High-Speed Hardware Implementation of Rainbow Signature on FPGAs

Shaohua Tang[1], Haibo Yi[1], Jintai Ding[2,3], Huan Chen[1], and Guomin Chen[1]

[1] School of Computer Science & Engineering,
South China University of Technology, Guangzhou, China
shtang@IEEE.org, {haibo.yi87,sarlmolapple}@gmail.com, huangege@qq.com
[2] Department of Applied Mathematics,
South China University of Technology, Guangzhou, China
[3] Department of Mathematical Sciences,
University of Cincinnati, OH, USA
jintai.ding@mail.uc.edu

Abstract. We propose a new efficient hardware implementation of Rainbow signature scheme. We enhance the implementation in three directions. First, we develop a new parallel hardware design for the Gauss-Jordan elimination, and solve a 12×12 system of linear equations with only 12 clock cycles. Second, a novel multiplier is designed to speed up multiplication of three elements over a finite field. Third, we design a novel partial multiplicative inverter to speed up the multiplicative inversion of finite field elements. Through further other minor optimizations of the parallelization process and by integrating the major optimizations above, we build a new hardware implementation, which takes only 198 clock cycles to generate a Rainbow signature, a new record in generating digital signatures and four times faster than the 804-clock-cycle Balasubramanian-Bogdanov-Carter-Ding-Rupp design with similar parameters.

Keywords: Multivariate Public Key Cryptosystems (MPKCs), digital signature, Rainbow, finite field, Field-Programmable Gate Array (FPGA), Gauss-Jordan elimination, multiplication of three elements.

1 Introduction

Due to the fast growth of broad application of cryptography, the use of secure and efficient hardware architectures for implementations of cryptosystems receives considerable attention. In terms of asymmetric cryptosystems, most schemes currently used are based on the hardness of factoring large numbers or discrete logarithm problems. However, a potential powerful quantum computer could put much of currently used public key cryptosystems in jeopardy due to the algorithm by Peter Shor [1].

Multivariate Public Key Cryptosystems (MPKCs) [2] is one of main families of public key cryptosytsems that have the potential to resist the attacks by

B.-Y. Yang (Ed.): PQCrypto 2011, LNCS 7071, pp. 228–243, 2011.

quantum computation. They are based on the difficulty of the problem of solving multivariate quadratic equations over finite fields, which is in general NP-hard.

The focus of this paper is to further speed up hardware implementation of Rainbow signature generation (without consideration of the area cost). The Oil-Vinegar family of Multivariate Public Key Cryptosystems consists of three families: balanced Oil-Vinegar, unbalanced Oil-Vinegar and Rainbow [3], a multilayer construction using unbalanced Oil-Vinegar at each layer. There have been some previous works to efficiently implement multivariate signature schemes, e.g. TTS on a low-cost smart card [4], minimized multivariate PKC on low-resource embedded systems [5], some instances of MPKCs [6], SSE implementation of multivariate PKCs on modern x86 CPUs [7]. Currently the best hardware implementations of Rainbow signature are:

1. A parallel hardware implementation of Rainbow signature scheme [8], the fastest work (not best in area utilization), which takes 804 clock cycles to generate a Rainbow signature;
2. A hardware implementation of multivariate signatures using systolic arrays [9], which optimizes in terms of certain trade-off between speed and area.

In generation of Rainbow signature, the major computation components are: 1. Multiplication of elements in finite fields; 2. Multiplicative inversion of elements in finite fields; 3. Solving system of linear equations over finite fields. Therefore, we focus on further improvement in these three directions.

Our contributions. In terms of multiplication over finite fields, we improve the multiplication according to the design in [10]. In terms of solving system of linear equations, our improvements are based on a parallel Gaussian elimination over $GF(2)$ [11], a systolic Gaussian elimination for computing multiplicative inversion [12], and a systolic Gauss-Jordan elimination over $GF(2^n)$ [13], and develop a new parallel hardware design for the Gauss-Jordan elimination to solve a 12×12 system of linear equations with only 12 clock cycles. In terms of multiplicative inversion, we design a novel partial multiplicative inverter based on Fermat's theorem.

Through further other minor optimizations of the parallelization process and by integrating the major optimizations above, we build a new hardware implementation, which takes only 198 clock cycles to generate a Rainbow signature, a new record in generating digital signatures and four times faster than the 804-clock-cycle Balasubramanian-Bogdanov-Carter-Ding-Rupp design [8] with similar parameters.

We test and verify our design on a Field-Programmable Gate Array (FPGA), the experimental results confirm our estimates.

The rest of this paper is organized as follows: in Section 2, we present the background information used in this paper; in Section 3, the proposed hardware design for Rainbow signature scheme is presented; in Section 4, we implement our design in a low-cost FPGA and experimental results are presented; in Section 5, the implementation is evaluated and compared with other hardware implementations; in Section 6, conclusions are summarized.

2 Background

2.1 Definitions

A finite field, $GF(2^8)$, including its additive and multiplicative structure, is denoted by k; The number of variables used in the signature construction, which is also equal to the signature size, is denoted by n.

