ELectronics EXpress

LETTER

IEICE Electronics Express, Vol.11, No.11, 1–12

# A low-time-complexity and secure dual-field scalar multiplication based on co-Z protected NAF

**Jizeng Wei**[a)], **Xulong Liu, Hao Liu, and Wei Guo**[b)]

*School of Computer Science and Technology, Tianjin University, Tianjin Key Laboratory of Cognitive Computing and Application, Tianjin, China*

a) *weijizeng@tju.edu.cn*

b) *weiguo@tju.edu.cn*

**Abstract:** In this paper, we incorporate the co-Z arithmetic with Protected NAF to build a dual-field scalar multiplication method in ECC (elliptic curve cryptography) with lower time complexity and higher security, called co-Z Protected NAF. The Protected NAF is the alteration of the original NFA against SPA (simple power analysis) attack. But the employed dummy operations, *double-and-add-always*, often results in two severe problems: the high time complexity and the vulnerability of safe-error attack. So, the speed advantage of co-Z point addition is leveraged to greatly compensate the time penalty incurred by Protected NAF. Meanwhile, not only does the co-Z not change the SPA immunity existed in Protected NAF, but the property of updating point P in it improves the security to resist the safe-error attack. Experiment results show that the co-Z Protected NAF can obtain 1.36 times speedup with respect to Protected NAF over GF(p), and is even faster than original NAF. And it can also counteract 30.7% time loss over GF($2^m$) caused by dummy operations. Furthermore, because the co-Z Protected NAF is only the optimization on scalar multiplication, only less than 1% extra area cost is generated to achieve its improvements in time complexity and security.

**Keywords:** elliptic curve cryptography (ECC), scalar multiplication, co-Z, non-adjacent form (NAF), safe-error attack

**Classification:** Integrated circuits

## References

[1] H. Darrel, S. Vanstone and A. J. Menezes: *Guide to Elliptic Curve Cryptography* (Springer, 2004).

[2] Y. Hitchcock and P. Montague: *Information Security and Privacy* (2002) 214.

[3] R. R. Goundar, M. Joye and A. Miyaji: CHES (2010) 65.

[4] N. Meloni: WAIFI (2007) 189.

[5] J. Marc and S.-M. Yen: CHES (2002) 291.

[6] A. Miyamoto, N. Homma, T. Aoki and A. Satoh: IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **19** (2011) 1136. DOI:10.1109/TVLSI.2010.2049037



DOI: 10.1587/elex.11.20140361
Received April 16, 2014
Accepted May 1, 2014
Publicized May 19, 2014
Copyedited June 10, 2014

[7] A. Satoh and K. Takano: IEEE Trans. Comput. **52** (2003) 449. DOI:10.1109/TC.2003.1190586

## 1 Introduction

Scalar multiplication, the fundamental operation in ECC, is usually computed over two typical types of filed arithmetic, i.e. prime field GF($p$) and binary filed GF($2^m$). Thus, rather than specifically requiring some field, it would be useful to combine them into a single datapath, referred to as dual-field architecture, to achieve the better tradeoff in relation to speed and silicon area. Of varied scalar multiplication algorithms, NAF (non-adjacent form) [1] has the widespread use due to its time complexity advantage based on the unique data representation. But its computation feature, *double-and-add*, often results in irregular operations during each iteration, which makes it easy to suffer from SPA attack. So the *double-and-add-always* based NAF algorithm, named Protected NAF [2], is proposed to regularize the iteration by the dummy operations. While it can protect NAF from SPA attack, the dummy operations increase the running time of NAF greatly, counteracting its natural speed advantage. Moreover, the dummy operations also provide opportunities to another fault attack technology called safe-error attack.

In this paper, from the perspective of speed and security, we propose the co-Z Protected NAF based scalar multiplication algorithm over the dual-field. The co-Z point addition [3] is extended to GF($2^m$) and combined with Protected NAF. The hardware implementation results show that, in this way, the lower time complexity of co-Z compensates the speed loss of Protected NAF caused by its dummy operations. Meanwhile, the characteristic of updating point $P$ in co-Z point addition as a countermeasure can make the co-Z Protected NAF own higher security and be capable of resisting SPA and safe-error attack.

## 2 Co-Z protected NAF based scalar multiplication over dual-field

### 2.1 Co-Z point addition over GF($p$)

It is generally known that point addition (PA) and point doubling (PD) are called iteratively in scalar multiplication. Therefore, one of the two has important impact on the scalar multiplication speed. In 2007, [4] introduced a new type of arithmetic that can significantly improve the addition speed of the points sharing the same Z-coordinate on an elliptic curve. [3] extended this idea to PA over GF($p$), named conjugate co-Z addition (Algorithm 1).

