High-Performance Implementation of SM2 Based on FPGA

Dan Zhang
Department of Microelectronics
Tsinghua University
Beijing, China
e-mail: zhangdan10560363@163.com

Guoqiang Bai
National Lab for Information Science and Technology
Department of Microelectronics
Tsinghua University
Beijing, China
e-mail: baigq@mail.tsinghua.edu.cn

Abstract—**This brief presents an FPGA-based ultra-high performance ECC implementation over SM2 prime field which can resist SPA. This processor is designed with bottom-up optimization focused on SM2 and make the best of advantages of modern FPGA. To counteract SPA more efficiently and reduce time cost, traditional MPL algorithm is modified to be the main algorithm which can execute point addition (PA) and point double (PD) in parallel. Then PA and PD are designed to be full-isochronous modules invoked by main algorithm to maximize the efficiency. Finite field operations adopt DSP blocks to increase frequency. Spliced multipliers are matched with same-frequency adders in the introduced pipeline structure, which improve hardware utilization to more than 95 percent. Run on Altera StratixII EP2S30F672 FPGA, this SM2 processor whose frequency reaches 62.3 MHz can be performed at a rate of about 1.3k point multiplications per second, and it only costs 8 DSPs and 4742 ALMs. Compared with other related works, our architecture offers not only ultra-high performance but also deep research about the FPGA-based implementation of SM2.**

Keywords-ECC; SM2; SPA; FPGA; pseudo-Mersenne

I. INTRODUCTION

To insure the data security of ever growing network communication, public-key cryptographic algorithms are becoming more and more indispensable, while also facing with more and more security threats. As we all know, public-key cipher algorithms are extremely arithmetic intensive since their security assumptions rely on hard problems. So with the improvement of computation capacity, public-key cryptographic systems must promote their computation complexity or even adopt more advanced algorithms to ensure safety. Since Miller [1] and Koblitz [2] proposed elliptic curve cryptography (ECC) in 1985 independently, it has always been a hot subject in the public-key research field, and has become the hottest choice to replace traditional ones. ECC can offer higher level of security than classical ones such as RSA and Diffie-Hellman with the same key sizes, providing greater computational efficiency and safety. At the same time, its mathematic background is more complicated and multivariate, which can provide more possibilities to satisfy the demand of different application environments.

Based on the broad application prospects, several well-known international standard organizations have accepted and standardized ECC successively, such as NIST [3], ANSI [4] and IEEE [5]. Then in December 2012, the State Cryptography Administration of China also published the national public-key cipher algorithm [6] based on ECC, named SM2. It is defined over a pseudo-Mersenne prime field called P_{SM2}. The special parameters can simplify the complicated operations in prime field considerably, and furthermore promote the application performance of SM2.

Having both optimization potential and government support, SM2 is now embracing a bright future. But due to the relative late birth, SM2 hasn't got sufficient study, especially in the field of implementation based on FPGA. There have been a lot of researches that aim at the FPGA implementation of ECC, but to date, there appear to be extremely few designs that focus only on SM2 and make the best of all its features, not to mention the achievement of high performance. In this paper, we conduct a detailed study about both the FPGA implementation of ECC and the algorithm structure of SM2, then propose a high performance design of SM2 implementation based on FPGA. This design is actually validated on Altera StratixII EP2S30F672 FPGA and achieves good operating result in line with expectations.

The remainder of this paper proceeds as follows. Section II presents the mathematic background of ECC and SM2. Section II introduces our high performance SM2 processor based on FPGA. The experimental results as well as comparison with previous work are given in Section IV, and we conclude in this section, too.

