation, has the ability to reduce 4288 cycles at most in one process of decryption, with an obvious optimization. The encryption and decryption hardware of PMI+ is efficiently realized on FPGA by the above optimization and improvement.

Our experiments verify that if parameters are selected as $(n, q, \theta, r, a) = (84, 2, 4, 6, 14)$, the length of a plaintext block is 84 bits and the length of a ciphertext block is 98 bits. Our designed hardware can complete an encryption operation within 497 clock cycles or 3.42 μs, wherein the clock frequency can be up to 145.60 MHz, and our designed hardware can complete a decryption operation within 438 clock cycles or 3.31 μs, wherein the clock frequency can be up to 132.21 MHz.

1.2. Organization

The rest of this paper is organized as follows. The theory of PMI+ scheme, including principles of encryption and decryption algorithms, and the choice of parameters, is briefly introduced in Section 2. Section 3 primarily focuses on hardware design and implementation of PMI+, including hardware structure design, algorithm description, and implementation of basic arithmetic units and hardware core modules. Our experimental results, performance, and comparisons with other public key crypto systems are given in detail in Section 4. Section 5 is the conclusion of this paper, which summarizes the main contributions of this paper and proposes further research directions.

2. Preliminaries

We describe the basic theory of the encryption and decryption of PMI+ [30] in this section. The basic idea of PMI+ is adding an internal perturbation and an external perturbation to the central map of MI scheme to resist linearization equation attack and differential attack. The ciphertext encrypted by PMI+ encryption scheme is unique, and it can be used to construct a multivariate public key signature scheme.

2.1. Notations for PMI+

Let k be a finite field of characteristic two and cardinality q, K be an extension of degree n over k. Let $\varphi : K \to k^n$ defined by $\varphi(a_0 + a_1x + \cdots + a_{n-1}x^{n-1}) = (a_0, a_1, \ldots, a_{n-1})$.

Fix θ so that $\gcd(q^\theta + 1, q^n - 1) = 1$ and define $\tilde{F} : K \to K$ by $\tilde{F}(X) = X^{1+q^\theta}$. Then F is invertible and $\tilde{F}^{-1}(X) = X^t$, where $t(1 + q^\theta) \equiv 1 \bmod (q^n - 1)$.

Define the map $F' : k^n \to k^n$ by $F'(x_1, \ldots, x_n) = \varphi \circ \tilde{F} \circ \varphi^{-1}(x_1, \ldots, x_n)$.

Fix a small integer r and randomly choose r invertible affine linear functions $z_1, \ldots, z_r$, written as $z_j(x_1, \ldots, x_n) = \sum_{i=1}^{n} \alpha_{ij}x_i + \beta_j$, for $j = 1, \ldots, r$. This defines a map $Z : k^n \to k^r$ by $Z(x_1, \ldots, x_n) = (z_1, \ldots, z_r)$. The map Z is the source of the internal perturbation.

Randomly choose n quadratic polynomials $\hat{f}_1, \ldots, \hat{f}_n \in k[z_1, \ldots, z_r]$. The $\hat{f}_i$ define a map $\hat{F} : k^r \to k^n$ by $\hat{F}(z_1, \ldots, z_r) = (\hat{f}_1, \ldots, \hat{f}_n)$. Let P be the set consisting of the pairs (λ, μ), where λ is a point that belongs to the image of $\hat{F}$ and μ is the set of pre-images of λ under $\hat{F}$.

Define an internal perturbation map by $F^*(x_1, \ldots, x_n) = \hat{F} \circ Z(x_1, \ldots, x_n) = (f_1^*, \ldots, f_n^*)$. Define a map by $F(x_1, \ldots, x_n) = (F' + F^*)(x_1, \ldots, x_n)$.

Randomly choose a non-linear equations on $x_1, \ldots, x_n$ for the central map F as external perturbation. Randomly choose an invertible affine map L_1 in $n + a$ dimensional vector space k^{n+a}, randomly choose an invertible affine map L_2 in n dimensional vector space k^n, and $\bar{F}(x_1, \ldots, x_n) = L_1 \circ F \circ L_2(x_1, \ldots, x_n)$ is a public key of PMI+, and the private key includes the central map F', the map $\hat{F}$, Z, L_1^{-1} and L_2^{-1}.

2.2. PMI + encryption

For a given plaintext block $(x_1, \ldots, x_n)$, when encrypting the plaintext, it only needs to apply the plaintext into the public key polynomials

$$\begin{aligned} y_1 &= \bar{f}_1(x_1, x_2, \ldots, x_n), \\ &\ldots \\ y_{n+a} &= \bar{f}_{n+a}(x_1, x_2, \ldots, x_n), \end{aligned} \tag{1}$$

to calculate the evaluation of $n + a$ quadratic polynomials that a ciphertext $(y_1, \ldots, y_{n+a})$ can be acquired.

2.3. PMI + decryption

The decryption algorithm of PMI+ is more complicated. For a given ciphertext block $(y_1, \ldots, y_{n+a})$, the decryption of the

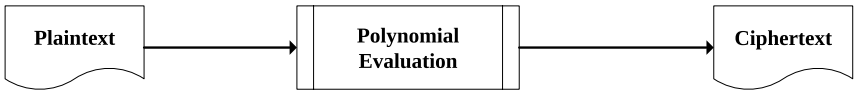


Fig. 1. The hardware structure of PMI+ encryption.