For a Rainbow scheme, the number of Vinegar variables used in the i^{th} layer of signature construction is denoted by v_i; the number of Oil variables used in the i^{th} layer of signature construction is denoted by o_i, and $o_i = v_{i+1} - v_i$; the number of layers is denoted by u, a message (or the hash value of a message) is denoted by Y; the signature of Rainbow is denoted by X'; O_i is a set of Oil variables in the the i^{th} layer; S_i is a set of Vinegar variables in the the i^{th} layer.

Rainbow scheme belongs to the class of Oil-Vinegar signature constructions. The scheme consists of a quadratic system of equations involving Oil and Vinegar variables that are solved iteratively. The Oil-Vinegar polynomial can be represented by the form

$$\sum_{i\in O_l, j\in S_l} \alpha_{ij}x_ix_j + \sum_{i,j\in S_l} \beta_{ij}x_ix_j + \sum_{i\in S_{l+1}} \gamma_i x_i + \eta. \tag{1}$$

2.2 Overview of Rainbow Scheme

Rainbow scheme consists of four components: private key, public key, signature generation and signature verification.

Private Key. The private key consists of two affine transformations ${L_1}^{-1}$, ${L_2}^{-1}$ and the center mapping F, which is held by the signer. L_1: $k^{n-v_1} \to k^{n-v_1}$ and L_2: $k^n \to k^n$ are two randomly chosen invertible affine linear transformations. F is a map consists of $n - v_1$ Oil-Vinegar polynomials. F has $u - 1$ layers of Oil-Vinegar construction. The first layer consists of o_1 polynomials where $\{x_i | i \in O_1\}$ are the Oil variables, and $\{x_j | j \in S_1\}$ are the Vinegar variables. The l^{th} layer consists of o_l polynomials where $\{x_i | i \in O_l\}$ are the Oil variables, and $\{x_j | j \in S_l\}$ are the Vinegar variables.

Public Key. The public key consists of the field k and the $n - v_1$ polynomial components of $\overline{F}$, where $\overline{F} = L_1 \circ F \circ L_2$.

Signature Generation. The message is defined by $Y = (y_1, ..., y_{n-v_1}) \in k^{n-v_1}$, and the signature is derived by computing ${L_2}^{-1} \circ F^{-1} \circ {L_1}^{-1}(Y)$.

Therefore, first we should compute $\overline{Y'} = {L_1}^{-1}(Y)$, which is a computation of an affine transformation (i.e. vector addition and matrix-vector multiplication).

Next, to solve the equation $\overline{Y'} = F$, at each layer, the v_i Vinegar variables in the Oil-Vinegar polynomials are randomly chosen and the variables at upper layer are chosen as part of the Vinegar variables. After that, the Vinegar variables are substituted into the multivariate polynomials to derive a set of linear equations with only Oil variables of that layer. If these equations have a solution, we move to next layer. Otherwise, a new set of Vinegar variables should be chosen. This procedure for each successive layer is repeated until the last layer. In this step, we obtain a vector $\overline{X} = (\overline{x_1}, ..., \overline{x_n})$. The computation of this part consists of multivariate polynomial evaluation and solving system of linear equations.

Finally, we compute $X' = {L_2}^{-1}(\overline{X}) = ({x_1}', ..., {x_n}')$. Then X' is the signature for messages Y.

It can be observed that in Rainbow signature generation, two affine transformations are computed by invoking vector addition and matrix-vector multiplication, multivariate polynomials are required to be evaluated, and system of linear equations are required to be solved.

Signature Verification. To verify the authenticity of a signature X', $\overline{F}(X') = Y'$ is computed. If $Y' = Y$ holds, the signature is accepted, otherwise rejected. In this paper, we only work on the signature generation not signature verification.

Parameters of Rainbow Signature. We adopt the parameters of Rainbow signature suggested in [14] for practical applications to design our hardware, which is also implemented in [9]. This is a two-layer scheme which has a security level above 2^{80}. There are 17 random-chosen Vinegar variables and 12 Oil variables in the first layer, and 1 random-chosen Vinegar variables and 12 Oil variables in the second layer. The parameters are shown in Table 1.

Table 1. Parameters of Rainbow in Proposed Hardware Design

Parameter	Rainbow
Ground field size	$GF(2^8)$
Message size	24 bytes
Signature size	42 bytes
Number of layers	2
Set of variables in each layer	(17, 12), (1, 12)

3 Proposed Hardware Design for Rainbow Signature

3.1 Overview of the Hardware Design

The flowchart to generate Rainbow signature is illustrated in Fig. 1. It can be observed that Rainbow signature generation consists of computing affine transformations, polynomial evaluations and solutions for system of linear equations.