II. MATHMATICAL BACKGROUND

A. Definition of SM2

A non-supersingular elliptic curve over GF(p) is defined to be the set of solutions (x, y) $\in$ GF (p) $*$ GF (p) to the equation in Eq. (1).

$$E: y^2 = x^3 + ax + b \tag{1}$$

where $a, b \in$ GF (p), $4a^3 + 27b^2 \neq 0 \pmod{p}$, together with the point at infinity denoted by P_∞. As mentioned, SM2 is defined over a unique prime field and its parameters in equation have been specified in [6], shown as follow:

(1) The pseudo-Mersenne prime field:

$$p_{SM2} = 2^{256} - 2^{224} - 2^{96} + 2^{32} - 1$$

(2) Parameters in Weierstrass equation:

978-1-5090-1781-2/16/ $31.00 ©2016 IEEE

$$a = p_{SM2} - 3$$

$$b = 28e9fa9e9d9f5e34\ 4d5a9e4bcf6509a7f39789f5\ 15ab8f92ddbcbd41\ 4d940e93$$

B. Arithmetic Formula of SM2

Elliptic curve forms a commutative finite group under the addition operation of "tangent and chord rule", and point P_∞ acts as the group identity. This addition formula varies for the case of equal and unequal points, called point addition and point double, given by Eq. (2). Note that these addition, subtraction, multiplication and division in equation are all finite field operations. Let $P = (x_1, y_1) \in E$, then $-P = (x_1, -y_1)$, and for all $P \in E$, $P_\infty + P = P + P_\infty = P$. If $Q = (x_2, y_2) \in E$ and $Q \neq -P$, then $P + Q = (x_3, y_3) \in E$, where:

$$\begin{cases} x_3 = (\frac{y_2 - y_1}{x_2 - x_1})^2 - (x_1 + x_2) \\ y_3 = (\frac{y_2 - y_1}{x_2 - x_1})(x_1 - x_3) - y_1 \end{cases} \quad \text{when } P \neq Q \qquad \begin{cases} x_3 = (\frac{3x_1^2 + a}{2y_1})^2 - 2x_1 \\ y_3 = (\frac{3x_1^2 + a}{2y_1})(x_1 - x_4) - y_1 \end{cases} \quad \text{when } P = Q \tag{2}$$

As we all know, multiplication equals repeated additions in general, and this also works for ECC addition formula. Repeated point addition and double constitute point multiplication, defined as:

$$kP = \sum_1^k P = P + P + \cdots + P \tag{3}$$

In practical applications, ECC is used to perform different functions, including encryption/decryption, digital signature and key establishment, but the only processing core is point multiplication. It is computed by a series of point addition and point double, further decomposed into a certain number of finite field operations, and finally translated into usual mathematical operations which can be performed by digital circuits.

C. Formula Simplification

The addition formula of ECC varies for different coordinates, so by clever using this kind of transformation, we can make the algorithm easier to be realized in hardware. Since affine coordinates require the complicated inversion operation, we focus on projective coordinates to avoid it. In some algorithms, operation formulas can be simplified by using extra conditions about data. In year 2002, Brier and Joy [7] deduced a kind of simplified formula for point addition and double in ECC. That is, the addition of two points with fixed difference can be computed without y-coordinates, and the y-coordinates can also be recovered at any time, shown as Eq. (4), (5) and (6). In projective coordinates, let point $Q_0 = (X_0, Y_0, Z_0) \in E$, $Q_1 = (X_1, Y_1, Z_1) \in E$, and they have fixed difference defined as $Q_1 = Q_0 + P$ (x_p, y_p), then

$$\begin{cases} X(Q_0 + Q_1) = (X_1X_2 - aZ_1Z_2)^2 - 4bZ_1Z_2(X_1Z_2 + X_2Z_1) \\ Z(Q_0 + Q_1) = x_p(X_1Z_1 - X_2Z_2)^2 \end{cases} \tag{4}$$

$$\begin{cases} X(2P) = (X_1^2 - aZ_1^2)^2 - 8bX_1Z_1^3 \\ Z(2P) = 4Z_1(X_1^3 + aX_1Z_1^2 + bZ_1^3) \end{cases} \tag{5}$$

$$y_1 = (2y_p)^{-1} x_2(x_1 - x_p)^2 - (a + x_1x_p)(x_1 + x_p) - 2b \tag{6}$$

Obviously, the simplified formulas can save almost all the calculation and storage cost for y-coordinates. If the fixed difference of points can be maintained during the computation, then the simplified PA/PD formulas would be able to be used in calculating kP, which will promote the performance of ECC processor greatly.

