Hardware Implementation and Study of Inverse Algorithm in Finite Fields

Bao kejin,and Song yonggang

School of Computer Science and Telecommunication Engineering, JU, Zhenjiang, 212013, China

Summary
Inverse calculation in finite fields is the base to implement Hyperelliptic Curve Cryptography (HECC) and HECC implementation is the key to fast implement calculating inverse in finite fields. In this paper, fast algorithm of calculating inverse in finite fields and its method to implement with hardware in HECC are discussed, the EEA algorithm and the MIMA algorithm, which are currently in common use, are compared and an improved MIMA algorithm is brought forward. In the algorithm, 2 bits parallel scheme is used and shift register only takes into account of two situations including 2 bits shift and 1 bit shift. The simplest (two situations) barrel shifter can be designed when realizing this algorithm with FPGA and each of shifts is completed in one cycle. Algorithm description carries out function simulation and timing simulation in QuartusII environment, which is improved in both speed and area compared with past algorithms.

Key words:
HECC, FPGA, Inverse calculation, Fast algorithm.

1. Introduction

Hyperelliptic Curve Cryptography (HECC) is a kind of cryptography more difficult to be resolved than Elliptic Curve Cryptography (ECC). At present, the HECC theory has already basically been mature and the study on HECC at home and abroad mainly focuses on how to implement [1][2][3]. Due to the high encryption density and large computation complexity of HECC, HECC implementation has important theoretical significance and higher value in use not only to strengthen the safety of information system but also to study higher-intensity encryption system.

Inverse calculation in finite fields is the base to implement ECC and HECC and fast implementation of calculating inverse in finite fields is the key to implement ECC and HECC. Since HECC has higher encryption density than ECC, it has a series of advantages such as smaller bandwidth can be used and calculation can be done in smaller field. In recent years, there have been large amount of studies focusing on HECC implementation, but it has brought forward much higher requirements for calculating inverse in finite fields to implement HECC. Furthermore, since the hardware encryption system is required to be reconfigured, it has become one of the hot spots of the current study to implement HECC on the base of FPGA.

2. Calculating inverse in finite fields

There have been some dissertations [4][5][6] discussing in detail the fast algorithm of calculating inverse in finite fields in HECC. There are three algorithms for implementation: the first is to use the repeated square-and-multiply algorithm of Fermat's theorem; the second is the expanded Euclidean Algorithm (EEA) and the third is Modified Almost Inverse Algorithm (MAIA). The first algorithm has not already been adopted due to larger calculation amount. The later two algorithms are the algorithms in common use at present, however, it still takes much time to implement inverse calculation with these two algorithms, therefore, it is one key point of the study in this paper to improve the speed of inverse calculation.

2.1 EEA Algorithm

EEA Algorithm is to calculate the inverse in finite fields through repeated iteration with basic ideas as follows: a and $f(x)$ are separately multiplied by or divided by x repeatedly and are added together, at the same time, 1 and 0 is made same inversion. So, when a becomes 1, 1 will become the inverse of a. EEA Algorithm is shown as follows:

Input: $a \in GF(2^n)$

Output: $b = a^{-1} \in GF(2^n)$

1. $b \leftarrow 1, c \leftarrow 0, u \leftarrow a, v \leftarrow f$;
2. While $\deg(u) \neq 0$
2.1. $j \leftarrow \deg(u) - \deg(v)$;
2.2. if $j < 0$ then
$u \leftarrow u + (v << j),\ b \leftarrow (c << j)$;
2.3. $u \leftarrow u + (v << j),\ b \leftarrow (c << j)$;
3. return b ;

The hardware structure to implement EEA algorithm is shown as in Fig.1.

Manuscript received September 5, 2006.
Manuscript revised September 25 , 2006.