Note that except point addition denoted by part (1) Algorithm 1 also includes some extra operations tagged by part (2), namely updating $P$ value. The reason is that the Z-coordinates of the points are better to keep the same during the iterations of the scalar multiplication in favor of the speed improvement according to the co-Z criterions in [4], if there are continuous PAs in scalar multiplication. Consequently, while completing the addition of


DOI: 10.1587/elex.11.20140361
Received April 16, 2014
Accepted May 1, 2014
Publicized May 19, 2014
Copyedited June 10, 2014

two points, it is necessary to update the original point $P$ to ensure that the result $R$ and $P$ share the same Z-coordinate. Because the values of $W_1$, $A_1$ and $Z_3$ are computed in the point addition, so no extra time is consumed to update point $P$. This method makes the time complexity of PA with Jacobian format is reduced from $12M + 4S$ to $5M + 2S$ ($M$: modular multiplication, $S$: modular square), the details of which can refer to [3].

**Algorithm 1** co-Z point addition over GF($p$) (ZADD)

**Require:** $P = (X_1, Y_1, Z)$ and $Q = (X_2, Y_2, Z)$

**Ensure:** $(R, P) \leftarrow \text{ZADD}(P, Q)$ where $R \leftarrow P + Q = (X_3, Y_3, Z_3)$ and $P \leftarrow (\lambda^2 X_1, \lambda^3 Y_1, Z_3) = (X_p, Y_p, Z_p)$ with $Z_3 = \lambda Z$ for some $\lambda \neq 0$

$$co\text{-}Z\ point\ addition \begin{cases} C = (X_1 - X_2)^2 \\ W_1 = X_1 C;\ W_2 = X_2 C \\ D = (Y_1 - Y_2)^2;\ A_1 = Y_1(W_1 - W_2) \\ X_3 = D - W_1 - W_2 \\ Y_3 = (Y_1 - Y_2)(W_1 - X3) - A_1 \\ Z_3 = Z(X_1 - X_2) \end{cases} \tag{1}$$

$$update \begin{cases} X_p = W_1 \\ Y_p = A_1 \\ Z_p = Z_3 \end{cases} \tag{2}$$

**Algorithm 2** co-Z point addition over GF($2^m$) (ZADD)

**Require:** $P = (X_1, Y_1, Z)$ and $Q = (X_2, Y_2, Z)$ with elliptic curve parameter a

**Ensure:** $(R, P) \leftarrow \text{ZADD}(P, Q)$ where $R \leftarrow P + Q = (X_3, Y_3, Z_3)$ and $P \leftarrow (\lambda^2 X_1, \lambda^3 Y_1, Z_3) = (X_p, Y_p, Z_p)$ with $Z_3 = \lambda Z$ for some $\lambda \neq 0$

$$co\text{-}Z\ point\ addition \begin{cases} A = X_1 + X_2;\ B = Y_1 + Y_2 \\ C = (X_1 + X_2)^2;\ D = (X_1 + X_2)^3 \\ E = Z^2;\ F = E^2;\ G = EF;\ H = B^2 \\ Z_3 = ZA \\ X_3 = aGD + H + Z_3 B + D \\ Y_3 = (B + Z_3)X_3 + (BX_2 + AY_2)C \end{cases} \tag{3}$$

$$update \begin{cases} X_p = X_1 C \\ Y_p = Y_1 D \\ Z_p = ZA \end{cases} \tag{4}$$

## 2.2 Co-Z point addition over GF($2^m$)

Due to the widespread use of GF($2^m$), a extension of co-Z PA from GF($p$) to GF($2^m$) is extremely neccessary to improve the speed of dual-field scalar multiplication. So, in this paper we propose a co-Z PA over GF($2^m$) as shown in Algorithm 2. Just as co-Z PA over GF($p$), Algorithm 2 also comprises PA and the point $P$ updating. The computational amount of this update is limited, only spending $2M$, because the $C$, $D$ and $A$ have been used in the PA. Finally, the proposed co-Z point addition over GF($2^m$) pushes the time complexity from $15M + 5S$ to $11M + 4S$.


DOI: 10.1587/elex.11.20140361
Received April 16, 2014
Accepted May 1, 2014
Publicized May 19, 2014
Copyedited June 10, 2014

### 2.3 Scalar multiplication based on co-Z protected NAF

The NAF, a binary signed-digit representation of the scalar, is one of mathematical "tricks" applied to the scalar multiplication ($Q = kP$). The most distinctive feature of NAF is that for a random $k$ the average number of non-zero digits is only $l/3$ ($l = \lceil \log_2(k) \rceil$) compared with average $l/2$ for normal representation [1] and $l$ for Montgomery Ladder [5], resulting in less number of point addition. But NAF still belongs to the well-known *double-and-add* method that may be subject to SPA attack. A simple countermeasure is to insert dummy operations, called *double-and-add-always*.

