Efficient Hardware Implementation of MQ Asymmetric Cipher PMI+ on FPGAs

Shaohua Tang * and Bo Lv ** ,Guomin Chen ** and Zhiniang Peng

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

Abstract. PMI+ is a Multivariate Quadratic (MQ) public key algorithm used for encryption and decryption operations, and belongs to post quantum cryptography. We designs a hardware on FPGAs to efficiently implement PMI+ in this paper. Our main contributions are that, firstly, a hardware architecture of encryption and decryption of PMI+ is developed, and description of corresponding hardware algorithm is 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, an optimized implementation for core module, including optimized large power operation, is 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 145.6MHz, and the designed hardware can complete a decryption operation within 438 clock cycles wherein the clock frequency can be up to 37.04MHz.

Keywords: Multivariate Quadratic (MQ) Public Key Algorithm, PMI+ Encryption and Decryption, Hardware Implementation, FPGA, Optimized Large Power Operation

1 Introduction

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 based on integer factorization problem, ElGamal based on discrete logarithm problem and elliptic curve cryptography, etc. In order to adapt various occasions, many efficiently hardware implementations are proposed by researchers [21,17,24,18,22,13,7].

The quantum algorithm of P.Shor is able to solve the integer factorization and discrete logarithm problem in polynomial time, including a calculation problem in elliptic curve field, which directly threatens classical cryptosystems based

* Corresponding Author: Shaohua Tang.
** The author Bo Lv and Guomin Chen contribute equally to this paper.

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[16], lattice-based public key cryptosystem[12], public key cryptosystem based on error correcting code[15] and multivariate public key cryptosystem[6]. 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[23,20,10,19,1,2,3].

PMI+ [5] is one kind of multivariate public key cryptosystem, and is a variant of MI[14]. Ding enhanced the security of MI by adding internal perturbation to the central map of MI in 2004, to produce a new variant of the MI cryptosystem which is called PMI cryptosystem[4]. However, the PMI cryptosystem has been broken by differential cryptanalysis by Fouque et al.[9] in 2005. Ding introduced new external perturbation to the central mapping of MI [5] in 2006, to produce PMI+ cryptosystem whose security has been greatly improved. Up to present, the PMI+ cryptosystem is still secure, and its hardware implementation is relatively less, so a hardware used to implement PMI+ is designed in this paper, which can be efficiently implemented in FPGA.

Our Contributions. The paper designs a hardware used to implement PMI+, which can be efficiently implemented on FPGA.

Firstly, a hardware architecture of encryption and decryption of PMI+ is developed, and description of corresponding hardware algorithm is 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 unit and has shorter latency.

Thirdly, we implement an optimized large power operation, and compared with general power operation, it can 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 $3.42us$, wherein the clock frequency can be up to 145.6MHz, and our designed hardware can complete an decryption operation within 438 clock cycles or $11.83us$, wherein the clock frequency can be up to 37.04MHz.

Organization. The structure of the rest of this paper goes as follows. Section 2 briefly introduces solution and theory of PMI+ encryption scheme, including the construction of algorithms, principles of encryption and decryption and the choice of parameters; Section 3 primarily focuses on hardware design and implementation of PMI+, including hardware structure design, algorithm description and implementation of basic arithmetic unit and hardware core module; Section 4 lists detailed experimental data, and makes performance contrast with other public key encryption schemes; and Section 5 is the conclusion of this paper, which summarizes the findings of this paper and proposes further research directions.

2 Preliminaries

We describes the basic theory of the encryption and decryption of PMI+ [5] in this section. The basic idea of PMI+ is adding internal perturbation and external perturbation to the central map of MI scheme to resist linearization equation attack and differential attack.

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 $\varphi(a_0 + a_1x + ... + a_{n-1}x^{n-1}) = (a_0, a_1, ..., a_{n-1})$.

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 $F'(x_1, ..., x_n) = \varphi \circ \tilde{F} \circ \varphi^{-1}(x_1, ..., x_n)$.

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

Randomly choose n quadratic polynomials $\hat{f}_1, ..., \hat{f}_n \in k[z_1, ..., z_r]$. The $\hat{f}_i$ define a map $\hat{F} : k^r \to k^n$ by $\hat{F}(z_1, ..., z_r) = (\hat{f}_1, ..., \hat{f}_n)$. 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 $F^*(x_1, ..., x_n) = \hat{F} \circ Z(x_1, ..., x_n) = (f_1^*, ..., f_n^*)$. Define a map by $F(x_1, ..., x_n) = (F' + F^*)(x_1, ..., x_n)$.