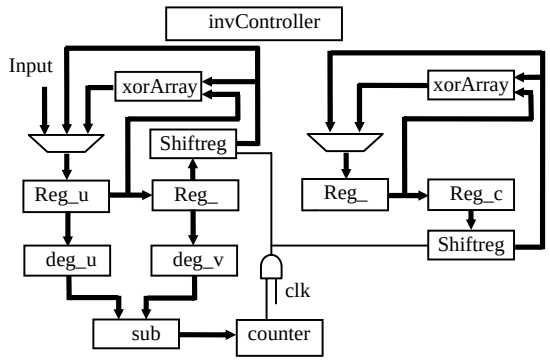

Fig.1 Hardware implementation structure of EEA Algorithm

This algorithm needs a shifter of j bit ($0 \le j \le 83$). In this paper, the method to control shift times of shifter via counter is adopted. In Fig.1, deg_u and deg_v are the modules of the highest of the times to evaluate Reg_u and Reg_v. The difference of deg_u and deg_v is stored in register of the counter and the shift times of Shiftreg are controlled by counter. The implementation of register is basically same, each of which is orginary register or shift register. However, every register has different initial value for resetting. The registers are all zero when Reg_u and Reg_c reset, the value is reduction polynomial when Reg_v resets and the value is "1" when Reg_b resets.

The deg_u and deg_v adopt same designs, whose inputs are 83-bit data and outputs are 8-bit data (representing deg_u) and output degu_zero indicating whether deg_u is 0, which is a combinational circuit. deg_u design needs a *conv_std_logic_vecotor()* function, which converts the integer data into *std_logic_vecotor* type. *For…loop* statement is used in the process to count the bits of the input data which are '0' and make conversion by using the abovementioned function with part of procedure as follows:

```
a_temp<="00000000";
for j in 83 downto 0 loop
   if i(j)='1' then
      a_temp<=conv_std_logic_vecotor(j,8);
      exit;
   end if;
end loop;
a<=a_temp;
```

In similar manner, synthesis tool also synthesizes this section of procedure into a combinational circuit, which in fact is a priority encoder to encode 83-bit input into 8-bit output.

Both the subtracter and the counter are simpler and easy to implement, so it will not be mentioned any more here. The controller is more complex, which has 6 inputs, 9 outputs and total 16 pieces of signal line. Fig.2 is the module chart of the controller. Besides the signals such as *clk*, *reset* and *start*, the controller also includes *degu_zero* signal indicating the highest of the times of register u, *sub82_co* signal indicating the result of subtracter as negative and the *count8_finish* signal indicating the self subtraction of the counter as 0. The output includes *reset_all* signal controlling the reset of all registers, *mux3in83_u_sel[1..0]* signal to select multiplexer, *reg83_u_oe* signal for reg_u loading, *sreg83_load* signal for Shiftreg loading, *sub82_en* signal to enable subtracter, *count8_load* signal for original data loading of the counter as well as *reg83_v_oe* and *reg83_b_oe* signals for deg_v and deg_b loading. Due to the simple implementation, the state table of the controller will not be listed here.

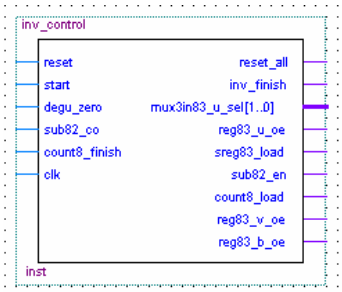

Fig.2 Module chart of module controller of EEA calculating inverse

It can be seen that the shortcoming of this implementation is that it will need j cycles to shift j bits each time, which will greatly prolong the whole period of calculating inverse. The implementation of this algorithm occupies 999 logic units, needs about 300 cycles and the frequency can reach 100M.

If barrel shifter is adopted, the average execution cycle can reach $(83/2)\times 6+5 \approx 257$, however, the hardware cost will be too great to do so. For the inverse calculator implemented in this paper, the value of the counter can be between 0 and 82 and it is shown by experiments that 32-bit barrel shifter has already required occupying more than 200 units, so this scheme can not be adopted in this paper to improve the speed of calculating inverse.