Protected NAF (Algorithm 3), referred to [2], can resist SPA attack by dummy doubling (step 6) and addition (step 8). In this way, it completes the regular operations whatever the secret key bit is. But we observe that Protected NAF still has two problems by. On the one hand, dummy operations slow down the NAF. On the other hand, also is the most severe that dummy operations protect from SPA but providing an opportunity to one kind of attack, namely safe-error attack. In Algorithm 3, if the adversary injects temporary faults in step 8, the final result $Q_0$ will be faulty when $k_{i+1}$ is 0. Otherwise, the $Q_0$ is not affected. Similarly, if faults are injected into step 6, $Q_0$ will be faulty when $doubleTwice$ is 1, or $Q_0$ is still correct. Under the safe-error attack, $k_i$ and $k_{i+1}$ can be recovered by checking the correctness of $Q_0$. To the best of our knowledge, the vulnerability of safe-error attack in Protected NAF has never been discussed in previous works.

**Algorithm 3** Protected NAF based Scalar Multiplication [3]

**Require:** $P_0$ (the point to multiply), $k$ (the scalar in protected NAF format), $n$ (Number of bits in $k$)
**Ensure:** $Q_0$ such that $Q_0 = kP$
1: $Q_0 = \emptyset$
2: $P_1 = -P_0$
3: **for** $i = 0$ to $(n-1)$ by 2 **do**
4: $\quad doubleTwice = 0$ if $(k_i k_{i+1} == 10)$, and 1 otherwise
5: $\quad Q_0 = 2Q_0$
6: $\quad Q_1 = 2Q_0$
7: $\quad Q_0 = Q_{doubleTwice}$
8: $\quad Q_1 = Q_0 + P_{k_i}$
9: $\quad Q_0 = Q_{k_{i+1}}$
10: **end for**
11: Return $Q_0$

In terms of time complexity and security we attempt to fuse the proposed dual-field co-Z point addition with Protected NAF. If directly applying co-Z point addition to step 8, the computation time loss of Protected NAF must be compensated partly. And more importantly, this combination also can resist safe-error attack. As described in Algorithm 1 and 2, $P$ value needs to be updated after completing co-Z point addition. So, $Q_0$ in step 8 will be updated according the criterion of dual-field co-Z point addition, which must lead to the error of final $Q_0$ no matter what the value of $k_{i+1}$ is, if some faults are


DOI: 10.1587/elex.11.20140361
Received April 16, 2014
Accepted May 1, 2014
Publicized May 19, 2014
Copyedited June 10, 2014

injected in step 8. In this way, $k_{i+1}$ cannot be recovered by safe-error attack. Thus it can be seen that co-Z point addition not only can accelerate the processing speed but also defense the soft-error attack. But step 6 in Protected NAF is a point doubling. So, when being a dummy operation ($doubleTwice = 0$), it cannot be protected by co-Z point addition directly so that soft-error attack still works well. But, if the dummy point doubling in step 6 can be changed to point addition, the security vulnerability will be overcome by co-Z point addition. Following this idea, the original operations in Algorithm 3 should be reorganized as follows.

**Algorithm 4** co-Z Protected NAF Resistant Safe-error Attack

**Require:** $P_0$ (the point to multiply), $k$ (the scalar in protected NAF format), $n$ (Number of bits in $k$)
**Ensure:** $Q_0$ such that $Q_0 = kP$
1: $Q_0 = \emptyset$
2: $P_1 = -P_0$
3: **for** $i = 0$ to $(n-1)$ by 2 **do**
4: $\quad doubleTwice = 0$ if $(k_i k_{i+1} == 10)$, and 1 otherwise
5: $\quad Q_0 = 2Q_0$
6: $\quad Q_1 = Q_0 + P_{k_i}$
7: $\quad Q_1 = Q_0 + Q_{(doubleTwice||k_{i+1})}$
8: $\quad Q_0 = Q_{doubleTwice}$
9: **end for**
10: Return $Q_0$

Firstly, when $doubleTwice = 0$, the step 6 and 8 in Algorithm 3 are all dummy operations. The result $Q_0$ is only equal to $2Q_0$. The Protected NAF can be equivalently transformed as follows:

$$\begin{aligned} Q_0 &= 2Q_0 \\ Q_1 &= Q_0 - P_0 \\ Q_1 &= Q_0 + Q_1 \\ Q_0 &= Q_0 \end{aligned} \tag{5}$$

In this way, the expressions, $Q_1 = Q_0 - P_0$ and $Q_1 = Q_0 + Q_1$, are also dummy operations so that the result $Q_0$ is still $2Q_0$. But, the dummy point doubling in step 6 is replaced by another dummy point addition ($Q_1 = Q_0 + Q_1$). Secondly, when $doubleTwice = 1$ and $k_{i+1} = 1$, all the operations in Algorithm 3 are all useful. The result $Q_0$ can be deduced as follows:

$$\begin{aligned} Q_0 &= Q_1 \\ &= 2(2Q_0) \pm P_0 \\ &= (2Q_0 \pm P_0) + 2Q_0 \end{aligned} \tag{6}$$