Table 1
Parameters for PMI+.

n	q	r	a	θ
84	2	4	6	14
136	2	8	6	18

ciphertext is equivalent to solve the equation set $\bar{F}(x_1, \ldots, x_n) = L_1 \circ F \circ L_2(x_1, \ldots, x_n) = (y_1, \ldots, y_{n+a})$.
It can be transformed to calculate:

$$X = (x_1, \ldots, x_n) = {L_2}^{-1} \circ F^{-1} \circ {L_1}^{-1}(y_1, \ldots, y_{n+a}). \quad (2)$$

The process is:

(1) calculating $Y' = {L_1}^{-1}(Y) = (y'_1, \ldots, y'_{n+a})$;
(2) removing external perturbation polynomials from Y' to obtain $\bar{Y} = (\bar{y}_1, \ldots, \bar{y}_n)$;
(3) calculating $(y_{\lambda 1}, \ldots, y_{\lambda n}) = F^{-1}((\bar{y}_1, \ldots, \bar{y}_n) + \lambda)$ for each $(\lambda, \mu) \in P$, and checking if $\mu = Z(y_{\lambda 1}, \ldots, y_{\lambda n})$, if not, continuing this step, otherwise, moving on to the next step;
(4) applying $(y_{\lambda 1}, \ldots, y_{\lambda n})$ into external perturbation polynomials, if the verification is successful, moving on to the next step, otherwise, returning to the previous step; and
(5) calculating $X = {L_2}^{-1}(y_{\lambda 1}, \ldots, y_{\lambda n}) = (x_1, \ldots, x_n)$, and X is the decrypted plaintext.

2.4. Security and parameter selection of PMI+

The obtained PMI+ instance can reach a corresponding security level after associated parameters are set. For example, Ding [30] has shown two sets of relatively practical PMI+ parameters in his paper that the security level can be up to over 2^{80}, and Table 1 shows the two sets of parameters.

The parameters for PMI+ encryption and decryption hardware implemented in this paper are as the first set of parameters shown in Table 1, and the security level can be up to over 2^{80}.

3. Design and implementation of PMI + hardware

3.1. Hardware structure design and algorithm process

3.1.1. Design of PMI + encryption

The hardware structure of PMI+ encryption is as shown in Fig. 1.

It can be shown from Eq. (1) in Section 2.2 that the operation process of PMI+ encryption is equivalent to applying the plaintext into the polynomials to calculate, and its hardware structure is illustrated in Fig. 1.

Algorithm 1 describes the PMI+ encryption mapping algorithm, wherein the input X is the plaintext block of PMI+ and is a n dimensional vector, and the input M is a public key multinomial coefficient of PMI+ and is $(n + a)\ (n + 1) * (n + 2)/2$ dimensional vector (because the public key multinomial is an arrangement and combination form of n factors quadratic polynomial). The output Y is the ciphertext block of PMI+ and is a $n + a$ dimensional vector. In PMI+ encryption mapping algorithm, $\oplus$ is an addition over the finite field $GF(2)$ by the operation of XOR, and $\otimes$ is a multiplication over the finite field $GF(2)$ by the operation of AND.

If parameters are selected as $(n, q, \theta, r, a) = (84, 2, 4, 6, 14)$, the length of the plaintext block is 84 bits and the length of the ciphertext block is 98 bits, it needs to add 14 external perturbations, the public key is 358,190 (3655*98) bits, i.e. 44,774 bytes.

Algorithm 1: PMI+ Encryption Mapping Algorithm

```
Input: X and M;
Output: Y;
Procedure:
1  begin
2    for (i = 0; i ≤ n + a − 1; i + +) do
3      index := 0;
4      Y_i := M_{i,index};
5      index := index + 1;
6      for (j = 0; j ≤ n − 1; j + +) do
7        Y_i := Y_i ⊕ (M_{i,index} ⊗ X_j);
8        index := index + 1;
9      end
10     for (j = 0; j ≤ n − 1; j + +) do
11       for (k = j; k ≤ n − 1; k + +) do
12         Y_i := Y_i ⊕ (M_{i,index} ⊗ X_j ⊗ X_k);
13         index := index + 1;
14       end
15     end
16   end
17   return Y;
18 end
```

3.1.2. Design of PMI + decryption

The hardware structure of PMI+ Decryption is shown in Fig. 2.

From Section 2.3, the process of decryption is equivalent to calculating Eq. (2) in Section 2.3. The process of PMI+ decryption is divided into four modules based on the process of calculating Eq. (2): affine transformation, internal perturbator, large power operation and polynomial calculation, as shown in Fig. 2. Wherein, the input of large power operation is a result of the affine transformed result adding the internal perturbator. The role of the polynomial calculation is to verify external perturbator, if the verification is successful, the result will be calculated in the affine transformation module again to obtain the plaintext block. Otherwise, select another element from the internal perturbator for large power operation after addition.

Based on Eq. (2), the process of PMI+ decryption can be abstracted into two parts: affine transformation and decryption mapping. In the parameters we selected, the process is that: firstly the ciphertext is operated by ${L_1}^{-1}$ affine transformation, wherein the parameter is 98 bits; then the result of the ${L_1}^{-1}$ affine transformation is mapped by decryption mapping algorithm to the plaintext space, and the result is 84 bits; finally the result of the PMI+ decryption mapping is operated by ${L_2}^{-1}$ affine transformation, and a 84-bit plaintext block is obtained.

Algorithm 2 describes the affine transformation algorithm, wherein the input X is a parameter of the affine transformation, which is a n dimensional vector, the input M is an affine matrix consisting of m n dimensional vectors, the input Y is an offset of the affine transformation, which is also a n dimensional vector, and the output Z is the result of the affine transformation. In the PMI+ algorithm, elements in the vector belong to finite base field $GF(2)$. In the finite field $GF(2)$, a XOR gate is used for add operation and a AND gate is used for multiply operation. In the algorithm, $\oplus$ is an addition of vectors and $\bullet$ is a dot product of vectors.