III. SM2 Processor Based on FPGA

Different hardware platforms, such as ASIC, smart card and FPGA, all have distinct characteristics. Focusing on one platform and making good use of its advantages is the assurance to achieve the high performance design. In this section, modified MPL algorithm is firstly introduced to be the main algorithm after a complete consideration. Then, based on the features of FPGA, a high performance SM2 processor is designed and optimized from the low-level circuit up to the high-level algorithm, which will be presented in detail in the rest of this section.

A. Selection for Main Algorithm

The way to implement an algorithm in hardware is to decompose the complex mathematical computation into operations executable for hardware, up down and level by level. In this process of SM2, main algorithm is the first step that translates the point multiplication into a series of point addition and double. Since there have been a wealth of researches regarding the main algorithm of ECC, both efficiency and security should be taken into consideration to choose the optimal one. Traditional algorithms read the bit of random number *k* serialized and transform *k* times of PA into $\log_2 k$ times of PA and PD. Different from traditional ones, Double-and-Add-always algorithm performs indiscriminate operations every cycle to resist SPA, but introduces redundant operations. Montgomery Powering Ladder [8] is the improved version of DAA which eliminates the redundancy. What's more, in this algorithm, two points to be added every cycle have the fixed difference so that the formula simplification can be adopted. We also modify the data schedule so that PA and PD can be executed in parallel, shown as Algorithm 1. The modified MPL can promote the performance of SM2 and resist SPA more efficiently, so it is selected to be the main algorithm of our processor.

Algorithm 1 Modified Montgomery Powering Ladder

Input: integer k and poin P, *l*=bit length of k
Initial: $Q_0 = P$, $Q_1 = 2P$, $Q_T = 0$, $i = l-2$
While $i \geq 0$ do
1: If $k_i = 0$ $Q_1 = Q_0 + Q_1$, $Q_0 = 2Q_0$
2: If $k_i = 1$ $Q_0 = Q_1 + Q_0$, $Q_1 = 2Q_1$
3: $i = i - 1$
End while
Output: $kP = \sum_1^k P = Q_0$

B. Optimization for Basic Operation

To satisfy the increasing demand for data processing, modern FPGA devices are usually equipped with specified arithmetic hardcores to accelerate the digital signal processing, called Digital Signal Processer (DSP). DSP can be programmed to add, subtract and multiply efficiently with few extra logic elements. So they are very suitable to be adopted in ECC implementation, which can not only promote performance but also save logical elements.

Balanced path delay and high hardware utilization are necessary for a good FPGA design. Aimed at high performance SM2 processor, we'd better ensure that there is no other path needs longer delay than one multiplication in this circuit since multiplication is the unavoidable frequency bottleneck. To be more precise, first we splice DSPs to get multipliers with appropriate bit width and frequency. Then we hold all the other path delay under this frequency by choosing faster adders and balancing control logic, and at the same time, improve the hardware utilization as much as possible. It should be noted that the multiplier can only be connected to registers to avoid extra delay. And the adders can only be connected in series instead of parallel, shown in Fig. 1. Parallel connection will cut the addition frequency in half while series connection nearly changes nothing.