For obtaining the similar expression structure with formulation (5), $(2Q_0 \pm P_0) + 2Q_0$ can be equivalently expressed as follows:


DOI: 10.1587/elex.11.20140361
Received April 16, 2014
Accepted May 1, 2014
Publicized May 19, 2014
Copyedited June 10, 2014

$$\begin{aligned} Q_0 &= 2Q_0 \\ Q_1 &= Q_0 \pm P_0 \\ Q_1 &= Q_0 + Q_1 \\ Q_0 &= Q_1 \end{aligned} \tag{7}$$

Obviously, any fault injected to formula (7) will result in the errors of the final $Q_0$. For the remainder case, that is $doubleTwice = 1$ and $k_{i+1} = 0$, only step 8 in Protected NAF ($Q_1 = Q_0 \pm P_0$) is a dummy operation, which can directly resist safe-error attack by co-Z addition. In this case, the final $Q_0$ is as follows:

$$\begin{aligned} Q_0 &= 2(2Q_0) \\ &= 2Q_0 + 2Q_0 \end{aligned} \tag{8}$$

Being consistent with the expression structure of formula (5) and (7), $2Q_0 + 2Q_0$ can be redefined as follows:

$$\begin{aligned} Q_0 &= 2Q_0 \\ Q_1 &= Q_0 \pm P_0 \\ Q_1 &= Q_0 + Q_0 \\ Q_0 &= Q_1 \end{aligned} \tag{9}$$

Although $Q_1 = Q_0 + Q_0$ is also a equivalent point doubling with step 6 in Algorithm 3, it is not a dummy operation in this case, meaning that any fault in it also will lead to error in the final $Q_0$. Taken together, the co-Z Protected NAF as shown in Algorithm 4 is the formulation of formula (5), (7) and (9) for all the cases. Compared with Algorithm 3, the dummy point doubling in step 6 of Protected NAF is converted to point addition. By the update feature of $P$ in proposed co-Z point addition, the security loophole produced by injecting faults into potential dummy operations, step 6 and step 8 in Algorithm 3, can be resisted ultimately. Moreover, each iteration in Algorithm 4 also executes regular operations based on different scalar bits. So co-Z Protected NAF does not alter the original SPA-resistance characteristic in Protected NAF. In conclusion, the co-Z Protected NAF can not only compensate for the computation time loss of Protected NAF due to the dummy operations, but also resist against SPA and safe-error attack.

## 3 Hardware implementation

### 3.1 Overview architecture of Co-Z protected NAF coprocessor

The overview architecture of scalar multiplication coprocessor based on the proposed dual-field co-Z Protected NAF is illustrated in Fig. 1, including arithmetic core, Jacobian converter, operation controller, RAM and et al. Operation controller and arithmetic core compose the core unit of proposed coprocessor, which is a three-level hierarchical structure. Level-1, the arithmetic core, supports dual-field modular operations. The Level-2 consists of the co-Z PA and PD controllers, which achieve PA (Algorithm 1 and 2) and PD over dual-field by invoking Level-1. Level-3 (co-Z Protected NAF controller),


DOI: 10.1587/elex.11.20140361
Received April 16, 2014
Accepted May 1, 2014
Publicized May 19, 2014
Copyedited June 10, 2014