Algorithm 3 describes the PMI+ decryption mapping algorithm, wherein the input X is a parameter of the decryption mapping,

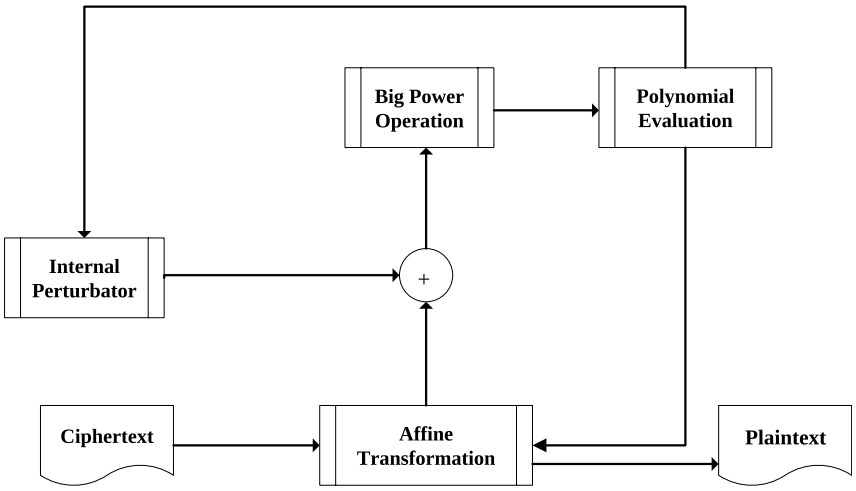


Fig. 2. The hardware structure of PMI+ decryption.

Algorithm 2: Affine Transformation Algorithm

Input: X, Y and M;
Output: Z;
Procedure:
begin
  $Y := Y \oplus X$;
  for $(i = 0; i \leq m; i++)$ **do**
    $Z_i := M_i \bullet Y$;
  end
  return Z;
end

which is a $n + a$ dimensional vector and the output of ${L_1}^{-1}$ linear affine transformation, and the output Y is the result of the inverse map of the central map, which is a n dimensional vector. In the algorithm, μ is an element in the vector space 2^r, and the whole PMI+ decryption traverses elements in the vector space 2^r. **perturbation1** is a map from the vector space 2^n to the vector space 2^r, wherein the input is a n dimensional vector and the output is a r dimensional vector. **perturbation2** is a map from the vector space 2^r to the vector space 2^n, wherein the input is a r dimensional vector and the output is a n dimensional vector. **powert** is powers of t of the n dimensional vector, wherein t is very large that is close to 2^n, so **powert** is the most important operation in the PMI+ decryption mapping algorithm. How to implement the **powert** arithmetic unit efficiently will be explained in further detail. Since the number of λ that satisfies the requirements is more than one, it needs to be checked by **checkExtraPolynomial**, which is used to check whether the calculated λ meets the external perturbation polynomial of PMI+. The process is to apply the λ into the external perturbation polynomials of PMI+ and determine whether the computed result is consistent with corresponding parts in the ciphertext. $\oplus$ in the algorithm is an addition of vectors.

3.2. Basic arithmetic unit

Firstly, the basic arithmetic units throughout the process of PMI+ encryption and decryption are described here. In our implementation, elements in $K = GF(2^{84})$ are represented by vector which can be stored in 84 bits of storage space. The basic arithmetic units described here include a full parallel multiplier, a full parallel vector dot product, a full parallel squarer and a full parallel power operator.

Algorithm 3: PMI+ Decryption Mapping Algorithm

Input: X;
Output: Y;
Procedure:
begin
  $\mu := 00\ldots0$;
  for $(i = 0; i \leq 2^r - 1; i++)$ **do**
    $\lambda :=$ perturbation2(μ);
    $Z := \lambda \oplus X$;
    $W :=$ powert(Z);
    $V :=$ perturbation1(W);
    if $\mu = V$ **then**
      if *checkExtraPolynomial*(W) **then**
        $Y := W$;
      end
    end
    $\mu = \mu + 00\ldots1$;
  end
  return Y;
end

3.2.1. Full parallel multiplier

Elements in the finite field K can be expressed by a polynomial as $a = \sum_{i=0}^{83} a_i x^i$, where $a_i \in \{0, 1\}$. A multiplication over the finite field can be expressed by $c = a \otimes b \bmod R(x) = M \bmod R(x)$.

By adopting Karatsuba multipliers [34,35] or other optimization techniques [36], we can achieve a good balance between area and speed. Table 2 lists the comparison of the related multipliers. In fact, by increasing the number of cycles, we can achieve resource optimization. Furthermore, the maximum clock frequency can be increased which will fasten the decryption. Here, a large field multiplication is completed in one clock cycle by an ordinary multiplication algorithm based on standard basis which contains merging similar items and conducting modulus reduction, and the main computation in the algorithm is on modulus reduction. The full parallel multiplier is structured as follows.

$$m_0 = a_0 \otimes b_0,$$
$$m_1 = (a_0 \otimes b_1) \oplus (a_1 \otimes b_0),$$
$$\ldots$$
$$m_{165} = (a_{82} \otimes b_{83}) \oplus (a_{83} \otimes b_{82}),$$
$$m_{166} = a_{83} \otimes b_{83};$$
$$c_0 = m_0 \oplus m_{84} \oplus \cdots \oplus m_{166},$$

Table 2
Performance comparison among some multipliers.