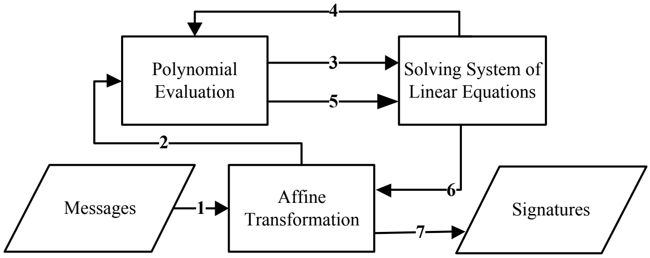

Fig. 1. The Flowchart to Generate Rainbow Signature

3.2 Choice of Irreducible Polynomial for the Finite Field

The choice of the irreducible polynomial for the finite field k is a critical part of our hardware design, since it affects the efficiency of the operations over the finite field. The irreducible polynomials for $GF(2^8)$ over $GF(2)$ can be expressed as 9-bit binary digits with the form $x^8+x^k+...+1$, where $0 < k < 8$ and the first bit and the last bit are valued one. There are totally 16 candidates. We evaluate the performance of the multiplications based on these irreducible polynomials respectively.

By comparing the efficiency of signature generations basing on different irreducible polynomials, $x^8+x^6+x^3+x^2+1$ is finally chosen as the irreducible polynomial in our hardware design.

3.3 Efficient Design of Multiplication of Three Elements

In Rainbow signature generation, we notice that there exist not only multiplication of two elements but also multiplication of three elements. An optimized design of the multiplier can dramatically improve the overall hardware execution efficiency.

Therefore, we design new implementation to speed up multiplication of three elements based on the multiplication of two elements [10]. The new design is based on a new observation that, in multiplication of three elements over $GF(2^8)$, it is much faster to multiply everything first than perform modular operation than the other way around. This is quite anti-intuitive and it works only over small fields. This idea, in general, is not applicable for large fields.

Suppose $a(x) = \sum_{i=0}^{7} a_i x^i$, $b(x) = \sum_{i=0}^{7} b_i x^i$ and $c(x) = \sum_{i=0}^{7} c_i x^i$ are three elements in $GF(2^8) = GF(2)[x]/f(x)$, and

$$d(x) = a(x) \times b(x) \times c(x)(\mathrm{mod}(f(x))) = \sum_{i=0}^{7} d_i x^i \tag{2}$$

is the expected multiplication result, where $f(x)$ is the irreducible polynomial.

First, we compute v_{ij} for $i = 0, 1, ..., 21$ and $j = 0, 1, ..., 7$ according to x^i mod $f(x) = \sum_{j=0}^{7} v_{ij}x^j$. Next, we compute S_i for $i = 0, 1, ..., 21$ via $S_i = \sum_{j+k+l=i} a_j b_k c_l$. After that, we compute d_i for $i = 0, 1, ..., 7$ via $d_i = \sum_{j=0}^{21} v_{ji}S_j$. Finally, the multiplication result of $a(x) \times b(x) \times c(x)\ mod\ f(x)$ is $\sum_{i=0}^{7} d_i x^i$.

3.4 Efficient Design of Partial Multiplicative Inversion

The multiplicative inverse over finite fields is a crucial but time-consuming operation in multivariate signature. An optimized design of the inverter can really help to improve the overall performance. Since multiplicative inversion is only used in solving system of linear equations, we do not implement a fully multiplicative inverter but adopt a partial inverter based on Fermat's theorem in our design.

Suppose $f(x)$ is the irreducible polynomial and β is an element over $GF(2^8)$, where $\beta = \beta_7 x^7 + \beta_6 x^6 + \beta_5 x^5 + \beta_4 x^4 + \beta_3 x^3 + \beta_2 x^2 + \beta_1 x + \beta_0$. According to the Fermat's theorem, we have $\beta^{2^8} = \beta$, and $\beta^{-1} = \beta^{2^8-2} = \beta^{254}$. Since $2^8 - 2 = 2 + 2^2 + 2^3 + 2^4 + 2^5 + 2^6 + 2^7$, then $\beta^{-1} = \beta^2\beta^4\beta^8\beta^{16}\beta^{32}\beta^{64}\beta^{128}$.

We can then construct the logic expressions of these items.

$$\begin{aligned} \beta^{2^i} = \beta_7 x^{2^i \times 7} + \beta_6 x^{2^i \times 6} + \beta_5 x^{2^i \times 5} + \beta_4 x^{2^i \times 4} + \\ \beta_3 x^{2^i \times 3} + \beta_2 x^{2^i \times 2} + \beta_1 x^{2^i} + \beta_0, \end{aligned} \tag{3}$$

The computation of $x^{2^i \times j}$ should be reduction modulo the irreducible polynomial, where $i = 1, 2, ..., 7$ and $j = 0, 1, ..., 7$, then β^{2^i} is transformed into the equivalent form. For instance, $\beta^{2^i} = \beta'_7 x^7 + \beta'_6 x^6 + \beta'_5 x^5 + \beta'_4 x^4 + \beta'_3 x^3 + \beta'_2 x^2 + \beta'_1 x + \beta'_0$.