Randomly choose a non-linear equations on $x_1, ..., x_n$ 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 $\bar{F}(x_1, ..., x_n) = L_1 \circ F \circ L_2(x_1, ..., x_n)$ 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 $(x_1, ..., x_n)$, when encrypting the plaintext, it only needs to apply the plaintext into the public key polynomial

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

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

2.3 PMI+ Decryption

We can decrypt the ciphertext $(y_1, ..., y_{n+a})$ by computing

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

The process is:

(1) calculating $Y' = {L_1}^{-1}(Y) = (y'_1, ..., y'_{n+a})$;

(2) removing a external perturbation polynomials from Y' to obtain $\bar{Y} = (\bar{y}_1, ..., \bar{y}_n)$;

(3) calculating $(y_{\lambda 1}, ..., y_{\lambda n}) = F^{-1}((\bar{y}_1, ..., \bar{y}_n) + \lambda)$ for each $(\lambda, \mu) \in P$, and checking if $\mu = Z(y_{\lambda 1}, ..., y_{\lambda n})$, if not, continuing this step, otherwise, moving on to the next step;

(4) applying $(y_{\lambda 1}, ..., y_{\lambda n})$ into a external perturbation polynomials, if the verification is successful, moving on to the next step, otherwise, returning to the previous step; and

(5) calculating $X = {L_2}^{-1}(y_{\lambda 1}, ..., y_{\lambda n}) = (x_1, ..., x_n)$, and X is a 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 [5] has shown two sets of relatively practical PMI+ parameters in his paper that the security level can be up to over 2^{80}, and the following table shows the two sets of parameters.

Table 1. Parameters for PMI+

n	q	r	a	θ
84	2	4	6	14
136	2	8	6	18

The parameters for PMI+ encryption and decryption hardware implemented in this paper is 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

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

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

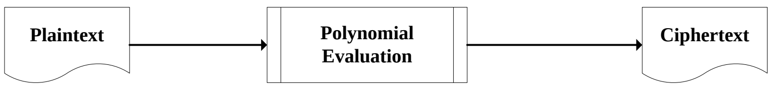

Fig. 1. The Hardware Structure of PMI+ Encryption

The 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 polynomial coefficient of PMI+ and is $(n + a)$ $(n + 1) * (n + 2)/2$ dimensional vector (because the public key polynomial 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 vectors. In PMI+ encryption mapping algorithm, $\oplus$ is an addition over the finite field $GF(2)$ by the operation of XOR, and $\otimes$ is an addition over the finite field $GF(2)$ by the operation of AND.

Algorithm 1: PMI+ Encryption Mapping Algorithm

```
Input: $X$ and $M$;
Output: $Y$;
Procedure:
begin
    for $(i = 0; i \leq n + a - 1; i + +)$ do
        index: = 0;
        $Y_i := M_{i,index}$;
        index: = index + 1;
        for $(j = 0; j \leq n - 1; j + +)$ do
            $Y_i := Y_i \oplus (M_{i,index} \otimes X_j)$;
            index: = index + 1;
        end
        for $(j = 0; j \leq n - 1; j + +)$ do
            for $(k = j; k \leq n - 1; k + +)$ do
                $Y_i := Y_i \oplus (M_{i,index} \otimes X_j \otimes X_k)$;
                index: = index + 1;
            end
        end
    end
    return $Y$;
end
```

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 (3,655*98) bits, i.e. 44,774 bytes.

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

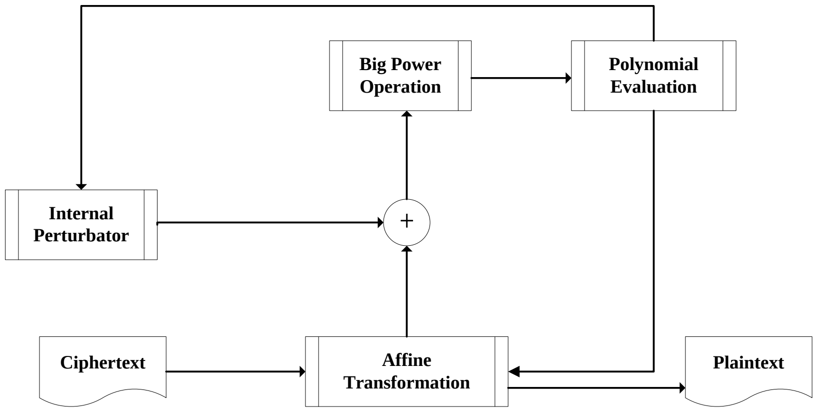

Fig. 2. The hardware structure of PMI+ Decryption

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, to 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 vectors of n-dimension, 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 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, and in the algorithm, $\oplus$ is an addition of vectors and $\bullet$ is a dot product multiply of vectors.