Multiplier	m	XOR(#)	AND(#)	Cycles
DB [36]	m	$k + w - 1$	k	q
Karatsuba [34]	128	12,352	12,288	6
Ours	84	9997	7056	1

Notation in Table 2:
m: field size; k: bit length; w: hamming weight of $R(x)$;
q: number of k-bit parts.

$$c_1 = m_1 \oplus m_{85} \oplus \cdots \oplus m_{166},$$
$$\cdots$$
$$c_{83} = m_{83} \oplus m_{110} \oplus \cdots \oplus m_{165}.$$

The full parallel multiplier can complete a multiplication over the finite field K in one clock cycle, which uses 7056 AND gate circuits and 9997 XOR gate circuits.

3.2.2. Full parallel vector dot product

In the process of PMI+ decryption, the affine transformation is used twice, where in the first time, n is 98, and in the second time, n is 84. It needs to implement two vector dot products: a 98 dimensional vector dot product and a 84 dimensional vector dot product. The scalar value in the vector is 0 or 1, so for scalar value in the vector, the addition uses a XOR gate circuit, and the multiplication uses a AND gate circuit.

Set $a = (a_0, \ldots, a_{n-1}), b = (b_0, \ldots, b_{n-1})$, where $a_i, b_i \in \{0, 1\}, i = 0, \ldots, n-1$, and the dot product of vectors is $c \in \{0, 1\} : c = (a_0 \otimes b_0) \oplus \cdots \oplus (a_{n-1} \otimes b_{n-1})$.

The vector dot product operation can be completed in one clock cycle, which uses n AND gate circuits and $n - 1$ XOR gate circuits.

3.2.3. Full parallel squarer

There is a very useful property in Frobenius mapping that for a map $T_i(X) = X^{q^i}$ over the finite field K, $q = 2$, X is represented as a polynomial basis $a_0 + a_1x + \cdots + a_{83}x^{83}$, and then the following equation holds: $T_i(X) = X^{q^i} = a_0 + a_1x^{q^i} + \cdots + a_{83}x^{83*q^i}$.

Set $a = \sum_{i=0}^{83} a_ix^i$ as any element in K, then: $a^2 = a_0 + a_1x^2 + \cdots + a_{83}x^{83*2}$. The full parallel squarer has the following hardware structure.

$$c_0 = a_0 \oplus a_{42} \oplus \cdots \oplus a_{83},$$
$$c_1 = a_{56} \oplus a_{61} \oplus \cdots \oplus a_{83},$$
$$\cdots$$
$$c_{83} = a_{55} \oplus a_{60} \oplus \cdots \oplus a_{82}.$$

The full parallel squarer can complete one squaring operation over the finite field K in one clock cycle, which uses 1525 XOR gate circuits.

3.2.4. Full parallel power operator

In order to implement the large power operation efficiently and reuse public arithmetic unit at the most extent, two power operators are implemented, where one is a full parallel power 16 operator and the other is a full parallel power 256 operator. Based on the nature of Frobenius mapping, set a as any element in K, and then:

$$a^{16} = a_0 + a_1x^{16} + \cdots + a_{83}x^{83*16},$$
$$a^{256} = a_0 + a_1x^{256} + \cdots + a_{83}x^{83*256}.$$

The full parallel power 16 operator has the following hardware structure:

$$c_0 = a_0 \oplus a_9 \oplus \cdots \oplus a_{81},$$
$$c_1 = a_1 \oplus a_7 \oplus \cdots \oplus a_{83},$$
$$\cdots$$
$$c_{83} = a_8 \oplus a_{10} \oplus \cdots \oplus a_{83}.$$

The full parallel power 256 operator has the following hardware structure:

$$d_0 = a_0 \oplus a_4 \oplus \cdots \oplus a_{83},$$
$$d_1 = a_1 \oplus a_2 \oplus \cdots \oplus a_{83},$$
$$\cdots$$
$$d_{83} = a_2 \oplus a_3 \oplus \cdots \oplus a_{83}.$$

The full parallel power operator that we implemented can complete one exponentiation over the finite field K in one clock cycle. Compared with a full parallel multiplier, the full parallel squarer uses about one in tenth logical units, and has a shorter latency.

3.3. Implementation of hardware core modules

3.3.1. Implementation of polynomial calculation

As you can see from Algorithm 1, the calculation of polynomial can be an addition or a multiplication over finite field $GF(2)$, which can be implemented by XOR operation and AND operation respectively. The polynomial calculation module is used in both PMI+ encryption and decryption. Wherein, in PMI+ encryption, the inputs of the polynomial calculation module are a plaintext block of PMI+ and a public key polynomial, the output Y of the polynomial calculation module is a ciphertext block, and the role of the polynomial calculation module is to implement PMI+ encryption; in PMI+ decryption, the inputs of the polynomial calculation module are a result of the large power operation and external perturbation polynomials of PMI+, the output of the polynomial calculation module is a result of PMI+ decryption mapping, and the role of the polynomial calculation module is to verify the external perturbation polynomials.

3.3.2. Implementation of affine transformation

As we can see from Algorithm 2, the affine transformation includes a vector addition and a vector dot product. The vector addition can be implemented by XOR operation directly. The vector dot product can be implemented by the full parallel vector dot product defined by us. In the PMI+ decryption, two affine transformations are used, respectively before and after decryption mapping, the first uses a 98 dimensional vector dot product, and the second uses a 84 dimensional vector dot product.

3.3.3. Implementation of internal perturbator

When we implement the PMI+ decryption, it needs to abstract a component to complete a transformation for mapping from $r = 6$ dimensional vector to 84 dimensional vector, which is called as internal perturbator. The expression of the map is calculated by a external program off-line, and the arithmetic unit is implemented by 1078 XOR gates and 627 AND gates.

3.3.4. Implementation of large power operation

PMI+ decryption mapping is a core algorithm of PMI+ decryption hardware, and the main computation of PMI+ decryption mapping is on large power operation, which takes up maximum calculation amount in PMI+ decryption mapping. In one PMI+ decryption, it needs 64 large power operations at most, so optimized large power operation can improve the performance of the PMI+ decryption hardware at a large extent. If the parameter t is selected as 10, 240, 312, 824, 970, 976, 538, 687, 608, it is unrealistic to find the solution of power by multiplication over the finite field K.

