New Attacks on Randomized ECC Algorithms

Zhijie Jerry Shi and Fan Zhang
Department of Computer Science and Engineering, University of Connecticut, Storrs, CT 06269, USA
Email: {zshi, fan.zhang}@engr.uconn.edu

***Abstract*- Elliptic curve cryptography (ECC) has attracted a lot of attention because it can provide similar levels of security with much shorter keys than the arithmetic of multiple-precision integers in finite fields, which has been widely used in many public-key and key-exchange algorithms. Small key sizes are especially important to resource constrained devices as shorter keys require less storage space and consume less power to transmit and compute. However, ECC algorithms are vulnerable to power analysis attacks, which exploit the instantaneous power consumptions of computing devices to retrieve secret data. Many countermeasures have been proposed to make ECC implementations secure against power analysis. One of the approaches is randomized algorithms that generate different power traces even if the input of the algorithm is the same. For example, the randomized scalar point multiplication algorithm proposed by Oswald et al. combines two algorithms and uses random variables to decide which algorithm to follow at different stages of the execution. The randomized algorithm can thwart traditional power analysis attacks. However, in this paper, we propose an effective attack on the randomized algorithms. Our attack does not require a large number of power traces and has a very high success rate.**

I. INTRODUCTION

The arithmetic of large integer in finite fields has been adopted in many public-key algorithms, such as RSA and ElGamal, and key-exchange algorithms, such as Diffie-Hellman. In the middle 1980s, Miller and Koblitz proposed the elliptic curve cryptography (ECC) [1][2] as an alternative to large integer arithmetic. ECC has attracted a lot of attention because it can provide equivalent security strengths with shorter keys than traditional large integer operations. Smaller key sizes have many advantages including higher performances, smaller storage space, and lower energy consumptions. These advantages are especially important in resource-constrained environments, such as smart cards, cellular phones, and personal digital assistants (PDAs), where processing power, storage space, communication bandwidth, and power supply are limited [3].

Power analysis is a type of side channel attacks that exploit the power consumptions of cryptographic devices to reveal the secret data used in cryptographic computations. Mobile devices are especially vulnerable to this type of attacks as adversaries can easily gain the physical access to these devices. There are several types of power analyses, such as Simple Power Analysis (SPA) and Differential Power Analysis (DPA). SPA exploits the correlation between the power outputs and the operations. By observing the power trace of cryptographic computations, adversaries can learn what operations are performed and then figure out part or all of the secret data. SPA can be launched even with only one power trace. Unlike SPA, DPA uses statistical methods to test whether a specific value is generated during cryptographic computations and then deduces the secrets. Typically, DPA needs a large number of power traces. Both of SPA and DPA can be used to attack ECC implementations.

Some countermeasures have been proposed to make ECC systems secure against power analysis. One of the countermeasures is the randomized algorithm proposed by Oswald et al. in [4]. The randomized algorithm seems secure against traditional power analyses. However, we found it is vulnerable to a new type of SPA attacks.

The rest of the paper is organized as follows. Section II describes the basic operations of ECC and Section III describes power analysis attacks and some countermeasures. In Section IV, we discuss our attacks on the randomized algorithm. And Section V concludes the paper.

II. ELLIPTIC CURVE CRYPTOGRAPHY

An elliptic curve is a set of points (x, y) specified by a bivariate cubic equation defined over a field K [5]:

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6. \qquad (1)$$

where a_i ($i \in \{1, 2, 3, 4, 6\}$), x, and $y \in K$. If K is GF (2^n), equation (1) can be reduced to:

$$y^2 + xy = x^3 + ax^2 + b. \qquad (2)$$

where a, b, x, and $y \in$ GF(2^n).