Algorithm 2: Affine Transformation Algorithm

```
  Input: $X, Y$ and $M$;
  Output: $Z$;
  Procedure:
1 begin
2     $Y := Y \oplus X$;
3     for $(i = 0; i \leq m; i++)$ do
4         $Z_i := M_i \bullet Y$;
5     end
6     return $Z$;
7 end
```

Algorithm 3 describes the PMI+ decryption mapping algorithm, wherein the input X is a parameter of the decryption mapping, 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 to implement 2^r loops for calculating. **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. Because the λ that satisfy the requirements is more than one, it needs to be checked by **checkExtraPolynomial**, the **checkExtraPolynomial** is used to check whether the calculated λ meets the external perturbation polynomial of PMI+, and the process is to apply the λ into a 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 unit throughout the process of PMI+ encryption and decryption is described here.

Algorithm 3: PMI+ Decryption Mapping Algorithm

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

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 $a_i \in \{0, 1\}$. And a multiplication over the finite field can be expressed by $c = a \otimes b \bmod R(x) = M \bmod R(x)$.

One large field multiplication can be 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. It can be pre-processed with external program. The full parallel multiplier is structured as follows.

$$\begin{aligned}
&m_0 = a_0 \otimes b_0,\\
&m_1 = (a_0 \otimes b_1) \oplus (a_1 \otimes b_0),\\
&...\\
&m_{165} = (a_{82} \otimes b_{83}) \oplus (a_{83} \otimes b_{82}),\\
&m_{166} = a_{83} \otimes b_{83};
\end{aligned}$$

$$\begin{aligned}
&c_0 = m_0 \oplus m_{84} \oplus ... \oplus m_{166},\\
&c_1 = m_1 \oplus m_{85} \oplus ... \oplus m_{166},\\
&...\\
&c_{83} = m_{83} \oplus m_{110} \oplus ... \oplus m_{165}.
\end{aligned}$$

The full parallel multiplier can complete one multiplication over the finite field K in one clock cycle, which uses 7,056 AND gate circuits and 9,997 XOR gate circuits. It was unrealistic to implement a direct look-up table over finite field $GF(2^{84})$ (the storage space of the table can be up to 2^{168} bits), in comparison the full parallel multiplier over the finite field K implemented in this paper should be the better.

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

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

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, X is represented as a polynomial basis $a_0 + a_1x + ... + a_{83}x^{83}$, and then the following equation holds: $T_i(X) = X^{q^i} = a_0 + a_1x^{q^i} + ... + a_{83}x^{83*q^i}$.

While in the finite field K, $q = 2$, set $a = \sum_{i=0}^{83} a_ix^i$ as any element in K, then:$a^2 = a_0 + a_1x^2 + ... + a_{83}x^{83*2}$. It can be pre-processed with external program. The full parallel squarer has the following hardware structure.

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

The full parallel squarer can complete one squaring operation over the finite field K in one clock cycle, which uses 1,525 XOR gate circuits. Compared with a full parallel multiplier, the full parallel squarer uses about one in twenty logical units, and has a shorter latency, so it seems worthwhile to implement the full parallel squarer.

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:

$$\begin{aligned}
&a^{16} = a_0 + a_1x^{16} + ... + a_{83}x^{83*16}, \\
&a^{256} = a_0 + a_1x^{256} + ... + a_{83}x^{83*256}.
\end{aligned}$$

It can be pre-processed with external program. The full parallel power 16 operator has the following hardware structure:

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

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

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

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

Implementation of Polynomial Calculation. The calculation of polynomial can be an addition or 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 input of the polynomial calculation module is 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 input of the polynomial calculation module is a result of the large power operation and a 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 a external perturbation polynomials.

Implementation of Affine Transformation. 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.