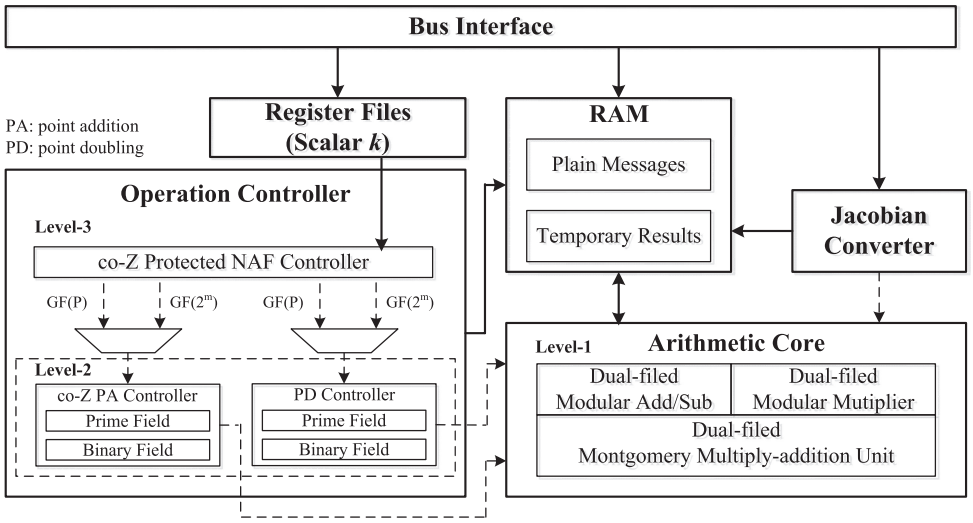


**Fig. 1.** Scalar multiplication coprocessor based on co-Z protected NAF

corresponding to Algorithm 4, completes the scalar multiplication with SPA and safe-error attack resistance. The basic flow of proposed coprocessor is as follows. The co-Z Protected NAF controller reads the scalar k from the register files and iteratively controls co-Z PA and PD controller according to Algorithm 4, respectively. Then, using field selection signals received from Level-3, Level-2 invokes the modular operations in arithmetic core to complete point addition and doubling over dual-field. The RAM stores plain message and temporary results. Because all the points in co-Z Protected NAF are Jacobian format, a Jacobian converter is designed to convert the initial points to Jacobian format from normal format and the final results must be converted back to normal format.

### 3.2 Arithmetic core based on FIOS

Note that this paper mainly takes advantage of co-Z based PA to improve the time complexity and security of Protected NAF. It just focuses on the level of scalar multiplication. So, its hardware implementation as Level 2 and 3 in Fig. 1 is only a simple finite-state machine (FSM), enabling the co-Z Protected NAF by invoking basic modular operations in Level 1 as long as following the control flow in Algorithm 1, 2 and 4. In other words, we give no more attention to the hardware design of modular operations and any state-of-art implementation can be directly applied to our coprocessor, which has no pros and cons effects on proposed algorithm.

So, a word-based (64-bit) high-radix Montgomery multiply-addition unit based on FIOS (Finely Integrated Operand Scanning) [6] is adopted by this paper to compute modular operations. Although the design in [6] only supporting modular multiplication over GF($p$), we just need to slightly modify the input datapath to complete dual-field modular operations. Fig. 2 illustrates the FIOS-based multiply-addition unit over GF(p) and GF($2^m$). We can see that the core function $z + xy + c$ to complete modular


DOI: 10.1587/elex.11.20140361
Received April 16, 2014
Accepted May 1, 2014
Publicized May 19, 2014
Copyedited June 10, 2014