Conventional large power operation. The large power operation is implemented by a "square–multiplication" method. The binary

Table 3
The Performance of PMI+'s basic arithmetic units.

Arithmetic units	Area (μm^2)	Equivalent gates	Logical units	Maximum latency	Clock cycles
Full parallel multiplier	277,998	27,800	4823	27.000	1
Full parallel vector dot product	3,560	356	57	23.948	1
Full parallel squarer	19,546	1,955	289	17.483	1
Full parallel power 16 operator	37,116	3,712	510	18.641	1
Full parallel power 256 operator	39,045	3,905	538	19.191	1

equivalent for t is 10000111 10000111 10000111 10000111 10000111 10000111 10000111 10000111 10000111 10000111 1000, X^t can be expressed as $B^t = B^{2^3} \otimes B^{2^4} \otimes \cdots \otimes B^{2^{83}}$, so one large power operation can be completed by 83 squaring operations and 40 multiplications. Algorithm 4 describes a large power operation based on square–multiplication method. The input X is a 84 dimensional vector, the output Y is also a 84 dimensional vector, both of which belong to vector space 2^{84}, where T is a power exponent that is represented with a binary representation and is also a 84 dimensional vector. It needs to calculate the values from B^2 to B^{2^3} in turn by a squarer, which takes 83 clock cycles, and when T_i is 1, the multiplication can be completed together with the next square operation in one clock cycle, so one whole large power operation can be completed in 84 clock cycles.

Algorithm 4: A Large Power Operation Based On Square–Multiplication

```
Input: X;
Output: Y;
Procedure:
begin
    tmp := square(X);
    Y := 1;
    for (i = 1; i ≤ 83; i + +) do
        tmp := square(X);
        if T_i = 1 then
            Y := multiply(Y, tmp);
        end
    end
    return Y;
end
```

Optimal implementation of large power operation. The basic idea to implement large power operation is reusing public arithmetic unit at the most extent, so as to reduce clock cycles of the large power operation. The software implementation of PMI+ has been completed in a 8051 microcontroller by Chen [37] in his master dissertation, where the large power operation uses a similar idea. The differences between the above paper and this paper are that, firstly, the size of t used in this paper is different (methods for optimization are different); secondly, implementation of PMI+ in this paper is based on FPGA hardware platform.

We find that fragment $S = 10000111$ appears 10 times in the binary string, so X^t can be expressed as $X^t = X^{2^3} \otimes (X^S)^{16} \otimes ((X^S)^{16})^{256} \otimes \cdots \otimes (((X^S)^{16}) \ldots)^{256}$.

In the optimized large power operation, $X^S = X^{10000111}$ is firstly calculated, and we implement it for optimization that X^S can be calculated in 5 cycles. Then, the operation of X^t can be quickly completed by adding new arithmetic unit, and the rest of the operation can be completed in 11 clock cycles.

Algorithm 5 describes the process of the optimized large power operation where the definitions of the input X and the output Y are the same as that in Algorithm 4 and the arithmetic units of **power16** and **power256** are a full parallel power 16 operator and a full parallel power 256 operator respectively.

Algorithm 5: Optimal Implementation of Large Power Operation

```
Input: X;
Output: Y;
Procedure:
begin
    B2 := square(X);
    B4 := square(B2); Y := multiply(X, B2);
    B8 := square(B4); Y := multiply(Y, B4);
    B128 := power16(B8); B16 := square(B8);
    B135 := multiply(B128, Y);
    tmp := power16(B135);
    Y := multiply(B16, tmp); tmp = power256(tmp);
    i = 8;
    while i >= 0 do
        Y := multiply(Y, tmp); tmp = power256(tmp);
        i − −;
    end
    Y := multiply(Y, tmp);
    return Y;
end
```

Our new proposed large power operation can complete a large power operation in 16 clock cycles which are less than one-sixth of those for "square–multiplication" method, and only the logical units taken up by the arithmetic units of **power16** and **power256** increases for its area, so the operational performance is greatly enhanced.

4. Experiment results and analyses

The hardware of PMI+ encryption and decryption is implemented in *QuartusII* 8.0 environment by VHDL with the idea of high speed and parallelization. Its hardware simulation is implemented in *EP2S130F*102014 of the family of *StratixII*, and the area of PMI+ encryption and decryption hardware is evaluated by *SynopsysDC*, where the process library is a 0.18 μm process library of TSMC and the working voltage is 1.62 V. The following results come from the real experimental data and are compared with current implemented hardware in performance.

4.1. PMI + basic arithmetic unit

Some basic arithmetic units of PMI+ are implemented in Section 3.2, including a full parallel multiplier, a full parallel vector dot product, a full parallel squarer and a full parallel power operator. The performance data of these basic arithmetic units is shown in Table 3.

It can be seen from the data in Table 3 that the basic arithmetic unit in PMI+ decryption hardware can complete one basic operation in one cycle, where the full parallel multiplier takes up the maximum area, and compared with the multiplier, the squarer and power operator complete an operation with lower latency while take up less area.

Table 4
Performance comparison between two large power operation methods.

Arithmetic units	Area (μm^2)	Equivalent gates	Logical units	Clock cycles
Implementation based on "square–multiplication"	334,715	33,472	5176	84
Our optimal implementation of large power operation	435,941	43,595	6367	16

Table 5
Cycles required by arithmetic units in PMI+ encryption.