2.2. MIMA Algorithm

As same as EEA Algorithm, MIMA Algorithm is also to calculate the inverse in finite fields via repeated iteration. What different is that MIMA Algorithm adopts the method of little endian. Speaking only from the algorithm effiency, two algorithms are same, because it can averagely eliminate two terms by each iteration. However, judging from the hardware implementation of algorithm, since each iteration of MIMA Algorithm can only shift one bit, the ordinary shifter can satisfy the requirement and the counter will not be needed for control.

Inverse algorithm in MAIA is shown as follows:

Input: $a \in GF(2^n)$

Output: $b = a^{-1} \in GF(2^n)$

1. $b \leftarrow 1, c \leftarrow 0, u \leftarrow a, v \leftarrow f$;
2. While x divides u do:

2.1 $u \leftarrow u / x$;

2.2 if x divides b then $b \leftarrow b / x$;

else $b \leftarrow (b + f) / x$;

3. If $u = 1$ then return(b) ;
4. If $\deg(u) < \deg(v)$ then: $u \leftrightarrow v, b \leftrightarrow c$;
5. $u \leftarrow u + v, b \leftarrow b + c$;
6. Goto step2 ;

The hardware structure for implementation of MAIA Algorithm is shown as in Fig.3.

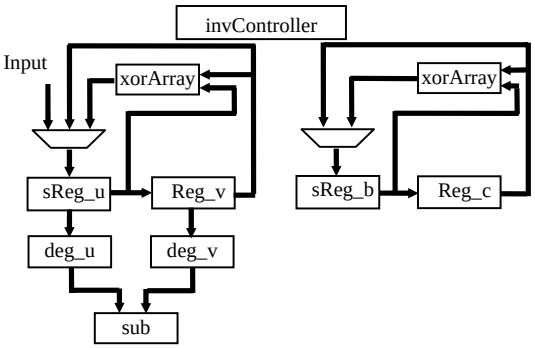

Fig.3 Hardware implementation structure of MIMA Algorithm

It can be seen from Fig.3 that compared with the first algorithm, this structure integrates the shift register shiftreg_u with register and shift register shiftreg_b with register reg_b to compose sReg_u register and sReg_b register. Though sReg_u is very simple, sReg_b register in fact is very complex. It needs reduction operation according to whether b can be exactly divided by x in this register (adding control lines of sel[1..0] input from sReg_u). Furthermore, in order to receive the control of the controller, this module also has one more *en* signal input terminal which controls the shift operation of sReg_u. The key part of sReg_b is shown as follows:

```
case sel(0) is
 when '0' =>
    reg83_temp(83 downto 0):='0' & reg83(83 downto 1);
    reg83_temp(82):=reg83_temp(82) xor selb_temp(0);
    reg83_temp(6):=reg83_temp(6) xor selb_temp(0);
    reg83_temp(3):=reg83_temp(3) xor selb_temp(0);
    reg83_temp(1):=reg83_temp(1) xor selb_temp(0);
 when others =>
   reg83_temp(83 downto 0):=reg83(83 downto 0);
end case;
```

The module chart of the controller is shown as in Fig.4. For the definition of each control signal, please refer to the description in last section. Since the design of state machine is more similar with the controller of the improved algorithm to be mentioned in this paper, the state transition table of the controller will not be listed here temporarily.

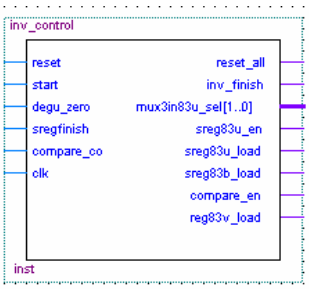

Fig.4 Module chart of module controller of MIMA calculating inverse

Stimulation graphics file is established after compiling, synthesis and adaptation with QuartusII. Firstly function simulation is carried out to verify the correctness of the design. Timing stimulation is carried out after successful verification. Stimulation diagram is not listed due to limited space.