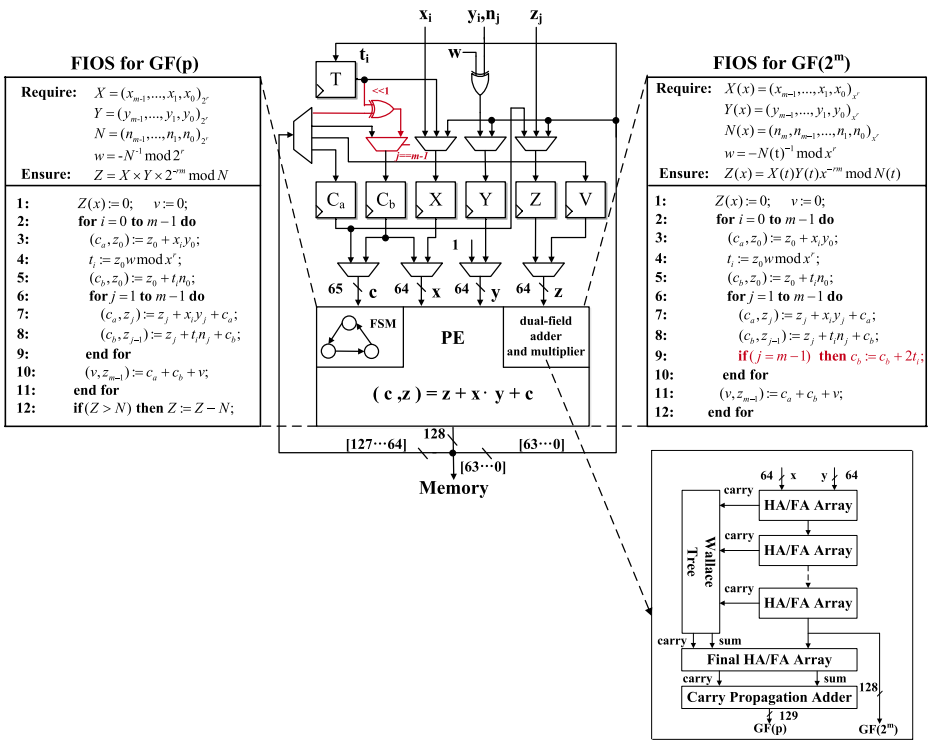


**Fig. 2.** Dual-field montgomery multiply-addition unit based on FIOS

multiplication, aka PE in FIOS method, is the same. The details of algorithm can be referred to [6]. $C_a$, $C_b$ and $z_j$ are the carries and intermediate results generated by each iteration, respectively. In each loop, $x_i$, $y_i$ and $n_j$ are the 64-bit inputs and modulus in modular multiplication. $t_j$ presents the constant value in FIOS. As described in [4], PE is made up of a FSM controlling iterative schedule, two 64-bit adders and one 64-bit multiplier. These two 64-bit adders can be further reused to implement modular addition and subtraction as long as setting the input x and y to "0". Note that the red part in Fig. 2, including one XOR gate and one multiplexer, is only active when computing over GF($2^m$), which is the only difference between FIOS-based multiply-addition unit over dual-field. This part is responsible for the step 9 in the FIOS over GF($2^m$), because the size of modulus over GF($2^m$) often goes beyond GF($p$) 2-bit that results in one time extra computation in the last iteration of each inner loop. Except this difference, whether over GF($p$) or GF($2^m$), the operation flow of PE is invariable. So, we rather just need the dual-field adder and multiplier. The architecture of dual-field adder can be constructed by some simple XOR and AND logics. Because multiplication can be completed by summing up the partial products, we can implement multiplication over GF($2^m$) through separating the results of XOR operations in the process of summing partial products. Then, the result of multiplication over GF(p) can be computed by adding the carries. In this way, a multiplier designed by Satoh [7] is introduced as shown in Fig. 2, utilizing the half/full adders to implement addition and multiplication simultaneously over dual-field.


DOI: 10.1587/elex.11.20140361
Received April 16, 2014
Accepted May 1, 2014
Publicized May 19, 2014
Copyedited June 10, 2014

## 4 Experimental results evaluation and comparison

### 4.1 Analysis of time complexity and security

So far, a wide range of algorithms are applied to scalar multiplication for different speed, area and power requirements, for example, Left-to-Right/ Right-to-Left (unsigned binary), NAF, Montgomery Power Ladder and so on. But the bottom implementations, namely basic modular operations, may be multifarious, making these algorithms hard to be compared directly in term of advantage and disadvantage. For fair comparison, in this paper, we firstly suppose that the related modular operations adopt the same method. In this way, the time complexity, gauged by the number of modular multiplication, can be directly utilized to qualitatively analyze the efficiency of different scalar multiplications. Then, the practical hardware results are compared to quantitatively verify the accuracy of previous analysis.

The number of point addition (PA) and point doubling (PD) depend on the scalar multiplication algorithm and the length of scalar $k$. In Jacobian coordinate, the complexity of PA and PD is $12M + 4S$ and $4M + 6S$ over GF($p$) as well as $15M + 5S$ and $5M + 5S$ over GF($2^m$), respectively. By the co-Z arithmetic the complexity of PA over GF($p$) and GF($2^m$) is reduced to $7M + 2S$ and $11M + 4S$. The comparison of different scalar multiplications is shown in Table I. Note that for briefness the time complexity is unified to the number of modular multiplication according to $S \approx 0.8M$ referred to [4].

**Table I.** Comparison of different scalar multiplication

| Algorithm | Number of PA | Number of PD | Time Complexity[2] | | Attack Resistance |
|---|---|---|---|---|---|
| | | | GF($p$) | GF($2^m$) | |
| Unprotected unsigned binary | $\frac{1}{2}$N[1] | N | (10M+8S)N[3] ≈16.4M·N | (12.5M+7.5S)N ≈18.5M·N | None |
| Original Montgomery Power Ladder | N | N | (16M+10S)N ≈24M·N | (20M+10S)N ≈28M·N | SPA Safe error |
| Unprotected NAF | $\frac{1}{3}$N | N | (8M+7$\frac{1}{3}$S)N ≈13.87M·N | (10M+6$\frac{2}{3}$S)N ≈15.3M·N | None |
| Protected NAF | $\frac{5}{9}$N | $\frac{10}{9}$N | (11$\frac{1}{9}$M+8$\frac{8}{9}$S)N ≈18.22M·N | (13$\frac{8}{9}$M+8$\frac{1}{3}$S)N ≈22.22M·N | SPA |
| co-Z Protected NAF | $\frac{35}{36}$N | $\frac{25}{36}$N | (7$\frac{23}{36}$M+6$\frac{1}{9}$S)N ≈12.53M·N | (14$\frac{1}{6}$M+7$\frac{13}{36}$S)N ≈20M·N | SPA Safe error |

[1]N stands for the length of scalar $k$.
[2]*Time Complexity = Time Complexity of PD × number of PD + Time Complexity of PA × number of PA.*
[3]M: modular multiplication; S: modular square; $S \approx 0.8M$