The set of points on an elliptic curve, together with a special point O called the point at infinity and an addition operation, form an abelian group [6]. For elliptic curves defined over GF(2^n), the addition operation of points is defined as follows. If a point $P = (x_1, y_1) \neq O$, $-P = (x_1, x_1+y_1)$. Given another point $Q = (x_2, y_2) \neq O$ and $Q \neq -P$, the sum $(x_3, y_3) = P + Q$ can be computed as:

$$\begin{aligned} x_3 &= \lambda^2 + \lambda + x_1 + x_2 + a. \\ y_3 &= \lambda(x_1 + x_3) + x_3 + y_1. \end{aligned} \qquad (3)$$

where

$$\lambda = \begin{cases} \dfrac{y_2 + y_1}{x_2 + x_1}, \text{ if } P \neq Q \\ \dfrac{y_1}{x_1} + x_1, \text{ if } P = Q. \end{cases}$$

The multiplication of an integer d and a point P is defined as adding d copies of P together.

$$Q = dP = \underbrace{P + \cdots + P}_{d \quad times}.$$

III. Power Analysis and Countermeasures

In this section, we briefly describe the power analysis of ECC and some countermeasures [7].

A. *Power Analysis of ECC*

Scalar point multiplication, $Q = dP$, is the basic operation of ECC. It is equivalent to the exponentiation operation in the multiplicative group of large integers. It can be performed with the binary algorithm BINALG as shown in Fig. 1a. In the figure, l is the number of bits in d and $d_{l-1} = 1$.

In BINALG, point doubling $2Q$ is performed in every iteration while point addition $Q+P$ is performed only for 1 bits. Since point doubling and point addition are performed with different formulae (see Section II) and thus have different power outputs, BINALG is vulnerable to SPA. An adversary can monitor the power consumption of a device performing BINALG and identify the point addition and doubling operations. A doubling followed by an addition indicates a 1 bit in d and two consecutive doublings indicate a 0 bit. The adversary can easily retrieve the value of d, which is supposed to be secret in many algorithms.

To make an algorithm SPA resistant, one can remove the correlations between the execution path and the data being processed. In BINALG, the execution of point additions depends on the value of d_i. The additions are performed only when $d_i = 1$. In order to thwart SPA attacks, Coron [5] proposed BINALG′, which is shown in Fig. 1b. BINALG’ is a variant of BINALG. It always performs the point addition. The correct value is selected by d_i and carried over to the following iterations. According to [5], BINALG′ is secure against SPAs. However, Coron discovered that it is vulnerable to DPA [5]. Oswald and Aigner further pointed out that any similar algorithms are also vulnerable to DPA attacks [4].

B. *Randomized Algorithm*

Oswald and Aigner proposed a randomized algorithm to thwart both SPA and DPA. They combined a variant of BINALG and an algorithm based on addition-subtraction chains [8]. The combined algorithm can be illustrated by the automaton in Fig. 2, which will be referred to as Randomized Automaton 1. In the automaton, there are three states: State 0, State 1 and State 11. When performing a scalar point multiplication dP, we start from State 0 and scan the bits in d from the least significant bit. For every bit, we either advance to a new state or stay in the same state, as specified in Fig. 2, and perform proper operations during transitions.

Since Randomized Automaton 1 is a combination of two automata, a random variable is used in States 1 and State 11 to decide which of the automata to follow. When the automaton enters State 1 or State 11, a random variable e is drawn. If the next bit is 1, only the transition with the correct random value will be followed. When the automaton is in State 1 and the next input bit is 1, for example, the automaton will stay in State 1 if $e = 1$. Otherwise it will go to State 11. The transitions dependent on random variables are illustrated with dashed lines in Fig. 2.

```
BINALG
Input :    P and d
Output:    Q = dP
Q←P
For i from l–2 to 0 do
    Q←2Q
    If d_i =1 then Q←Q+P
Return Q
```

(a) Binary algorithm

```
BINALG′
Input:     P and d
Output:    Q[0] = dP
Q[0]←P
For i from l–2 to 0 do
    Q[0]←2Q[0]
    Q[1]←Q[0]+P
    Q[0]←Q[d_i]
Return Q[0]
```