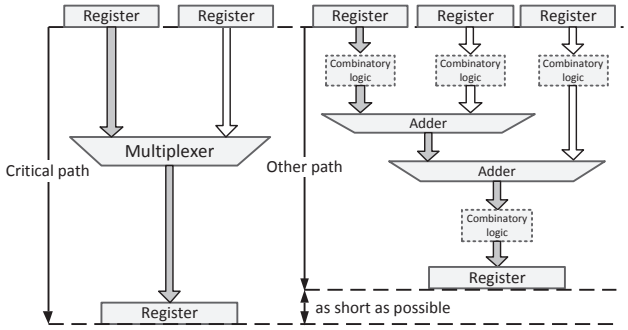

Figure 1. Balanced path delay

Our design is based on an EP2S30F672 FPGA, and its DSP can only be programmed to be a 36-bit multiplier. Since a SM2 processor needs to perform thousands of 256-bit multiplications, we splice four DSPs together into a 64-bit multiplier with frequency up to 62.9 MHz. Based on this FPGA, the 128-bit adder reaches frequency of 117MHz with appropriate hardware cost, which allows time for other logic and matches the 256-bit addition perfectly. Given all above, this SM2 processor is designed based on 128-bit adders and 64-bit multipliers made up of DSPs.

C. Optimization for Modular Addition/Subtraction

Different from usual addition/subtraction, modular addition/subtraction in finite field need to guarantee that the final result stay in the range of [0, P-1] before it can be used in next operation. And this can be achieved by adding or subtracting prime P to traditional results which are out of range. Once modular addition/subtraction needs three 128-bit addition and one 129-bit addition. Considering efficiency and frequency, we introduce a two-stage series of 129-bit adder into this modular unit, with first stage for usual add/sub and second stage for sub/add prime P. Addition and subtraction is performed from low 128 bits to high 128 bits, and the final result is chosen by the carry of subtraction. Besides this, due to the introduction of a add/sub control signal, this modular unit can perform both modular addition and subtraction and cost only two cycles for once, shown as Fig. 2. Let A, B $\in$ [0, P -1], LOW for 127-0 bits, HIGH for 255-128 bits. ADD is the control signal, ADD=1 for add, ADD=0 for subtract.

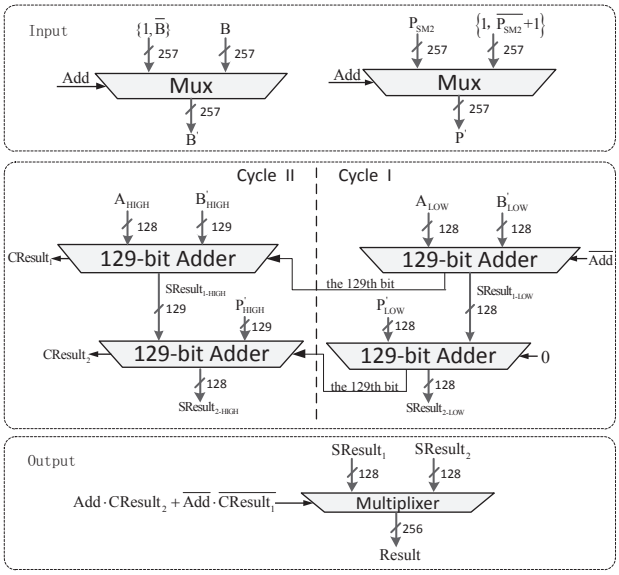

Figure 2. Architecture of modular addition/subtraction unit

D. Optimization for Modular Multiplication and Inversion

Once scalar multiplication usually needs thousands of modular multiplications, hundreds of modular additions or subtractions and only one inversion. Obviously, modular multiplication deserves to be the most important finite field operation. Modular multiplication is made up of 256-bit multiplication and 512-bit modular reduction. To improve hardware utilization and frequency, we introduce a two-stage pipeline to execute multiplication and reduction in parallel, which may also need some adjustment in data scheduling, example is shown in Fig. 3.

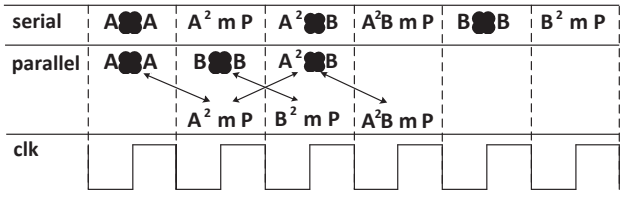

Figure 3. Example of data scheduling in pipeline modular multiplication

Since SM2 is defined over a pseudo-Mersenne prime field, modular reduction can be transformed into a fixed number of addition and subtraction, called Fast Reduction Scheme, shown in Algorithm 2. To execute one 256-bit multiplication with only one 64-bit multiplier, it needs 2 +

$(256/64)^2$ = 18 cycles, with two extra cycles for data preparation and partial products addition. Then we need to equip the pipeline with enough 128-bit adders to guarantee that the modular reduction can be finished in 18 cycles. To improve hardware implementation performance, we re-organize the traditional Fast Reduction Scheme to make the best of its characteristics.