Total logic elements are 650 and Total pins are 172 according to stimulation result, which shows that the implementation of MIMA Algorithm occupies 650 logic units. Judging from the structure, the hardware saving is mainly two register modules fewer, and furthermore the control circuit is also simplified a little. What the *clk* adopts in stimulaiton is 100M clock, so the highest frequency of this inverse calculator can reach 100M. Additionally, judging from algorithm, the average execution cycle of this inverse calculator is $(83/2) \times 6 + 5 \approx 257$ and the experimental result is in accordance with this data.

3. Comparison of EEA Algorithm and MIMA Algorithm

Each iteration of EEA Algorithm will carry out many times of shifts, but each iteration of MIMA Algorithm will only carry out one time of shift, then whether will the efficiency of whole calculating inverse greatly decrease? Why are the average execution cycles of the two algorithms same? The two algorithms are analyzed and compared in this paper.

3.1 EEA Algorithm analysis

For two polynomials:

$$u = x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 x + a_0 \quad (1)$$

$$v = x^m + b_{m-1}x^{m-1} + b_{m-2}x^{m-2} + \cdots + b_1 x + b_0 \quad (2)$$

If $m > n$, i.e. $\deg(u) < \deg(v)$, then

$$u \leftrightarrow v,\ u \leftarrow v << (m-n);$$

$$u = b'_{m-1}x^{m-1} + b'_{m-2}x^{m-2} + \cdots + b'_1 x + b'_0 \quad (3)$$

$$v = x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 x + a_0 \quad (4)$$

If $b_{m-1}\cdots b_{m-j+1} = 0,\ b_{m-j} \neq 0$, then if $(m-j) > n$, then $u \leftarrow v << (m-j-n)$.

It can be seen that u can eliminate at least one highest term every time, the term to eliminate 2 highest terms every time is $a_{n-1} = b_{m-1}$ and the term to eliminate 3 highest terms is $a_{n-1} = b_{m-1}$ and $a_{n-2} = b_{m-2}$. Since $a_{n-1}, a_{n-2}, b_{m-1}, b_{m-2} \in \{0,1\}$, probability $p(a_{n-1} = b_{m-1}) = \frac{1}{2}$, i.e. the probability to eliminate two highest terms every time is $\frac{1}{2}$. And since $p((a_{n-1} = b_{m-1}) \cap (a_{n-2} = b_{m-2})) = \frac{1}{2} \times \frac{1}{2} = \frac{1}{4}$, the probability to eliminate 3 highest terms every time is $\frac{1}{4}$ and by analogy. Taking $GF(2^{83})$ as example, it is assumed that the number of the highest terms to be eliminated by each iteration is c and probability is $p(c)$, its distribution rate is shown as in Table 1:

Table 1 Probability distribution table of $p(c)$

c	1	2	3	4	….	81	82	83
$p(c)$	1	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	….	$\frac{1}{2^{80}}$	$\frac{1}{2^{81}}$	$\frac{1}{2^{82}}$

The mean of c is:

$$Ec = 1\times 1 + 2\times\frac{1}{2} + 3\times\frac{1}{4} + \cdots + 81\times\frac{1}{2^{80}} + 82\times\frac{1}{2^{81}} + 83\times\frac{1}{2^{82}} \quad (5)$$

$$\frac{1}{2}\times Ec = 1\times\frac{1}{2} + 2\times\frac{1}{4} + 3\times\frac{1}{8} + \cdots + 81\times\frac{1}{2^{81}} + 82\times\frac{1}{2^{82}} + 83\times\frac{1}{2^{83}} \quad (6)$$

(5)−(6) can obtain:

$$\frac{1}{2}\times Ec = 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots + \frac{1}{2^{81}} + \frac{1}{2^{82}} - 83\times\frac{1}{2^{83}}$$

$$= 2 - \frac{1}{2^{82}} - 83\times\frac{1}{2^{83}}$$

$$\approx 2$$

So $Ec \approx 4$

Since u and v will exchange in the iteration process, i.e. the times of u and v will decrease, the number of highest terms to be eliminated averagely by each iteration is $1/2$ of Ec, i.e. 2. The average times of iteration needed for completion of inverse calculation is $83/2 = 42$ times.