Step	Main arithmetic units	Clock cycles
1	Calculate the invertible affine map function of $L_1{}^{-1}$	85
2	Sum of the affine transformed result and the internal perturbator	1–64
3	Large power operation	16–1024
4	Calculate the map of Z	6–384
5	Calculate the invertible affine map function of $L_2{}^{-1}$	85
6	Check extra polynomial	14–56

4.2. Large power operation in PMI+

A comparison of the number of logical units and the number of clock cycles between two different large power operations is listed in Table 4.

These results show that the performance of the optimized large power operation has a significant improvement. The clock cycles of the optimized large power operation reduce by 80.9% and the area adds about 30.2% for one large power operation. In PMI+ decryption, it needs 64 large power operations at most, so it can save up to 4416 clock cycles at most for a period of decryption.

4.3. PMI + encryption and decryption

We implement the first PMI+ encryption and decryption hardware on FPGA. Compared with other public key encryption and decryption hardware, our hardware implementation of PMI+ possesses advantages such as small space, fast speed of encryption and decryption, and practical security level.

The whole PMI+ decryption needs at least 207 clock cycles (excepting cycles of reading ROM) to complete a decryption operation (see Table 5), and it takes up a total of 11,005 logical units with a area of 680,302 μm^2.

Table 6 lists performance data of the PMI+ encryption and decryption hardware. Observing the experiment data, it is easy to see the number of cycles of the PMI+ decryption is mutable, where 438 cycles for least and 2915 cycles for most, and the running speed of the PMI+ encryption hardware is faster than that of the PMI+ decryption hardware and the area of it is less than the PMI+ decryption hardware.

4.4. Performance comparison

The performance of our hardware is compared with other public key cryptographic hardware in this subsection. The comparison results are given in Table 7, where the cycle-area products are normalized to our PMI+ encryption.

The data in Table 7 shows that our design, compared with RSA and ECC, has many advantages in performance, such as small product of cycle and area, and high operating efficiency.

5. Conclusion

In mobile cloud computing, protection of critical data is the premise to ensure the completion of applications or services. In order to ensure data privacy, we design a hardware that can efficiently implement PMI+ for low-resource devices in mobile cloud computing. It is verified by experiments that our designed hardware can complete an encryption operation within 497 clock cycles, and the clock frequency can be up to 145.60 MHz, and the designed hardware can complete a decryption operation within 438 clock cycles wherein the clock frequency can be up to 132.21 MHz. Our main contributions are to develop hardware architecture of encryption and decryption of PMI+ and describe corresponding hardware algorithms. Meanwhile, basic arithmetic units are implemented with higher efficiency which can complete the operation with lesser latency. Thirdly, an optimized large power operation is implemented which needs only 16 cycles to complete one exponentiation, and compared with general power operation, it can reduce 4288 cycles at most in one process of decryption, with an obvious optimization. The experimental results also confirm that our design can be deployed in low-resource devices as thin client of mobile cloud computing.

Future studies will include: (1) using registers in hardware more accurately to reduce the area and power consumption of hardware; and (2) reducing the number of logical units of multiplier and latency on the premise that the clock cycles do not increase.

Acknowledgments

This work is supported by the National Natural Science Foundation of China under Grant Nos. U1135004 and 61170080, Guangdong Province Universities and Colleges Pearl River Scholar Funded Scheme (2011), High-level Talents Project of Guangdong Institutions of Higher Education (2012), and 973 Program under Grant Nos. 2014CB360501 and 2014CB360500.

Table 6
Performance of our PMI+ encryption and decryption.

Hardware implementation	Area (μm^2)	Equivalent gates	Logical units	Frequency (MHz)	Clock cycles	Total time (μs)
PMI+ encryption	160,385	16,039	3,468	145.60	497	3.42
PMI+ decryption	680,302	68,031	11,005	132.21	438–2915	3.31–22.05

Table 7
Performance comparison among some public key crypto hardwares.

The hardware implementation scheme	Area	Clock cycles	Frequency (MHz)	Total time (μs)	Cycle-area product
RSA1024 [8]	153,862 Gates	282,700	421.94	670	5594
ECC163 [13]	4014 Slices 42 DSP48E1s	245,430	210	1170	>2471
Our PMI+ encryption	16,039 Gates	497	145.60	3.42	1
Our PMI+ decryption	68,031 Gates	438–2915	132.21	3.31–22.05	3.74–24.91

References

[1] S. Tang, L. Xu, Towards provably secure proxy signature scheme based on isomorphisms of polynomials, Future Gener. Comput. Syst. 30 (2014) 91–97.

[2] A. Gouglidis, I. Mavridis, V. Hu, Security policy verification for multi-domains in cloud systems, Int. J. Inf. Secur. 13 (2014) 97–111.

[3] Y. Yu, M. Au, Y. Mu, S. Tang, J. Ren, W. Susilo, L. Dong, Enhanced privacy of a remote data integrity-checking protocol for secure cloud storage, Int. J. Inf. Secur. (2014) 1–12.

[4] X. He, T. Chomsiri, P. Nanda, Z. Tan, Improving cloud network security using the tree-rule firewall, Future Gener. Comput. Syst. 30 (2014) 116–126.

[5] Y. Yu, L. Niu, G. Yang, Y. Mu, W. Susilo, On the security of auditing mechanisms for secure cloud storage, Future Gener. Comput. Syst. 30 (2014) 127–132.

[6] I. Abbadi, A framework for establishing trust in cloud provenance, Int. J. Inf. Secur. 12 (2013) 111–128.

[7] G. Sutter, J. Deschamps, J. Imana, Modular multiplication and exponentiation architectures for fast RSA cryptosystem based on digit serial computation, IEEE Trans. Ind. Electron. 58 (2011) 3101–3109.