Algorithm 2 Fast Reduction Scheme for P_{SM2}

Input: $M = m_{15}\cdot 2^{480} + m_{14}\cdot 2^{448} + \cdots\cdots + m_0$
$= (m_{15}, m_{14}, \cdots, m_0),\ m_i \in [0,\ 2^{32})$

1: define S_i as follows:

Bit	256 ~ 225	224 ~ 193	192 ~ 161	160 ~ 129	128 ~ 97	96 ~ 65	64 ~ 33	32 ~ 1
S_1	m_{15}		m_{15}	m_{14}	m_{13}		m_{15}	m_{14}
S_2	m_{14}						m_{14}	m_{13}
S_3	m_{13}							m_{15}
S_4	m_{12}							
S_5	m_{15}	m_{15}	m_{14}	m_{13}	m_{12}		m_{11}	m_{10}
S_6	m_{11}	m_{14}	m_{13}	m_{12}	m_{11}		m_{10}	m_9
S_7	m_{10}	m_{11}	m_{10}	m_9	m_8		m_{13}	m_{12}
S_8	m_9			m_{15}	m_{14}		m_9	m_8
S_9	m_8				m_{15}		m_{12}	m_{11}
S_{10}	m_7	m_6	m_5	m_4	m_3	m_2	m_1	m_0

2: $Sum = 2\cdot\sum_{i=1}^{4} S_i + \sum_{i=5}^{10} S_i - 2^{64}\cdot (m_8 + m_9 + m_{13} + m_{14})$

Output: $M \bmod p_{SM2} = Sum \bmod p_{SM2}$

Traditionally, one modular division needs to execute 18 times 256-bit modular add/sub, which is equivalent to 72 times 128-bit addition. Instead of executing 18 times of modular add/sub one by one, we perform all the usual addition and subtraction together and then the modular step. We find that the high 4 bits of multiple of prime P_{SM2} are very regular. It can simply be concluded as: if there is a 260-bit number N, it fulfills Eq. (7):

$$N[260:257] * P_{SM2} < N < (N[260:257] + 2) * P_{SM2} \quad (7)$$

Then we can judge the range of results easily and perform the modular operation by only two times addition. This is presented by Algorithm 3.

Algorithm 3 The speed up of modular reduction

Input: 260-bit integer $m[260:1]$, prime P_{SM2}

1: $\{C_1, Result_1[260:1]\} = m - m[260:251]\cdot p_{SM2}$

2: $\{C_2, Result_2[260:1]\} = Result_2 - p_{SM2}$

Output: If C_1=1 then $m \bmod p_{SM2} = Result_2$
else $m \bmod p_{SM2} = Result_1$

Besides, most S_i in Algorithm 3 have fixed null bits, and double operation can be realized by shift operation. These are also used to reduce the computation amount. At last, we only need 34 times 128-bit additions for one modular division. And this can be executed by two 128-bit adder in series within 18 cycles. We encapsulate modular multiplication into a unit with a 64-bit multiplier and two 128-bit adders and this unit costs only 18 cycles for once operation. Its frequency is just the frequency of multiplier which meets the design requirement. For the modular inversion, we adopt the radix-4 unified inversion algorithm to speed up the whole processor.