Referred to [5], the average number of PA and PD in the original Montgomery Power Ladder are all N. Therefore, its time complexity is the highest one in Table I. But Montgomery Power Ladder can resist both SPA and safe-error attack. As discussed in [1], because the unsigned binary computes PA only when the bit in scalar $k$ is not zero, the average number of PA is $\frac{N}{2}$, while it is further cut to $\frac{N}{3}$ in the original NAF. The time


DOI: 10.1587/elex.11.20140361
Received April 16, 2014
Accepted May 1, 2014
Publicized May 19, 2014
Copyedited June 10, 2014

complexity of NAF is reduced by 15.4% and 17.3% over GF($p$) and GF($2^m$) with respect to unsigned binary as shown in Table I. According to reference [3], the length of scalar $k$ presented by Protected NAF ranges from $\lceil \frac{N}{2} \rceil$ to $2\lceil \frac{N+1}{3} \rceil + (N+1) mod 3$. Upon the SPA-resistant feature of Protected NAF, each iteration in the scalar multiplication has regular operations. In this way, the average number of PA is equal to the average length of $k$, i.e. $\frac{5N}{9}$ and the average number of PD becomes $\frac{10N}{9}$ since each iteration has two times point doubling [2]. Although the Protected NAF can fix the loophole in NAF for the vulnerability of SPA attack, it sacrifices more time complexity (31.4% and 45.2% loss over GF($p$) and GF($2^m$)) than unprotected NAF.

The proposed co-Z Protected NAF has the same presentation of scalar $k$ with Protected NAF. In each iteration of Algorithm 4, the step 7 is completed by PA or PD with a probability of 75% or 25% according to the value of $k_i k_{i+1}$, respectively. So the average number of PA in co-Z Protected NAF is $(1+\frac{3}{4}) \times \frac{5}{9} N = \frac{35}{36} N$ in which "1" stands for the step 6 in Algorithm 4. And the average number of PD is $(1+\frac{1}{4}) \times \frac{5}{9} N = \frac{25}{36} N$, where "1" corresponds to the step 1 in co-Z Protected NAF. As shown in Table I, because of the introduction of dual-field co-Z PA, the time complexity of co-Z Protected NAF is lower than Protected NAF. Over the GF($p$), the number of modular multiplication in co-Z Protected NAF only occupies 68.8% of Protected NAF and even is smaller than original NAF. In the GF($2^m$), although the higher time complexity existing in the co-Z Protected NAF, it still compensates approximate 32% time loss caused by Protected NAF. Furthermore, there is also inevitable NAF conversion overhead in NAF-based algorithms. For co-Z Protected NAF, it consists of two parts: from scalar to NAF representation and then to Protected NAF format. Referred to the details of [2], both consume $2N$ cycles and $3N$ cycles (N is the length of scalar $k$). And this conversion is computed only once during one time scalar multiplication, resulting in total $5N$ cycles consumption. For the FIOS-based multiply-addition unit the time of once modular multiplication is about 81 and 41 cycles in average over GF($p$) and GF($2^m$) when the size of scalar $k$ in the range of 256-bit. So the conversion overhead only occupies about 0.49% ($\frac{5N}{12.53 \times 81 \times N}$) and 0.61% ($\frac{5N}{20 \times 41 \times N}$). It is too small to offset the time complexity improvement brought by co-Z Protected NAF.

Besides, with the help of updating point $P$ value in co-Z PA and reordering the control flow of Protected NAF, co-Z Protected NAF can resist not only SPA attack but also safe-error attack. Although, in the Table I, the Montgomery Power Ladder also can resist these two attacks, it has the higher time complexity. So the co-Z Protected NAF, by virtue of the properties in co-Z, reach the better tradeoff between time complexity and security.

### 4.2 Results comparison of the practical hardware implementation

In order to compare more practically and effectively, we implement all the scalar multiplication algorithms in Table I and show the comparison results over the dual-field and three kinds of scalar $k$ size based on 0.13 um CMOS


DOI: 10.1587/elex.11.20140361
Received April 16, 2014
Accepted May 1, 2014
Publicized May 19, 2014
Copyedited June 10, 2014

**Table II.** Speed and area comparison of hardware implementation

| Algorithm | Max Freq. (MHZ) | | N (bits)[1] | SM[2] Time (ms)[4] | | Area (gates)[4] |
|---|---|---|---|---|---|---|
| | GF($p$) | GF($2^m$) | | GF($p$) | GF($2^m$) | |
| Unprotected Unsigned binary | 303.56 | 418.41 | 160 | 0.87 | 0.28 | 69.4 K |
| | | | 192 | 1.02 | 0.32 | |
| | | | 256 | 1.85 | 0.65 | |
| Original Montgomery Power Ladder | | | 160 | 1.13 | 0.41 | 74.1 K |
| | | | 192 | 1.37 | 0.47 | |
| | | | 256 | 2.40 | 0.96 | |
| Original[3] NAF | | | 160 | 0.76 | 0.24 | 70.8 K |
| | | | 192 | 0.91 | 0.29 | |
| | | | 256 | 1.63 | 0.58 | |
| Protected[3] NAF | | | 160 | 0.92 | 0.31 | 72.6 K |
| | | | 192 | 1.10 | 0.34 | |
| | | | 256 | 2.10 | 0.77 | |
| co-Z[3] Protected NAF | | | 160 | 0.71 | 0.28 | 73.3 K |
| | | | 192 | 0.85 | 0.34 | |
| | | | 256 | 1.50 | 0.69 | |