We adopt the three-input multiplier described in Section 3.3 to design the partial inverter, where $ThreeMult(v1, v2, v3)$ stands for multiplication of three elements and $v1$, $v2$, $v3$ are operands and S_1, S_2 are the multiplication results.

$$\begin{aligned} S_1 &= ThreeMult(\beta^2, \beta^4, \beta^8), \\ S_2 &= ThreeMult(\beta^{16}, \beta^{32}, \beta^{64}). \end{aligned} \tag{4}$$

We call the triple (S_1, S_2, β^{128}) the partial multiplicative inversion of β. Below we will present how we adopt partial inversion in solving system of linear equations.

3.5 Optimized Gauss-Jordan Elimination

We propose a parallel variant of Gauss-Jordan elimination for solving a system of linear equations with the matrix size 12×12. The optimization and parallelization of Gauss-Jordan elimination can enhance the overall performance of solving system of linear equations.

Algorithm and Architecture. We give a straightforward description of the proposed algorithm of the parallel variant of Gauss-Jordan elimination in Algorithm 1, where *operation*(i) stands for operation performed in the i-th iteration, and $i = 0, 1, ..., 11$. The optimized Gauss-Jordan elimination with 12 iterations consists of pivoting, partial multiplicative inversion, normalization and elimination in each iteration.

We enhance the algorithm in four directions. First, multiplication of three elements is computed by invoking three-input multipliers designed in Section 3.3. Second, we adopt a partial multiplicative inverter described in Section 3.4 in our design. Third, the partial multiplicative inversion, normalization and elimination are designed to perform simultaneously. Fourth, during the elimination in the i-th iteration, we simultaneously choose the right pivot for the next iteration, namely if element $a_{i+1,i+1}$ of the next iteration is zero, we swap the $(i+1)$-th row with another j-th row with the nonzero element a_{ji}, where $i, j = 0, 1, ..., 11$. The difference from usual Gauss-Jordan elimination is that the usual Gauss-Jordan elimination choose the pivot after the elimination, while we perform the pivoting during the elimination. In other words, at the end of each iteration, by judging the computational results in this iteration, we can decide the right pivoting for the next iteration. By integrating these optimizations, it takes only one clock cycle to perform one iteration.

Algorithm 1. Solving a system of linear equations $Ax = b$ with 12 iterations, where A is a 12×12 matrix

```
var
    i: Integer;
begin
    i := 0;
    Pivoting(i = 0);
    repeat
        Partial_inversion(i), Normalization(i), Elimination(i);
        Pivoting(i+1);
        i:= i+1;
    until i = 12
end.
```

The proposed architecture is depicted in Fig. 2 with matrix size 12×12, where a_{ij} is the element located at the i-th row and j-th column of the matrix.

There exist three kinds of cells in the architecture, namely I, N_l, and E_{kl}, where $k = 1, 2, ..., 11$ and $l = 1, 2, ..., 12$. The I cell is for partial multiplicative inversion. As described in 3.4, two three-input multipliers are included in the I cell for computed partial multiplicative inversion. The N_l cells are for normalization. And the E_{kl} cells are for elimination. The architecture consists of one I cell, 12 N_l cells and 132 E_{lk} cells.

The matrixes depicted in Fig. 2 are used only to illustrate how the matrix changes. The left-most matrix is the one in the first clock cycle while the i-th matrix is the one in the i-th clock cycle. In the first clock cycle, the left-most matrix is sent to the architecture. a_{00} is sent to I cell for partial multiplicative inversion. The first row is sent to N_l for normalization. And the other rows except the first row are sent to E_{lk} for elimination. In this clock cycle, one iteration of Gauss-Jordan elimination is performed and the matrix has been updated. In the following clock cycles, the pivot element is sent to I cell for partial multiplicative inversion. The pivot row is sent to N_l for normalization. And the other rows except the pivot row are sent to E_{lk} for elimination. It can be observed that the system of linear equations with matrix size 12×12 can be solved with 12 clock cycles.

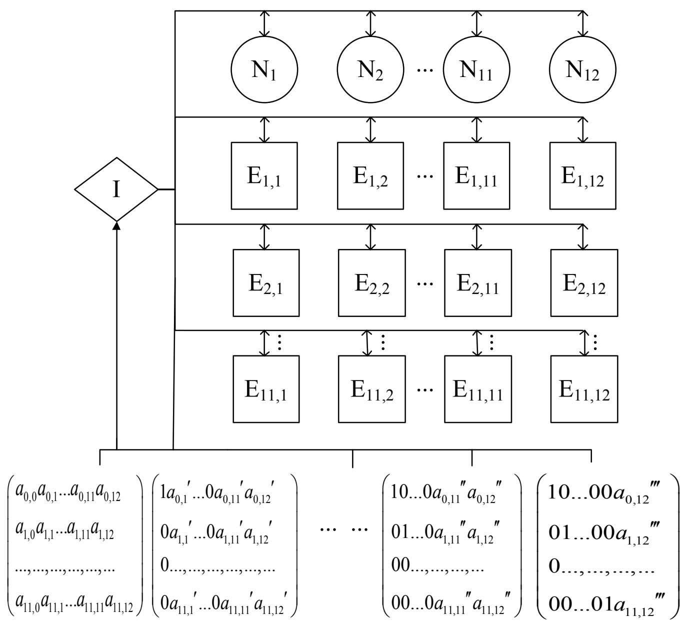

Fig. 2. Proposed Architecture for Parallel Solving System of Linear Equations with Matrix Size 12×12