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 1,078 XOR gates and 627 AND gates.

Implementation of Large Power Operation. 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 10240312824970976538687608, 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 equivalent for t is 10000111 10000111 10000111 10000111 10000111 10000111 10000111 10000111 10000111 10000111 1000, X^t can be expressed as $B^t = B^{2^3} \otimes B^{2^4} \otimes ... \otimes B^{2^{83}}$, 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}, the arithmetic units of **square** and **multiply** are a full parallel squarer and a full parallel multiplier respectively, 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 ($i = 1; i \leq 83; i++$) 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 [25] in his master dissertation, where the large power operation uses a similar idea. The differences between the above paper and this paper are that the size of t used in this paper is different (methods for optimization are different), and the 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 $X^t = X^{2^3} \otimes (X^S)^{16} \otimes ((X^S)^{16})^{256} \otimes ... \otimes ((X^S)^{16})...)^{256}$.

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: Optimal Implementation of Large Power Operation

```
   Input: $X$;
   Output: $Y$;
   Procedure:
1  begin
2      $B2 := \text{square}(X)$;
3      $B4 := \text{square}(B2)$; $Y := \text{multiply}(X, B2)$;
4      $B8 := \text{square}(B4)$; $Y := \text{multiply}(Y, B4)$;
5      $B128 := \text{power16}(B8)$; $B16 := \text{square}(B8)$;
6      $B135 := \text{multiply}(B128, Y)$;
7      $tmp := \text{power16}(B135)$;
8      $Y := \text{multiply}(B16, \text{tmp})$;$\text{tmp} = \text{power256}(\text{tmp})$;
9      $i = 8$;
10     while $i >= 0$ do
11         $Y := \text{multiply}(Y, \text{tmp})$; $\text{tmp} = \text{power256}(\text{tmp})$;
12         $i--$;
13     end
14     $Y := \text{multiply}(Y, \text{tmp})$;
15     return $Y$;
16 end
```

Algorithm 5 describes the process of the optimized large power operation. The input X is a 84 dimensional vector, the output Y is also a 84 dimensional vector, the arithmetic units of **square** and **multiply** are a full parallel squarer and a full parallel multiplier respectively, and the arithmetic units of **power16** and **power256** are a full parallel power 16 operator and a full parallel power 256 operator respectively.

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 algorithm of PMI+ encryption and decryption is implemented in *Quartas II* 8.0 environment by VHDL with the idea of high speed and parallelization, its hardware simulation is implemented in $EP2S130F102014$ of the family of $StratixII$, and the area of PMI+ encryption and decryption hardware is evaluated by $SynopsysDC$, where the process library is 0.18 *nm* process library of TSMC and the working voltage is 1.62 *volt*. The following results come from the real experimental data and 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, and the performance data of these basic arithmetic units is shown in Table 2.

Table 2. The Performance of PMI+'s Basic Arithmetic Units

Arithmetic Units	Area (um^2)	Number of Equivalent Gate	Logical Unit (ALUT)	Maximum Latency	Clock Cycles
Full Parallel Multiplier	277997.2	27800	4823	27.000	1
Full Parallel Vector Dot Product	3559.25	356	57	23.948	1
Full Parallel Squarer	19546	1955	289	17.483	1
Full Parallel Power 16 Operator	37115.8	3712	510	18.641	1
Full Parallel Power 256 Operator	39045	3905	538	19.191	1

It can be seen from the data in Table 2 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.

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 3.

Table 3. Performance Comparison between two Large Power Operation Methods

Arithmetic Units	Area/ (um^2)	Number of Equivalent Gate	Logical Unit (ALUT)	Clock Cycles
Implementation Based on "Square - Multiplication"	334715	33472	5176	84
Our Optimal Implementation of Large Power Operation	435941	43595	6367	16

These results show that the performance of the optimized large power operation has a significant improvement that 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 4,416 clock cycle 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 of advantages such as small space, fast speed of encryption and decryption, and practical security level.

Table 4. 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 Enternal 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

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

Table 5. Performance of our PMI+ Encryption and Decryption

Hardware Implementation	Area (um^2)	Number of Equivalent Gate	Logical Unit (A-LUT)	Clock Frequency (MHz)	Period (ns)	Clock Cycles	Total Time (us)
PMI+ Encryption	160385	16039	3468	145.60	6.868	497	3.42
PMI+ Decryption	680302	68031	11005	37.04	27.000	438-2915	11.83-78.71

Table 5 lists performance data of the PMI+ encryption and decryption hardware. Using experiment data, it's easy to see the number of cycles of the PMI+ decryption is mutable, where 438 cycles for least and 2,915 cycles for most, and the running speed of the PMI+ encryption hardware is far faster than that of the PMI+ decryption hardware and the area of it is far less than the PMI+ decryption hardware.