[8] A. Miyamoto, N. Homma, T. Aoki, A. Satoh, Systematic design of RSA processors based on high-radix montgomery multipliers, IEEE Trans. Very large Scale Integr. (VLSI) Syst. 19 (2011) 1136–1146.

[9] D. Wang, Y. Ding, J. Zhang, J. Hu, H. Tan, Area-efficient and ultra-low-power architecture of RSA processor for RFID, Electron. Lett. 48 (2012) 1185–1187.

[10] C. Rebeiro, S. Roy, D. Mukhopadhyay, Pushing the limits of high-speed $GF(2^m)$ elliptic curve scalar multiplication on FPGAs, in: E. Prouff, P. Schaumont (Eds.), Cryptographic Hardware and Embedded Systems—CHES 2012, in: Lecture Notes in Computer Science, vol. 7428, Springer, Berlin, Heidelberg, 2012, pp. 494–511. http://dx.doi.org/10.1007/978-3-642-33027-8_29.

[11] G. Sutter, J. Deschamps, J. Imana, Efficient elliptic curve point multiplication using digit-serial binary field operations, IEEE Trans. Ind. Electron. 60 (2013) 217–225.

[12] H. Mahdizadeh, M. Masoumi, Novel architecture for efficient FPGA implementation of elliptic curve cryptographic processor over $GF(2^{163})$, IEEE Trans. Very large Scale Integr. (VLSI) Syst. 21 (2013) 2330–2333.

[13] J. Fan, F. Vercauteren, I. Verbauwhede, Efficient hardware implementation of Fp-arithmetic for pairing-friendly curves, IEEE Trans. Comput. 61 (2012) 676–685.

[14] Z. Wang, Z. Jia, L. Ju, R. Chen, ASIP-based design and implementation of RSA for embedded systems, in: 2012 IEEE 14th International Conference on High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems, HPCC-ICESS, 2012, pp. 1375–1382. http://dx.doi.org/10.1109/HPCC.2012.202.

[15] P. Sasdrich, T. Gneysu, Efficient elliptic-curve cryptography using Curve25519 on reconfigurable devices, in: D. Goehringer, M. Santambrogio, J. Cardoso, K. Bertels (Eds.), Reconfigurable Computing: Architectures, Tools, and Applications, in: Lecture Notes in Computer Science, vol. 8405, 2014, pp. 25–36. http://dx.doi.org/10.1007/978-3-319-05960-0_3.

[16] K. Abdellatif, R. Chotin-Avot, H. Mehrez, FPGA-based high performance AES-GCM using efficient Karatsuba Ofman algorithm, in: D. Goehringer, M. Santambrogio, J. Cardoso, K. Bertels (Eds.), Reconfigurable Computing: Architectures, Tools, and Applications, in: Lecture Notes in Computer Science, vol. 8405, 2014, pp. 13–24. http://dx.doi.org/10.1007/978-3-319-05960-0_2.

[17] S.-H. Weng, Y. Zhang, J. Buckwalter, C.-K. Cheng, Energy efficiency optimization through codesign of the transmitter and receiver in high-speed on-chip interconnects, IEEE Trans. Very large Scale Integr. (VLSI) Syst. 22 (2014) 938–942.

[18] R.C. Merkle, Secrecy, authentication, and public key systems (Ph.D. thesis), Stanford University, 1979.

[19] J. Hoffstein, J. Pipher, J. Silverman, NTRU: a ring-based public key cryptosystem, in: J. Buhler (Ed.), Algorithmic Number Theory, in: Lecture Notes in Computer Science, vol. 1423, Springer, Berlin, Heidelberg, 1998, pp. 267–288. http://dx.doi.org/10.1007/BFb0054868.

[20] R.J. McEliece, A Public-key Cryptosystem Based on Algebraic Coding Theory, DSN Progress Report 42, 1978, pp. 114–116.

[21] J. Ding, B.-Y. Yang, Multivariate public key cryptography, in: D. Bernstein, J. Buchmann, E. Dahmen (Eds.), Post-Quantum Cryptography, Springer, Berlin, Heidelberg, 2009, pp. 193–241. http://dx.doi.org/10.1007/978-3-540-88702-7_6.

[22] S. Tang, H. Yi, J. Ding, H. Chen, G. Chen, High-speed hardware implementation of rainbow signature on FPGAs, in: B.-Y. Yang (Ed.), Post-Quantum Cryptography, in: Lecture Notes in Computer Science, vol. 7071, Springer, Berlin, Heidelberg, 2011, pp. 228–243. http://dx.doi.org/10.1007/978-3-642-25405-5_15.

[23] A. Shoufan, T. Wink, H. Molter, S. Huss, E. Kohnert, A novel cryptoprocessor architecture for the McEliece public-key cryptosystem, IEEE Trans. Comput. 59 (2010) 1533–1546.

[24] S. Ghosh, I. Verbauwhede, BLAKE-512 based 128-bit CCA2 secure timing attack resistant McEliece cryptoprocessor, IEEE Trans. Comput. 63 (2012) 1124–1133.

[25] J.-R. Shih, Y. Hu, M.-C. Hsiao, M.-S. Chen, W.-C. Shen, B.-Y. Yang, A.-Y. Wu, C.-M. Cheng, Securing M2M with post-quantum public-key cryptography, IEEE Journal on Emerging and Selected Topics in Circuits and Systems 3 (2013) 106–116.

[26] S. Balasubramanian, H. Carter, A. Bogdanov, A. Rupp, J. Ding, Fast multivariate signature generation in hardware: the case of rainbow, in: Application-Specific Systems, Architectures and Processors, 2008, pp. 25–30. http://dx.doi.org/10.1109/ASAP.2008.4580149.