Pivoting Operation. If the pivot a_{ii} of the i-th iteration is zero, we should find a nonzero element a_{ji} in the pivot column, i.e, the i-th column, as the new pivot element, where $i, j = 0, 1, ..., 11$. Then the computational results of the j-th row is sent to the N_l cells for normalization as the new pivot row. At the same time, the computational results of the i-th row is sent to the E_{jl} cells for elimination. In this way, we can ensure that the pivot element is nonzero in a new iteration. Therefore, the I cell, the N_l cells and the E_{kl} cells can execute simultaneously.

An example of pivoting is shown in Fig. 3. Before the second iteration, the second row is the pivot row but the pivot element is zero. The fourth row can be chosen as the new pivot row since a_{31} is nonzero. Then a_{31} is sent to I

cell for partial multiplicative inversion. The fourth row is sent to N_l cells for normalization, and then the other rows including the second row are sent to E_{1l} cells for elimination. Therefore, the computation of one iteration can be performed with one clock cycle.

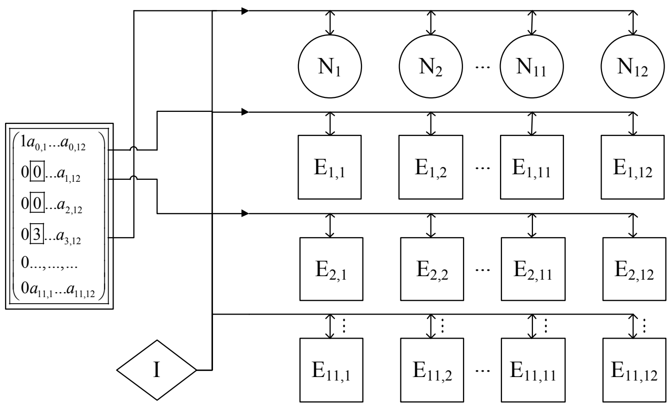

Fig. 3. Pivoting in Solving System of Linear Equations

Normalizing Operation. The normalizing operation invokes multiplicative inversions and multiplications, then we can enhance the implementation in two aspects.

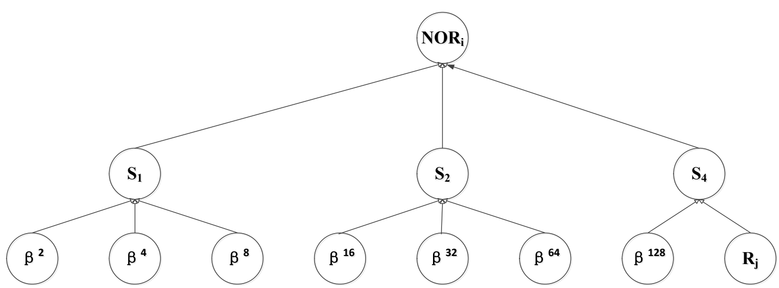

Fig. 4. Optimized Normalization in Solving System of Linear Equations

First, the multiplicative inverse β^{-1} over $GF(2^8)$ is optimized to the multiplication of 7 elements due to $\beta^{-1} = \beta^2\beta^4\beta^8\beta^{16}\beta^{32}\beta^{64}\beta^{128}$, as mentioned in Section 3.4.

Second, a new multiplier is designed to speed up the multiplication of three elements that denoted by $ThreeMult(v1, v2, v3)$, where $v1$, $v2$ and $v3$ are operands, while the multiplication of two elements is defined by $TwoMult(v1, v2)$.

The schematic diagram of normalization is shown in Fig. 4, where R_i for the i-th element in the pivot row, and NOR_i for the normalizing result, respectively. Then, we have the expressions

$$\begin{aligned}
S_1 &= ThreeMult(\beta^2, \beta^4, \beta^8), \\
S_2 &= ThreeMult(\beta^{16}, \beta^{32}, \beta^{64}), \\
S_4 &= TwoMult(\beta^{128}, R_i), \\
NOR_i &= ThreeMult(S_1, S_2, S_4).
\end{aligned} \tag{5}$$

S_1 and S_2 are executed in I cell for partial multiplicative inversion while S_4 and NOR_i are executed in N_i cells for normalization. Thus one two-input multiplier as well as another three-input multiplier are included in N_i cells. Since S_1, S_2 and S_4 can be implemented in parallel in each iteration, the critical path of normalizing consists of only two multiplications of three elements.

Eliminating Operation. The schematic diagram of normalization is shown in Fig. 5, where R_j stands for the j-th element in the pivot row, C_i for the i-th element in the pivot column, and ELI_{ij} is the eliminated result of a_{ij}.