4.4 Performance Comparison

The performances of the PMI+ decryption hardware is compared with other public key cryptosystem hardware in this section. Table 6 lists the results after comparing the implementation of the PMI+ decryption hardware with other public key cryptosystems.

Table 6. Performance Comparison among some Public Key Crypto Hardwares

The Hardware Implementation Scheme	Number of Equivalent Gate	Clock Cycles	Frequency (MHz)	Total time (us)	Area*time
RSA1024-PSS[11]	250000	357142	200	1785.71	554.70
ECC128[8]	183000	592976	204	2910	661.69
EN-TTS[26]	21000	60000	67	895.53	23.37
Our Parallelized PMI+ Decryption	68031	438-2915	37.04	11.83-78.71	1-6.66

The data in the table shows that compared with RSA and ECC, parallelization PMI+ decryption hardware that we implemented has a higher performance advantage, such as small product of area and time, and high operating efficiency.

5 Conclusion

We design a hardware on FPGAs used to efficiently implement PMI+. 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 145.6MHz, and the designed hardware can complete a decryption operation within 438 clock cycles wherein the clock frequency can be up to 37.04MHz. 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 in this paper 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.

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.

Acknowledgment

This work is supported by the National Natural Science Foundation of China under Grant No. U1135004 and 61170080, Guangdong Province Universities and Colleges Pearl River Scholar Funded Scheme (2011), and High-level Talents Project of Guangdong Institutions of Higher Education (2012).

References

1. Balasubramanian, S., Carter, H., Bogdanov, A., Rupp, A., Ding, J.: Fast Multivariate Signature Generation in Hardware: The Case of Rainbow. In: Application-Specific Systems, Architectures and Processors, 2008. ASAP 2008. International Conference on. pp. 25–30 (2008)
2. Bogdanov, A., Eisenbarth, T., Rupp, A., Wolf, C.: Time-Area Optimized Public-Key Engines: MQ-Cryptosystems as Replacement for Elliptic Curves? (2008), a revised version of the original paper accepted for CHES 2008 abogdanov@crypto.rub.de 14101 received 10 Aug 2008
3. Czypek, P., Heyse, S., Thomae, E.: Efficient Implementations of MQPKS on Constrained Devices. In: Prouff, E., Schaumont, P. (eds.) Cryptographic Hardware and Embedded Systems CHES 2012, Lecture Notes in Computer Science, vol. 7428, pp. 374–389. Springer Berlin Heidelberg (2012)
4. Ding, J.: A New Variant of the Matsumoto-Imai Cryptosystem through Perturbation. In: Bao, F., Deng, R., Zhou, J. (eds.) Public Key Cryptography PKC 2004, Lecture Notes in Computer Science, vol. 2947, pp. 305–318. Springer Berlin Heidelberg (2004)