[27] A. Bogdanov, T. Eisenbarth, A. Rupp, C. Wolf, Time-area optimized public-key engines: MQ-cryptosystems as replacement for elliptic curves? in: E. Oswald, P. Rohatgi (Eds.), Cryptographic Hardware and Embedded Systems—CHES 2008, in: Lecture Notes in Computer Science, vol. 5154, 2008, pp. 45–61. http://dx.doi.org/10.1007/978-3-540-85053-3_4.

[28] P. Czypek, S. Heyse, E. Thomae, Efficient implementations of MQPKS on constrained devices, in: E. Prouff, P. Schaumont (Eds.), Cryptographic Hardware and Embedded Systems—CHES 2012, in: Lecture Notes in Computer Science, vol. 7428, Springer, Berlin, Heidelberg, 2012, pp. 374–389. http://dx.doi.org/10.1007/978-3-642-33027-8_22.

[29] J. Ding, D. Schmidt, Z. Yin, Cryptanalysis of the new TTS scheme in CHES 2004, Int. J. Inf. Secur. 5 (2006) 231–240.

[30] J. Ding, J. Gower, Inoculating multivariate schemes against differential attacks, in: M. Yung, Y. Dodis, A. Kiayias, T. Malkin (Eds.), Public Key Cryptography—PKC 2006, in: Lecture Notes in Computer Science, vol. 3958, Springer, Berlin, Heidelberg, 2006, pp. 290–301. http://dx.doi.org/10.1007/11745853_19.

[31] T. Matsumoto, H. Imai, Public quadratic polynomial-tuples for efficient signature-verification and message-encryption, in: D. Barstow, W. Brauer, P. Brinch Hansen, D. Gries, D. Luckham, C. Moler, A. Pnueli, G. Seegmller, J. Stoer, N. Wirth, C. Gnther (Eds.), Advances in Cryptology—EUROCRYPT'88, in: Lecture Notes in Computer Science, vol. 330, Springer, Berlin, Heidelberg, 1988, pp. 419–453. http://dx.doi.org/10.1007/3-540-45961-8_39.

[32] J. Ding, A new variant of the Matsumoto–Imai cryptosystem through perturbation, in: F. Bao, R. Deng, J. Zhou (Eds.), Public Key Cryptography—PKC 2004, in: Lecture Notes in Computer Science, vol. 2947, Springer, Berlin, Heidelberg, 2004, pp. 305–318. http://dx.doi.org/10.1007/978-3-540-24632-9_22.

[33] P.-A. Fouque, L. Granboulan, J. Stern, Differential cryptanalysis for multivariate schemes, in: Eurocrypt 2005, in: Lecture Notes in Computer Science, vol. 3494, Springer, 2005, pp. 341–353.

[34] G. Zhou, H. Michalik, L. Hinsenkamp, Complexity analysis and efficient implementations of bit parallel finite field multipliers based on Karatsuba–Ofman algorithm on FPGAs, IEEE Trans. Very large Scale Integr. (VLSI) Syst. 18 (2010) 1057–1066.

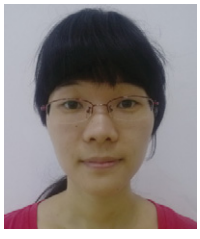
[35] J. Lima, D. Panario, Q. Wang, A Karatsuba-based algorithm for polynomial multiplication in Chebyshev form, IEEE Trans. Comput. 59 (2010) 835–841.

[36] S. Bayat-Sarmadi, M. Kermani, R. Azarderakhsh, C.-Y. Lee, Dual-basis superserial multipliers for secure applications and lightweight cryptographic architectures, IEEE Trans. Circuits Syst. Express Briefs 61 (2014) 125–129.

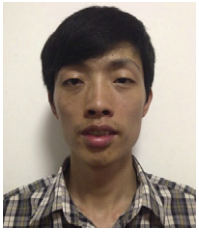
[37] Y. Chen, An implementation of PMI+ on low-cost smartcard (Master's thesis), National Taiwan University, 2006.

[38] S. Tang, B. Lv, G. Chen, Z. Peng, Efficient hardware implementation of MQ asymmetric cipher PMI+ on FPGAs, in: X. Huang, J. Zhou (Eds.), Information Security Practice and Experience, in: Lecture Notes in Computer Science, vol. 8434, 2014, pp. 187–201. http://dx.doi.org/10.1007/978-3-319-06320-1_15. URL: http://dx.doi.org/10.1007/978-3-319-06320-1_15.

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. He is a member of the IEEE and the IEEE Computer Society.



Bo Lv is a doctoral candidate at the School of Computer Science and Engineering, South China University of Technology. Her current research interests include multivariate public key cryptography, crypto chip design, and provably-secure cryptography.



Guomin Chen is a postgraduate student at the School of Computer Science and Engineering, South China University of Technology. His research interests include multivariate public key cryptography, algorithm design and analysis.

Zhiniang Peng is a doctoral candidate at the School of Computer Science and Engineering, South China University of Technology. His current research interests include multilinear map, crypto chip design, and multivariate public key cryptography.

Adama Diene is currently employed at the United Arab Emirates University in Al Ain, Abu Dhabi UAE, as an Associate Professor in Mathematical Sciences. He received his education first at the University of Montreal where he got his B.S. and M.S. in Mathematics, before joining the University of Cincinnati in Ohio to pursue his Ph.D. in Applied Algebra and Post Quantum Cryptography. His fields of interest include Cryptography, Applied Algebra, finite fields, and Number Theory.

Xiaofeng Chen received his B.S. and M.S. on Mathematics in Northwest University, China. He got his Ph.D. degree in Cryptography in Xidian University in 2003. Currently, he works at Xidian University as a professor. His research interests include applied cryptography and cloud computing security.