3.2 MAIA Algorithm analysis

For two polynomials:

$$u = x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 x + a_0 \quad (7)$$

$$v = x^m + b_{m-1}x^{m-1} + b_{m-2}x^{m-2} + \cdots + b_1 x + b_0 \quad (8)$$

The number of the highest terms which can be eliminated when u shifts right every time (While statement) is at least 1 and the probability to eliminate 2 terms simultaneously is $\frac{1}{2}$. But taking into account of $u \leftrightarrow v$ and $u \leftarrow u + v$ operations, the number of terms to be eliminated every time is 2. The analysis method is as same as EEA. However, MAIA Algorithm can only shift one bit when shifting right every time and it needs to determine whether the lowest order is zero for whether to continue to shift right or not next time. If shift right can not be implemented, it shall determine the size of $\deg(u)$ and $\deg(v)$ as well as calculate $u = u + v$. Since the lowest order of v will certainly be 1, the value of $u = u + v$ can shift right at least one bit.

3.3 Comparison of two algorithms

In each time of iteration, both EEA Algorithm and MAIA Algorithm can at least eliminate one highest term and need to shift many bits. Neither of them needs complex multiplication or division, which can be easily implemented with hardware. It can be seen from the abovementioned analysis that 2 terms can be averagely eliminated when shifting j bits in EEA Algorithm. Though one term can be eliminated by every operation of shift right in MAIA Algorithm, the probability for continuous execution can also satisfy the probability distribution relation of EEA. Therefore, the number of terms which can be averagely eliminated by each iteration in two algorithms are equal, both of which are 2.

4. Improved MIMA Algorithm

As mentioned before, hyperelliptic curve encryption requires the sub-module to be with very fast speed and neither of the abovementioned two algorithms can meet this requirement. An improved scheme taken into account in this paper is to carry out part parallel processing of the abovementioned algorithms. Since MIMA Algorithm adopts little endian, the method of shift times can be

decided by firstly determining the low order. For EEA Algorithm, because it adopts big endian with unfixable shift times, it will not be convenient for prejudgment. The improved MIMA Algorithm adopts 2 bits-parallel sheme. Inverse algorithm in improved MAIA field:

Output: $a \in GF(2^n)$

1. $b \leftarrow 1, c \leftarrow 0, u \leftarrow a, v \leftarrow f, \deg\_v \leftarrow \deg(f)$,

   $b \leftarrow b >> 2, f_1 \leftarrow f >> 1, f_2 = f >> 2$;

2. case $u[1..0]$ is

2.1 when "00" =>

$u \leftarrow u >> 2$;

$b \leftarrow \underline{b >> 2} + \underline{f_2 \bullet b[0]} + \underline{\underline{f_1 \bullet (b[1] f[1] + b[1] b[0] + b[1] b[0] f[1])}}$;

goto 2;

2.2 when "10" =>

$u \leftarrow u >> 1$; $b \leftarrow b >> 1 + f_1 \bullet b[0]$;

2.3 when others =>;

end case;

3. if $u = 1$ then return b;

4. if $\deg(u) < \deg\_v$ then

$u \leftrightarrow v; b \leftrightarrow c; \deg\_u \leftrightarrow \deg\_v$;

5. $u \leftarrow u + v; b \leftarrow b + c$;

6.Goto step2 ;

The shift registers in above algorithms only need to take into account of two situation including 2 bits shift and 1 bit shift. The simplest (two situations) barrel shifter can be designed for implementation of this algorithm with FPGA. Barrel shifter selects the shift bits with $u[1], u[0]$ encoding and all shifts are completed in one cycle. Thus, it can save one cycle for the situation with larger probability of occurrence that two bits are 0 simultaneously. The hardware structure of algorithm implementation is shown as in Fig.5.