[1]N stands for the length of scalar $k$.
[2]SM stands for scalar multiplication.
[3]The computation time and area for all NAF-based algorithms include NAF conversion.
[4]All the values are estimated by the authors.

technology (synthesized by Synopsys Design Compiler) in Table II. Furthermore, the FIOS-based modular operation unit adopted in this paper is applied in all the designs so as to remove the disturbance of different bottom implementations. So the computation time and area in Table II are all evaluated by ourselves for the fair comparison.

Firstly, although the time complexity over GF($2^m$) based on Jacobian coordinate, in Table I, is higher than GF($p$), the actual running time is less. The reason is that there is no carry chain over GF($2^m$), shortening the critical timing path and increasing operating frequency to compensate the time complexity penalty. As shown in Table II, the operating frequency is 303.56 MHZ over GF($p$) and goes up to 418.41 MHZ over GF($2^m$).

Secondly, the practical time of scalar multiplication in Table II shows the same trend with the time complexity in Table I. The time of co-Z Protected NAF over GF($p$) comes with 74% of Protected NAF and even is slightly less than the original NAF. For GF($2^m$) it makes up to about 30% time penalty from Protected NAF and is only 0.12 ms more than the original NAF. These evaluation results is consistent with Table I, which effectively verify the co-Z Protected NAF advantage on computation time achieve by co-Z PA with higher performance. In addition, the time of co-Z Protected NAF has included NAF conversion that only consumes about 5.3 µm and 3.4 µm in average over GF($p$) and GF($2^m$). It is trivial enough to be omitted against the time improvement, which is still in accord with the previous analysis.


DOI: 10.1587/elex.11.20140361
Received April 16, 2014
Accepted May 1, 2014
Publicized May 19, 2014
Copyedited June 10, 2014

Finally, because the algorithms discussed in this paper are all on the level of scalar multiplication, the differences between them only affect the scheduling order and iterative number of modular operations. And the hardware design of bottom modular operations are all based on FIOS-based multiply-addition unit. So there are no significant gaps in the area cost as shown in Table II. The area cost of proposed co-Z Protected NAF coprocessor is about 73.3 K gates (including memory area), which only increases by about 1% area with respect to the Protected NAF, resulting from the extra memory space to store some immediate results. Furthermore, the NAF conversion can also use the FIOS-based multiply-addition unit, not inducing extra area cost.

In conclusion, by practical hardware design and synthesis, the values in Table II eventually turn out that, based on the properties of co-Z arithmetic, co-Z Protected NAF can effectively offset the loss of time and security loophole in the Protected NAF and does not introduce extra cost area. Compared with other state-of-art algorithms, co-Z Protected NAF can reach the better tradeoff between computation time, security and area cost.

## 5 Conclusion

In this paper, we propose the co-Z Protected NAF based dual-field scalar multiplication algorithm and coprocessor with low time complexity and high security. The co-Z arithmetic drops the time complexity of PA over GF($p$) from $12M + 4S$ to $5M + 2S$ as well as the reduction in PA over GF($2^m$) from $15M + 5S$ to $11M + 4S$. Upon this advantage, co-Z Protected NAF is designed to compensate the time loss of Protected NAF due to using dummy operations to resist SPA attack. Experimental results show that the time complexity of co-Z Protected NAF over GF($p$) is indeed lower than original NAF. Even over GF($2^m$), it still counteract about 30% time penalty against Protected NAF at the expense of less than 1% area cost. Moreover, through reordering the control flow of Protected NAF, another feature of co-Z point addition, i.e. updating the value of point $P$, is leveraged to patch the security loophole of Protected NAF, when encountering the safe-error attack. In short, the proposed co-Z Protected NAF not only can accelerate the scalar multiplication with minimal area cost but also strengthen the security to resist SPA and safe-error attacks.

## Acknowledgments

This work is supported in part by both of the Natural Science Foundation of Tianjin (No. 11JCZDJC15800) and the Open Project Program of State Key Laboratory of Computer Architecture, the Institute of Computing Technology, Chinese Academy of Sciences.


DOI: 10.1587/elex.11.20140361
Received April 16, 2014
Accepted May 1, 2014
Publicized May 19, 2014
Copyedited June 10, 2014