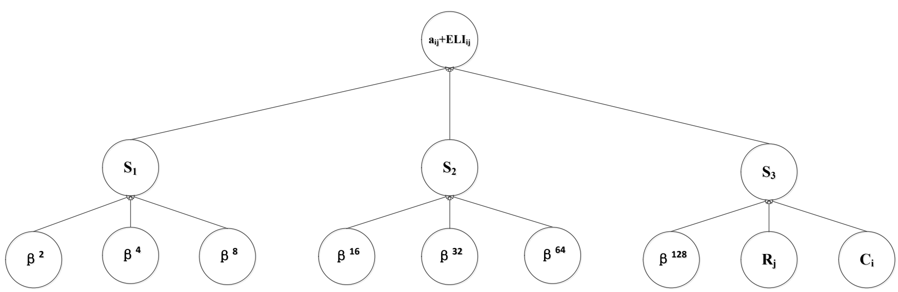

Fig. 5. Optimized Elimination in Solving System of Linear Equations

Then, we have the expressions

$$\begin{aligned}
S_1 &= ThreeMult(\beta^2, \beta^4, \beta^8), \\
S_2 &= ThreeMult(\beta^{16}, \beta^{32}, \beta^{64}), \\
S_3 &= ThreeMult(\beta^{128}, R_j, C_i), \\
ELI_{ij} &= a_{ij} + ThreeMult(S_1, S_2, S_3).
\end{aligned} \tag{6}$$

S_1 and S_2 are executed in I cell for partial multiplicative inversion while S_3 and ELI_{ij} are executed in E_{ij} cells for elimination. Thus two three-input multipliers and one adder are included in E_{ij} cells. Since S_1, S_2 and S_3 can be implemented in parallel in each iteration, the critical path of elimination consists of only two multiplications of three elements and one addition.

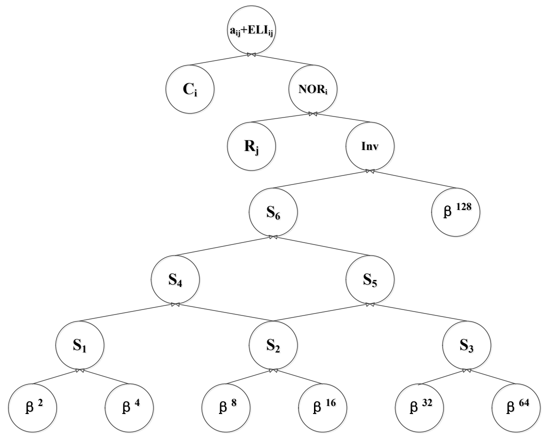

Fig. 6. Original Design of Gauss-Jordan Elimination

Overall Optimization. By integrating the optimizations above, Fig. 7 shows that the critical path of our design is reduced from five multiplications and one addition to two multiplications and one addition, compared with the original principle of Gauss-Jordan elimination illustrated in Fig. 6.

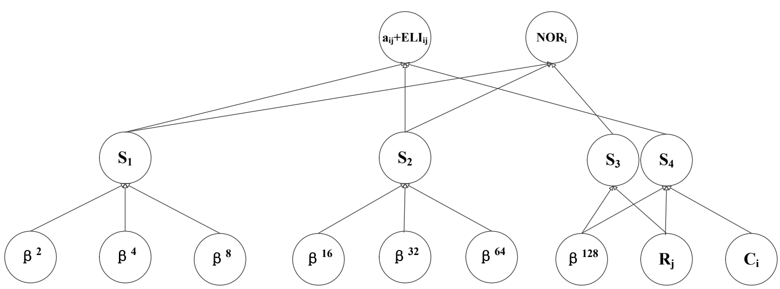

Fig. 7. Optimized Design of Gauss-Jordan Elimination

Therefore, our design takes one clock cycle to perform the operations in each iteration of solving system of linear equations. In the end, it takes only 12 clock cycles to solve a system of linear equations where the matrix size is 12×12.

3.6 Designs of Affine Transformations and Polynomial Evaluations

${L_1}^{-1}$: $k^{24} \to k^{24}$ and ${L_2}^{-1}$: $k^{42} \to k^{42}$ affine transformations are computed by invoking vector addition and vector-multiplication over a finite field. Two-layer Oil-Vinegar constructions including 24 multivariate polynomials are evaluated by invoking multiplication over a finite field. Thus multiplication over a finite field is

Table 2. Number of Multiplications in ${L_1}^{-1}$, ${L_2}^{-1}$ Affine Transformations and Polynomial Evaluations

Components	Number of multiplications
${L_1}^{-1}$ transformation	576
The first 12 polynomial evaluations	6324
The second 12 polynomial evaluations	15840
${L_2}^{-1}$ transformation	1764
Total	24504

the most time-consuming operation in these computations. Table 2 summarizes the numbers of multiplications in two affine transformations and polynomial evaluations. The number of multiplications of the components of polynomial evaluations is summarized in Table 3.