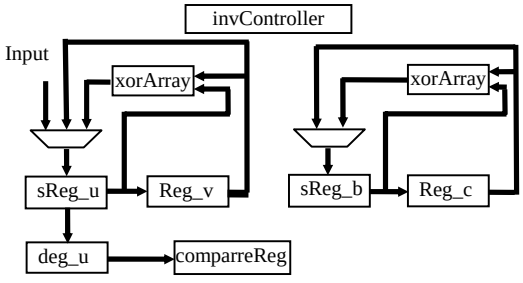

Fig.5 Hardware implementation structure of improved MIMA Algorithm

In Fig.5, both shift register u and shift register b are more complex than the aforementioned two structures, which is so called "trading area for time". The module charts of sReg_u and sReg_b are separately shown as in Fig.6 and Fig.7.

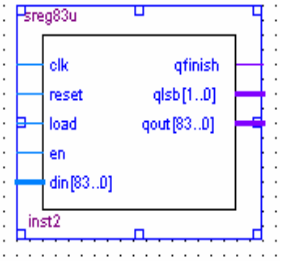

Fig.6 Module chart of sReg_u

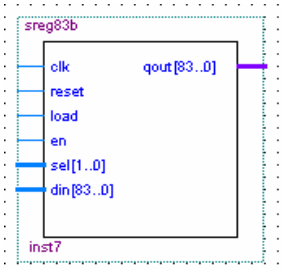

Fig.7 Module chart of sReg_b

In module of sReg_u, what *qlsb* outputs are the low two bits of register value and what *qout* outputs is the value of entire register. Compared with the abovementioned second implementation , if the low two bits reg83 (1 down to 0) determining register value in sReg_u is "00", then the register will shift right 2 bits, if it is "10", then shift right one bit and no shift for other situations. Therefore, the representation of sReg_u module shall be:

```
case reg83(1 down to 0) is
    when "00" =>
     reg83(83 down to 0)<="00" & reg83(83 down to 2);
    when "10" =>
      reg83(83 down to 0)<='0' & reg83(83 down to 1);
    when others =>
end case;
```

It needs to complete the reduction process after shift for implementation of sReg_b and the reduction is carried out according to the lowest two bits of the register. sReg_b is divided into three situations according to the output sel[1..0], which is same with the situation of sReg_u module. In fact, sel[1..0] here is the reg83(1 down to 0) of sReg_u module. The implementation procedure of sReg_b is shown as follows:

```
case sel is
when "00" =>
 reg83_temp(83 downto 0):="00" & reg83(83 downto 2);
  reg83_temp(81):=reg83_temp(81) xor selb_temp(0);
  reg83_temp(5):=reg83_temp(5) xor selb_temp(0);
  reg83_temp(2):=reg83_temp(2) xor selb_temp(0);
  reg83_temp(0):=reg83_temp(0) xor selb_temp(0);
  reg83_temp(82):=reg83_temp(82) xor selb_temp(1);
  reg83_temp(6):=reg83_temp(6) xor selb_temp(1);
  reg83_temp(3):=reg83_temp(3) xor selb_temp(1);
  reg83_temp(1):=reg83_temp(1) xor selb_temp(1);
when "10" =>
```

```
    reg83_temp(82 downto 0):='0' & reg83(82 downto 1);
    reg83_temp(82):=reg83_temp(82) xor selb_temp(0);
    reg83_temp(6):=reg83_temp(6) xor selb_temp(0);
    reg83_temp(3):=reg83_temp(3) xor selb_temp(0);
    reg83_temp(1):=reg83_temp(1) xor selb_temp(0);
when others =>
    reg83_temp(83 downto 0):=reg83(83 downto 0);
end case;
```

The controller totally has 6 input signals and 8 output signals, whose module chart is basically same with the aforementioned design. Stimulation graphics file is established after compiling, synthesis and adaptation with QuartusII. Firstly function simulation is carried out to verify the correctness of the design. Timing stimulation is carried out after successful verification. In order to save space, the report after compiling, synthesis and adaptation is pasted with timing simulation report together in the paper as shown in Fig.8.