(b) Double-and-add algorithm

Fig. 1: Scalar point multiplication algorithms

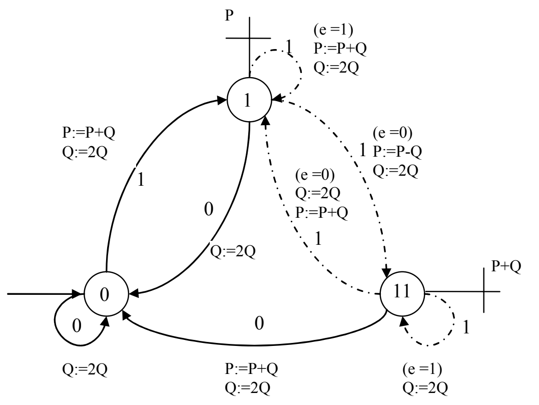

Fig. 2: Randomized Automaton 1 [4]

Because of the random variables, the scalar point multiplication algorithm follows different paths and thus performs different operations even if the scalar d remains the same. Therefore it is difficult to launch traditional SPA or DPA to Randomized Automaton 1.

IV. Attacks on Randomized Algorithms

A. *Attacks on Randomized Automaton 1*

Randomized Automaton 1 seems secure against power analyses. Each of its executions generates a different power trace even if the scalar d remains the same. It is difficult to find out d by examining only one trace. However, we can collect many power traces. Each of the traces will tell us a different bit of information about the same d. With enough number of power traces, we can figure out all the bits in d.

Let us denote the doubling operation $2Q$ in Randomized Automaton 1 as D and the addition or subtraction operation ($P + Q$ or $P - Q$) as A. Here, we assume we cannot distinguish additions from subtractions. Also, we will refer to the bits in d as input bits as they are the input string to the automaton during the computation.

When launching the attack, we first collect a set of power traces by performing the scalar point multiplication N times. We assume that we can tell what operations, additions or doublings, were performed from the power traces. The point in the *N* multiplications can be the same. The power traces will be different because of the random variables.

Our strategy is to guess the states each power trace has visited. If we know all the states a power trace visited, we can retrieve the bits in the scalar. Although all the power traces start from State 0, they may visit different states, which are decided by random variables. After an input string is fed into the automaton, the states of the power traces can be placed into three categories. 1) All the power traces are in State 0, 2) all the power traces are in State 1, and 3) some of the power traces are in State 1 and some of the power traces are in State 11. We call these categories AS0, AS1, and AS1/11, respectively. AS stands for *attack state*. In an attack, we start from AS0 and transit to one of the attack states when some input bits are detected. In AS1/11, the states of power traces could be either State 1 or State 11. The uncertainty is introduced by the random variables. The power traces are evenly distributed into State 1 and State 11.

In any of the attack states, we will try to figure out one or several input bits a time by looking at the operations performed in *all* traces. From the input bits we detected, we determine the corresponding power outputs and the new state for each power trace. Then we append some of the detected input bits to the recovered input string and remove their corresponding outputs from the power traces. We say a bit is *committed* if the bit is appended to the recovered input string and its corresponding power outputs are removed from all power traces. After some or all detected bits are committed, we go to a new attack state with shortened power traces and continue to detect more input bits until all input bits are recovered.

We now look at in each attack state how to detect the input bits from the power outputs and how to transit from one attack state to another. The cases in AS0 are trivial because all the power traces are in State 0 and the power outputs reflect the input bits. If we see *A* is the first operation in the power traces, the input bit must be 1. If we see *D*, then the input bit is 0. We commit the detected bit. The power outputs corresponding to the detected bit, either D or AD, are removed from power traces. We then move into the next attack state, which is AS0 if the input bit is 0 and AS1 otherwise.