5. Ding, J., Gower, J.: Inoculating Multivariate Schemes Against Differential Attacks. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) Public Key Cryptography - PKC 2006, Lecture Notes in Computer Science, vol. 3958, pp. 290–301. Springer Berlin Heidelberg (2006)
6. Ding, J., Yang, B.Y.: Multivariate Public Key Cryptography. In: Bernstein, D., Buchmann, J., Dahmen, E. (eds.) Post-Quantum Cryptography, pp. 193–241. Springer Berlin Heidelberg (2009)
7. Fan, J., Vercauteren, F., Verbauwhede, I.: Efficient Hardware Implementation of Fp-Arithmetic for Pairing-Friendly Curves. Computers, IEEE Transactions on 61(5), 676–685 (2012)
8. Fan, J., Vercauteren, F., Verbauwhede, I.: Faster Fp-arithmetic for Cryptographic Pairings on Barreto-Naehrig Curves
9. alain Fouque, P., Granboulan, L., Stern, J.: Differential Cryptanalysis for Multivariate Schemes. In: In Eurocrypt 2005, LNCS 3494:341353. pp. 341–353. Springer (2005)
10. Ghosh, S., Verbauwhede, I.: BLAKE-512 Based 128-bit CCA2 Secure Timing Attack Resistant McEliece Cryptoprocessor. IEEE Transactions on Computers 99(PrePrints), 1 (2012)
11. Groschdl, J.: High-Speed RSA Hardware Based on Barret's Modular Reduction Method. In: Proceedings of the Second International Workshop on Cryptographic Hardware and Embedded Systems. pp. 191–203. CHES '00, Springer-Verlag, London, UK, UK (2000)
12. Hoffstein, J., Pipher, J., Silverman, J.: NTRU: A Ring-Based Public Key Cryptosystem. In: Buhler, J. (ed.) Algorithmic Number Theory, Lecture Notes in Computer Science, vol. 1423, pp. 267–288. Springer Berlin Heidelberg (1998)
13. Mahdizadeh, H., Masoumi, M.: Novel Architecture for Efficient FPGA Implementation of Elliptic Curve Cryptographic Processor Over $GF(2^{163})$. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 21(12), 2330–2333 (2013)
14. Matsumoto, T., Imai, H.: Public Quadratic Polynomial-Tuples for Efficient Signature-Verification and Message-Encryption. pp. 419–453. Springer-Verlag (1998)
15. McEliece, R.J.: A Public-Key Cryptosystem Based on Algebraic Coding Theory. DSN progress report 42(44), 114–116 (1978)
16. Merkle, R.C.: Secrecy, Authentication, and Public Key Systems. Ph.D. thesis, Stanford, CA, USA (1979), aAI8001972
17. Miyamoto, A., Homma, N., Aoki, T., Satoh, A.: Systematic Design of RSA Processors Based on High-Radix Montgomery Multipliers. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 19(7), 1136–1146 (2011)
18. Rebeiro, C., Roy, S., Mukhopadhyay, D.: Pushing the Limits of High-Speed GF(2^m) Elliptic Curve Scalar Multiplication on FPGAs. In: Prouff, E., Schaumont, P. (eds.) Cryptographic Hardware and Embedded Systems CHES 2012, Lecture Notes in Computer Science, vol. 7428, pp. 494–511. Springer Berlin Heidelberg (2012)
19. Shih, J.R., Hu, Y., Hsiao, M.C., Chen, M.S., Shen, W.C., Yang, B.Y., Wu, A.Y., Cheng, C.M.: Securing M2M With Post-Quantum Public-Key Cryptography. Emerging and Selected Topics in Circuits and Systems, IEEE Journal on 3(1), 106–116 (2013)
20. Shoufan, A., Wink, T., Molter, H., Huss, S., Kohnert, E.: A Novel Cryptoprocessor Architecture for the McEliece Public-Key Cryptosystem. Computers, IEEE Transactions on 59(11), 1533–1546 (2010)

21. Sutter, G., Deschamps, J., Imana, J.: Modular Multiplication and Exponentiation Architectures for Fast RSA Cryptosystem Based on Digit Serial Computation. Industrial Electronics, IEEE Transactions on 58(7), 3101–3109 (2011)
22. Sutter, G., Deschamps, J., Imana, J.: Efficient Elliptic Curve Point Multiplication Using Digit-Serial Binary Field Operations. Industrial Electronics, IEEE Transactions on 60(1), 217–225 (2013)
23. Tang, S., Yi, H., Ding, J., Chen, H., Chen, G.: High-Speed Hardware Implementation of Rainbow Signature on FPGAs. In: Yang, B.Y. (ed.) Post-Quantum Cryptography, Lecture Notes in Computer Science, vol. 7071, pp. 228–243. Springer Berlin Heidelberg (2011)
24. Wang, D., Ding, Y., Zhang, J., Hu, J., Tan, H.: Area-Efficient and Ultra-Low-Power Architecture of RSA Processor for RFID. Electronics Letters 48(19), 1185–1187 (2012)
25. Y, C.: An Implementation of PMI+ on Low-Cost SmartCard. Master's thesis, National Taiwan University
26. Yang, B.Y., Cheng, C.M., Chen, B.R., Chen, J.M.: Implementing Minimized Multivariate PKC on Low-Resource Embedded Systems. In: Clark, J., Paige, R., Polack, F., Brooke, P. (eds.) Security in Pervasive Computing, Lecture Notes in Computer Science, vol. 3934, pp. 73–88. Springer Berlin Heidelberg (2006)