Table 3. Number of Multiplications in Components of Polynomial Evaluations

	The first layer	The second layer
V_iO_j	2448	4320
V_iV_j	3672	11160
V_i	204	360
Total	6324	15840

4 Implementations and Experimental Results

4.1 Overview of Our Implementation

Our design is programmed in VHDL and implemented on a EP2S130F1020I4 FPGA device, which is a member of ALTERA Stratix II family. Table 4 summarizes the performance of our implementation of Rainbow signature measured in clock cycles, which shows that our design takes only 198 clock cycles to generate a Rainbow signature. In other words, our implementation takes 3960 ns to generate a Rainbow signature with the frequency of 50 MHz. All the experimental results mentioned in this section are extracted after place and route.

Table 4. Running Time of Our Implementation in Clock Cycles

Steps	Components	Clock cycles
1	${L_1}^{-1}$ transformation	5
2	The first 12 polynomial evaluations	45
3	The first round of solving system of linear equations	12
4	The second 12 polynomial evaluations	111
5	The second round of solving system of linear equations	12
6	${L_2}^{-1}$ transformation	13
	Total	198

4.2 Implementation of Multiplier, Partial Inverter and LSEs Solver

Our multipliers and partial inverter can execute a multiplication and partial multiplicative inversion over $GF(2^8)$ within one clock cycle respectively. As mentioned in Section 3.5, the critical path of each iteration of optimized Gauss-Jordan elimination includes two multiplications and one addition. Since there exist some overlaps in two serial multiplications, one iteration of optimized Gauss-Jordan elimination can be computed in 20 ns with one clock cycle. Therefore, it takes 12 clock cycles to solve a system of linear equations of matrix size 12×12, which is 240 ns with a frequency of 50 MHz.

Table 5. FPGA Implementations of the Multiplier, Partial Inverter and Optimized Gauss-Jordan Elimination over $GF(2^8)$

Components	Multiplier	Partial inverter	Gauss-Jordan elimination
Combinational $ALUTs$	37	22	21718
Dedicated logic registers	0	0	1644
Clock cycles	1	1	12
Running time (ns)	10.768	9.701	240

Table 5 is extracted after place and route of multiplication, partial multiplicative inversion and optimized Gauss-Jordan elimination over $GF(2^8)$. Three different kinds of cells included in our proposed architecture have been described and their resource consumptions are given in Table 6.

Table 6. The Resource Consumptions for Each Cell in the Proposed Architecture for Solving System of Linear Equations

Cell	Use	Two-input multiplier	Three-input multiplier	Adder
I cell	Partial inversion	0	2	0
N cell	Normalization	1	1	0
E cell	Elimination	0	2	1

4.3 Implementation of Transformations and Polynomial Evaluations

The affine transformations ${L_1}^{-1}$ and ${L_2}^{-1}$ invoke vector addition and matrix-vector multiplication over $GF(2^8)$. Table 7 shows that two affine transformations take 18 clock cycles, which is 360 ns with a frequency of 50 MHz, where the second and fourth columns are the performance of vector additions using L_1 offset and L_2 offset respectively and the third and fifth columns are the performance of matrix-vector multiplications using the matrixes of ${L_1}^{-1}$ and ${L_2}^{-1}$ respectively.

Table 8 illustrates that polynomial evaluations takes 156 clock cycles, which is 3120 ns with a frequency of 50 MHz, where the second, third and fourth columns are the performances of components of multivariate polynomials, respectively.

Table 7. Clock Cycles and Running Time of Two Affine Transformations

Components	L_1 offset	${L_1}^{-1}$	L_2 offset	${L_2}^{-1}$	Total
Clock cycles	1	4	1	12	18
Running time (ns)	20	80	20	240	360

Table 8. Clock Cycles and Running Time of Polynomial Evaluations

Components	V_iO_j	V_iV_j	V_i	Total cycles	Total time
The first layer	17	26	2	45	900 ns
The second layer	30	78	3	111	2220 ns

Note here that our implementation focuses solely on speeding up the signing process, and, in terms of area, we compute the size in gate equivalents (GEs), about 150,000 GEs, which is 2-3 times the area of [8].

5 Comparison with Related Works

We compare the implementations of solving system of linear equations and Rainbow signature generation with related works by the following tables, which clearly demonstrate the improvements of our new implementation.

Table 9. Comparison of Solving System of Linear Equations with Matrix Size 12×12

Scheme	Clock cycles
Original Gauss-Jordan elimination	1116
Original Gaussian elimination	830
Wang-Lin's Gauss-Jordan elimination [12]	48
B. Hochet's Gaussian elimination [13]	47
A Bogdanov's Gaussian elimination [11]	24
Implementation in this paper	12

Table 10. Performance Comparison of Signature Schemes

Scheme	Clock cycles
en-TTS [5]	16000
Rainbow (42,24) [9]	3150
Long-message UOV [9]	2260
Rainbow [8]	804
Short-message UOV [9]	630
This paper	198

6 Conclusions

We propose a new optimized hardware implementation of Rainbow signature scheme, which can generate a Rainbow signature with only 198 clock cycles, a new record in generating digital signatures.