The cases in AS1 are also trivial. If D is the first operation in any power trace, the input bit must be 0. If the first operation is A, then the input bit is 1. After we commit this bit, the next attack state is AS0 if the bit is 0 or AS1/11 if it is 1. In real attacks, however, we detect three input bits in AS1 each time, commit only one bit, and withhold two bits before going into AS1/11.

The cases in AS1/11 are more complicated because the same input bit may generate different power outputs and different input bit may generate the same outputs. If the input bit is 1, the power output bit can be AD (e.g., from State 1 to State 1) or DA (e.g., from State 11 to State 1). If the input bit is 0, the power output bit can be D (e.g., from State 1 to State 0) or AD (e.g., from State 11 to State 0). AD can be generated by both 0 and 1. So looking at the power output of the next input bit does not tell us what the next input bit is.

The problem can be solved by considering three input bits a time in AS1 and AS1/11. Table I lists all possible power outputs of three input bits when the automaton is in State 1. The first column is the current automaton state. The second column is the three input bits. 0xx is a 3-bit string starting with 0. The third column is all possible power outputs that may be generated by the three input bits. The fourth column lists the detected bits to be committed. Note that in some cases, e.g., when going into AS1/11, we commit only one bit although three bits are known. We withhold other two bits and use them to help us identify further input bits in AS1/11. The last column lists the state after the committed bits are fed into the automaton.

Since AS1/11 can be reached only from AS1 or itself, we first look at AS1 again and consider 3-bit input strings. If the first bit of the 3-bit string is 0, the automaton will go to State 0 after the first bit. The value of the first bit can be revealed by the first operation in power traces. If *D* is the first operation, the next input bit must be 0 and the automaton will go to State 0, for all power traces. We commit the 0 bit and go to AS0.

If the 3-bit input string starts with 1, we would be in AS1/11 after the first bit. In this case, we would like to detect more bits in AS1 before going into AS1/11. There are four 3-bit strings starting with 1, as listed in Table I. Because the automaton takes random transitions, we observe different power traces even for the same input string. For example, for string 111, we may observe six different power traces. Meanwhile, a power trace can be generated by different input strings. For instance, *ADADD* can be generated by 111, 110, or 100. So normally we cannot determine the input string by looking at *only one* power trace.

Among all power outputs that may be generated by a 3-bit input string starting with 1, we noticed that some power outputs are unique to specific inputs. We call a power output sequence X *a unique sequence* to the input string Y if it satisfies the following conditions.

1) X can not be generated by two different input string of the same length, i.e., if both Y and Y′ generate X, Y ≠ Y′.

2) Adding or removing *A* and *D* operations at the end of X cannot convert it into another valid sequence X′ which can be generated by a different input string Y′ of the same length.

For example, *ADDAAD* is a unique sequence of 111 and it is the only unique sequence of 111. Note that although *ADDDA* is generated by 111 only, it is not a unique sequence. When the trailing A is removed, it becomes *ADDD*, a valid output of 111 and 100.

With unique sequences, we can test whether their corresponding strings are the correct input string. We check if any power traces start with a unique sequence. If a unique sequence is found, the corresponding string must be the input bits. For example, if a power trace starts with *ADDAAD*, we can conclude that the next three input bits must be 111. Note that the unique sequences do not appear in every power trace. However, once they appear, it is certain that the corresponding string is the correct input. Since transitions are decided by random variables, we should be able to observe all possible outputs, including the unique sequences, if we have enough power traces. If none of unique sequences appears in a large number of power traces, we can exclude the strings that have unique sequences and then identify the input string among other strings. For example, in State 1, if *ADDAAD* does not appear in a large number of traces, we can exclude 111 and look at 110, 101, and 100 only.

When strings are excluded, their power outputs are also removed from the power output set from which we look for unique sequences. New unique sequences will be identified for the remaining strings and then be used to check whether the corresponding strings are the correct input. We repeat the process until the correct input string is identified. For example, in State 1, after 111 is excluded, 100 has a unique sequence *ADDD*. We can check whether 100 is the correct input by searching for any power traces that start with *ADDD*. If no such power trace is found, we exclude 100 as well. Then 110 will have a unique sequence *ADADD*. After 110 is also excluded, 101 must be the correct input string as it is the only one left. So in AS1, we can identify all 3-bit input strings starting with 1.