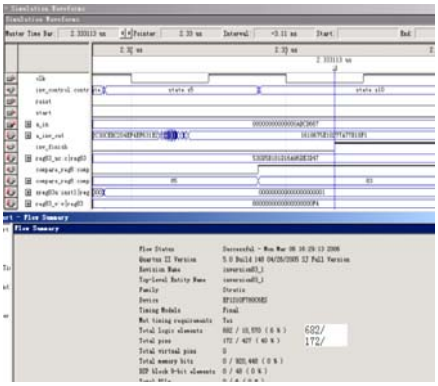

Fig.8 Timing simulation chart of inverse calculator in improved MIMA Algorithm

It can be seen from Fig.8 that , Total logic elements are 682 and Total pins are 172 according to stimulation result, which shows that the implementation of improved MIMA Algorithm occupies 682 logic units. Judging from the simulation chart, what clk adopts is 100M clock, so the highest frequency of this inverse calculator can reach 100M.

The improved algorithm optimizes the deg module and sub module in Fig.5, so the resource occupation does not add so much. The improved algorithm can not enable the speed to improve one time, because the speed improvement also has something to do with the number itself to calculate inverse. Under the worst situation, the efficiency of this implementation is equal to MIMA Algorithm. But the speed is obviously improved in the general situation.

Taking 0xabcd667 calculating inverse (inverse is 0x1618675e10277a77b18f1) as example, the performance comparison of inverse calculators designed separately according to three algorithms is shown as in Table 2.

Table 2 Comparison of three algorithms to implement $GF(2^{83})$

Algorithm	Les (piece)	Time (us)	Inverse time (us)	Notes
1	999	13.50	2.96	Without use of barrel shifter
2	650	3.10	2.42	
3	682	2.33	1.79	

5. Conclusion

HECC application needs a large amount of fast modules. Calculating inverse in finite fields is one of very important modules. The optimized MAIA Algorithm can effectively take advantage of the character that two adjacent bits of random number have larger probability to be zero and use parallel structure to speed up the inverse calculation in finite fields. At the same time, the optimized MAIA Algorithm has improved the default that the past algorithm takes into account of $\deg\_u$ and $\deg\_v$ separately, leaved out the module to evaluate $\deg\_v$ and saved large amount of chip resources. It can take into account to increase parallel degree for further study. For example, the situation of low four bits can be continuously determined. Furthermore, for hardware implementation, it is an issue worthy of further study how to make the flow more reasonable and thus increase the throughput of data. In a word, the algorithm brought forward in this paper is improved in both speed and area compared with past algorithms. It is believed that this kind of improved algorithm can be applied in large amount in future HECC implementation.

References

[1] Y.Sakai and K.Sakurai, On the practical performance of hyperelliptic curve cryptosystems in software implementations, IEICE Transaction Fundamentals,vol.E83-A Apr 2000,pp.692-703

[2] N.Smart, On the practical performance of hyperelliptic curve cryptosystems, in Eurocrypt, vol.1592 of Lecture Notes in Computer Sceince,1999,pp.165-175

[3] T. Wollinger. Software and Hardware Implementation of Hyperelliptic Curve Cryptosystems. PhD thesis, Department of Electrical Engineering and Information Sciences, Ruhr-Universitaet Bochum, Bochum, Germany, July 2004

[4] L.Song and K.Parhi, Low-energy digit-serial/parallel finite field multipliers, Journal of VHDL Signal Processing Systems, 1997.pp.1-17,

[5] Chang Hoon Kim1, Soonhak Kwon2, Jong Jin Kim1,etc. A New Arithmetic Unit in GF(2m) for Reconfigurable Hardware Implementation. Springer-Verlag Berlin Heidelberg .2003, FPL 2003, LNCS 2778, pp. 670–680
[6] JOSÉ LUIS IMAÑA. Bit-Parallel Arithmetic Implementations over Finite Fields GF(2m) with Reconfigurable Hardware. Acta Applicandae Mathematicae © 2002 Kluwer Academic Publishers. 73: 337–356

Bao kejin is an Associate Professor in the School of Computer Science and Telecommunication Engineering, Jiangsu University, China. He received the M.D. from Jiangsu University in 1993. Now, he research interests include hardware implementation of algorithm, network security, embedded system application, real-time control system.