Our main contributions include three parts. First, we develop a new parallel hardware design for the Gauss-Jordan elimination, and solve a 12×12 system of linear equations with only 12 clock cycles. Second, a novel multiplier is designed to speed up multiplication of three elements over finite fields. Third, we design a novel partial multiplicative inverter to speed up the multiplicative inversion of finite field elements. Through further other minor optimizations of the parallelization process and by integrating the major optimizations above, we build a new hardware implementation, which takes only 198 clock cycles to generate a Rainbow signature, four times faster than the 804-clock-cycle Balasubramanian-Bogdanov-Carter-Ding-Rupp design [8] with similar parameters. Our implementation focuses solely on speeding up the signing process not area utilization.

The optimization method of three-operand multiplier, partial multiplicative inverter, and LSEs solver proposed can be further applied to various applications like matrix factorization, matrix inversion, and other multivariate PKCs.

Acknowledgement. This work is supported by National Natural Science Foundation of China under Grant No. 61170080 and 60973131, and supported by Guangdong Province Universities and Colleges Pearl River Scholar Funded Scheme (2011). This paper is also supported by the Fundamental Research Funds for the Central Universities of China under Grant No.2009ZZ0035 and No.2011ZG0015.

References

1. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Review 41(2), 303–332 (1999)
2. Ding, J., Schmidt, D.: Multivariate public key cryptosystems. In: Advances in Information Security, vol. 25, Springer, Heidelberg (2006)
3. Ding, J., Schmidt, D.: Rainbow, a New Multivariable Polynomial Signature Scheme. In: Ioannidis, J., Keromytis, A.D., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 164–175. Springer, Heidelberg (2005)
4. Yang, B.-Y., Chen, J.-M., Chen, Y.-H.: TTS: High-Speed Signatures on a Low-Cost Smart Card. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 371–385. Springer, Heidelberg (2004)
5. Yang, B.-Y., Cheng, C.-M., Chen, B.-R., Chen, J.-M.: Implementing Minimized Multivariate PKC on Low-Resource Embedded Systems. In: Clark, J.A., Paige, R.F., Polack, F.A.C., Brooke, P.J. (eds.) SPC 2006. LNCS, vol. 3934, pp. 73–88. Springer, Heidelberg (2006)
6. Chen, A.I.-T., Chen, C.-H.O., Chen, M.-S., Cheng, C.-M., Yang, B.-Y.: Practical-Sized Instances of Multivariate PKCs: Rainbow, TTS, and ℓIC-derivatives. In: Buchmann, J., Ding, J. (eds.) PQCrypto 2008. LNCS, vol. 5299, pp. 95–108. Springer, Heidelberg (2008)

7. Chen, A.I.-T., Chen, M.-S., Chen, T.-R., Cheng, C.-M., Ding, J., Kuo, E.L.-H., Lee, F.Y.-S., Yang, B.-Y.: SSE Implementation of Multivariate PKCs on Modern X86 CPUs. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 33–48. Springer, Heidelberg (2009)
8. Balasubramanian, S., Bogdanov, A., Rupp, A., Ding, J., Carter, H.W.: Fast multivariate signature generation in hardware: The case of Rainbow. In: 16th International Symposium on Field-Programmable Custom Computing Machines, pp. 281–282 (April 2008)
9. Bogdanov, A., Eisenbarth, T., Rupp, A., Wolf, C.: Time-Area Optimized Public-Key Engines: $\mathcal{MQ}$-Cryptosystems as Replacement for Elliptic Curves? In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 45–61. Springer, Heidelberg (2008)
10. Mastrovito, E.: VLSI Designs for Multiplication over Finite Fields $GF(2^m)$. In: Huguet, L., Poli, A. (eds.) AAECC 1987. LNCS, vol. 356, Springer, Heidelberg (1989)
11. Bogdanov, A., Mertens, M.C., Paar, C., Pelzl, J., Rupp, A.: A parallel hardware architecture for fast Gaussian elimination over $GF(2)$. In: 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, pp. 237–248. IEEE (2006)
12. Wang, C.-L., Lin, J.-L.: A systolic architecture for computing inverses and divisions in finite fields $GF(2^m)$. IEEE Transactions on Computers 42(9), 1141–1146 (1993)
13. Hochet, B., Quinton, P., Robert, Y.: Systolic Gaussian elimination over $GF(p)$ with partial pivoting. IEEE Transactions on Computers 38(9), 1321–1324 (1989)
14. Ding, J., Yang, B.-Y., Chen, C.-H.O., Chen, M.-S., Cheng, C.-M.: New Differential-Algebraic Attacks and Reparametrization of Rainbow. In: Bellovin, S.M., Gennaro, R., Keromytis, A.D., Yung, M. (eds.) ACNS 2008. LNCS, vol. 5037, pp. 242–257. Springer, Heidelberg (2008)