E. Optimization for PA and PD

In our main algorithm, PA and PD can be executed simultaneously. So we encapsulate PA and PD into two separate modules which can save control logic and promote speed as well. Shown as Eq. (4) and (5), both once PA and PD need 9 times modular multiplications and several modular addition/subtraction. By hiding these add/sub in parallel with multiplication, PA and PD can be executed strictly isochronously. However, there might be data dependencies between contiguous field operations in the formulas, and the extra cycles for pipeline must be taken into consideration either. Fortunately, we have found the optimal scheduling scheme. Table I gives the optimal execution order of PA and PD. Based on this, we have got isochronous PA and PD units with high hardware utilization and the frequency is still equal to a 64-bit multiplier. LM means the result of last modular step.

TABLE I. EXECUTION ORDER OF POINT OPERATION

Point Double		Point Addition	
Multiply	Register	Multiply	Register
$Z_1 \cdot Z_1$		$X_1 \cdot Z_2$	
$X_1 \cdot X_1$		$X_2 \cdot Z_1$	
$Z_1^2 \cdot b$	$T_2 = LM$	$Z_1 \cdot Z_2$	$T_1 = X_1 Z_2$
$X_1 \cdot Z_1$	$T_1 = LM$	$X_1 \cdot X_2$	$T_2 = X_2 Z_1$ $T_1 = T_1 + T_2$
$T_2 \cdot bZ_1^2$	$T_2 = T_1 - 3T_2$	$Z_1 Z_2 \cdot T_1$	$T_1 = T_1 - 2T_2$
$T_1 \cdot X_1 Z_1$	$T_2 = 2T_2$ $T_1 = T_1 + 3T_2$	$T_1 \cdot T_1$	$T_1 = X_1 X_2$ $T_2 = Z_1 Z_2$
$T_2 \cdot X_1 Z_1$	$T_2 = LM$	$LM \cdot 4b$	$T_1 = T_1 - 3T_2$
$T_1 \cdot T_1$	$Z = 4(T_2 + T_1)$	$LM \cdot x$	
$2X_1 Z_1^3 \cdot 4b$		$T_1 \cdot T_1$	$T_2 = LM$
	$T_1 = (X_1^2 + 3Z_1^2)^2$		$Z = LM$
	$X = T_1 - LM$		$X = LM - T_2$

F. SM2 Architecture

The whole architecture of SM2 processor is shown as Fig. 4. We combined main control FSM, point addition module, point double module and dual port RAM together to build the ECC core. To execute PA/PD independently, PA and PD modules are respectively equipped with a modular add/sub unit, a modular multiply unit and a control FSM with

registers. Register heap stores the real-time data, while the non-real-time data stored in the dual port RAM to save hardware. Main control module instructs all the other components to perform the main algorithm with high efficiency, just like an excellent governor.

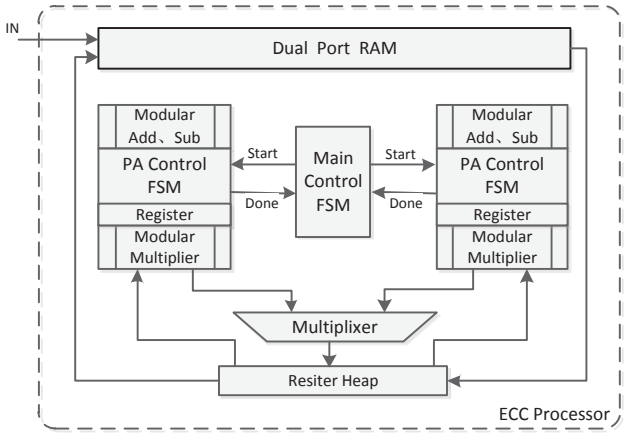

Figure 4. Architecture of ECC processor

IV. Comparison and Conclusion

This architecture has been verified in Verilog-HDL and actually run on FPGA. The experimental platform SASEBO-B is equipped with a control FPGA EP2S15F484 and a cryptographic FPGA EP2S30F672 which can be connected and exchange data. Our SM2 processor is run on the former while the data and result is transmitted and received by the later. By using this platform, we can consecutively test the SM2 processor with large amounts of data to get reliable results. Results are shown in Table II. Previously published results targeting 256-bit prime field and detailed performance comparison are also presented in this table.

TABLE II. Performance Comparison with Previous Work

Paper	Ours	[9]	[10]	[11]	[12]
FPGA	Altera EP2S30	Altera EP2S180	Altera EP2S30	Xilinx XC2VP125	Xilinx XCV4FX1
Field	GF(p) SM2	GF(p) SM2	GF(p) -256	GF(p) -256	GF(p) NIST
ALM	4742	8596	9177	0	0
Slice	0	0	0	15755	20793
Len of Mul	36	36	9	18	18
Num of Mul	8	64	96	256	32
Freq/MHz	62.3	62.9	157	39.5	60
Cycles/k	48.0	22	107	152	366
Time/ms	0.77	0.35	0.68	3.86	6.1

In this paper, we present an ultra-high performance point multiplication processor for SM2 based on FPGA. Run on Altera StratixII EP2S30F672 FPGA, this processor whose frequency reaches 62.3MHz can be performed at a rate of about 1.3k point multiplications per second, and it only uses 8 DSPs and 4742 ALMs. Superior performance comes from three points: thorough study and analyze on SM2, effective use of DSP and FPGA platform, hardware to algorithm optimization. For the low-level integer arithmetic operations, we splice DSPs together into high-performance multiplier and matches it with adders of appropriate frequency. For the field operation level, modular addition and subtraction are combined in one unit with a serial of adder to achieve higher efficiency. As for the most important modular multiplication, we optimize the Fast reduction scheme to the most, then multiplication and reduction are pipelined with exhaustive hardware utilization. For the point addition level, we rearrange the operation order of PA/PD and eliminate the delay caused by data dependency. For kP level, we modify the MPL algorithm to execute PA and PD in parallel and introduce computation simplification. All of these achieve this high performance processor.

Future work will target the following: 1)improving frequency by further segmenting the operation, 2)promoting security protection level, 3)supporting high-performance non-NIST processor based on FPGA.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (Grant U1135004), and by the National Key Basic Research Program of China (Grant 2013CB338004).

Reference

[1] Victor S Miller. Use of elliptic curves in cryptography. In Advances in CryptologyłCRYPTO85 Proceedings, pages 417– 426. Springer, 1985.

[2] Neal Koblitz. Elliptic curve cryptosystems. Mathematics of computation, 48(177):203–209, 1987.

[3] National Institute of Standards and MD Technology, Gaithersburg. Fips 186łdigital signature standard. 1994.

[4] WashingtonDC American National Standards Institute. X 9.62 public key cryptography for the financial services industry: Elliptic curve digital signature algorithm (ecdsa). 1999.

[5] Institute of Electrical and NY Electronic Engineers. P1363 standard specifications for public key cryptography. 2010.

[6] State Cryptography Administration of China. Public Key Cryptographic Algorithm SM2 Based on Elliptic Curves, 2010.

[7] Eric Brier and Marc Joye. Weierstraß elliptic curves and sidechannel attacks. In Public Key Cryptography, pages 335–345. Springer, 2002.

[8] Peter L Montgomery. Speeding the pollard and elliptic curve methods of factorization. Mathematics of computation, 48(177):243–264, 1987.

[9] Zhenwei Zhao. Research on sm(2) algorithm high speed integrated circuit implementation. 2015.

[10] Nicolas Guillermin. A high speed coprocessor for elliptic curve scalar multiplications over fp. In Cryptographic Hardware and Embedded Systems, CHES 2010, pages 48–64. Springer, 2010.

[11] Ciaran J McIvor, John V McCanny, et al. Hardware elliptic curve cryptographic processor over gf(p). Circuits and Systems I: Regular Papers,IEEETransactionson,53(9):1946–1957,2000

[12] Kendall Ananyi, Hamad Alrimeih, and Daler Rakhmatov. Flexible hardware processor for elliptic curve cryptography over nist prime fields. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, 17(8):1099–1112, 2009.