After the correct input string is identified, we will commit one or three detected bits. When the input strings are 110, 101, and 100 we commit all the three bits because all power traces end up in the same state, either State 0 or State 1. We can then go to AS0 or AS1, respectively, after the bits are committed.

TABLE I: AD SEQUENCES OF THREE INPUT BITS IN STATES 1 AND 11 (RANDOMIZED AUTOMATON 1)

Current State	Input Bits	Power Traces	Bits Committed	Next State
1	0xx	D	0	0
	111	ADDD	1	11
		ADADD		1
		ADDDA		11
		ADADAD		1
		ADDAAD		11
		ADADDA		1
	110	ADADD	110	0
		ADDAD		
		ADADAD		
	101	ADDAD	101	1
		ADADAD		
	100	ADDD	100	0
		ADADD		
11	111	**DDD**	1	11
		DDDA		11
		DDAAD		11
		DAADD		1
		DAADDA		1
		DAADAD		1
	110	DDAD	110	0
		DAADD		
		DAADAD		

When the input string is 111, the automaton could be in State 1 or State 11 and the attack state becomes AS1/11 after all three detected bits are committed. In this case, we commit only one bit and withhold two bits. The first 1 bit also makes the new attack state become AS1/11. However, with the withheld bits, the input string to be detected in AS1/11 can only be 110 or 111. When only two strings are considered, we can find a unique sequence for 111 no matter a power trace is State 1 or State 11.

We now look at the cases in AS1/11 in detail. AS1/11 can be reached from AS1 or AS1/11. If 111 is detected in AS1, we commit the first 1 and get into AS1/11. When 111 is detected in AS1/11, we also commit the first 1 only and stay in AS1/11. In either case, we have two bits 11 withheld when we transit to AS1/11. Therefore, only two input strings, 110 and 111, need to be considered in AS1/11, and we need to focus only on the power traces that may be generated by these two strings. In addition to the power outputs generated in State 1, the possible power outputs generated by 110 and 111 in State 11 are also listed in Table I. It can be computed that 111 has several unique sequences: *ADDAAD* (for power traces in State 1) and *DDAAD*, *DDDA*, and *DDD* (for power traces in State 11). If any of the unique sequences is found in power traces, the input bits are 111. We commit the first 1 and stay in AS1/11 with 11 withheld. If none of the unique sequences is observed, the input bits are 110. We commit all the three bits (110) and transit to AS0.

Table II summarizes the attack state transitions. The first column is the current attack state. The second column is the bits that are withheld when we go into the current state. The third column is the input bits that can be detected by examining the power traces. Because of the withheld bits, only two possible input strings may be detected in AS1/11. The fourth column is the bits to be committed. And the last column is the next attack state.

With the method described above, we can successfully detect the input bits in all attack states with enough power traces. We can therefore figure out all the bits in *d* and break Randomized Automaton 1.

TABLE II: ATTACK STATE TRANSITIONS

Current AS	Bits Withheld	Bits Detected	Bits Committed	Next AS
AS0	None	0	0	AS0
	None	1	1	AS1
AS1	None	0xx	0	AS0
	None	111	1	AS1/11
	None	100	100	AS0
	None	110	110	AS0
	None	101	101	AS1
AS1/11	11	111	1	AS1/11
	11	110	110	AS0

B. *Success Rate*

Assume X is a random variable that indicates the current attack state, Y is a random variable that indicates the real input bits, and Z is the event that the next detected bit is correct. $P[Z|(X=x\cap Y=y)]$ is the success rate in an attack state x when the real input string is y, where x can be AS0, AS1, or AS1/11 and y can be 000, 001, 010, $\cdots$, or 111. Table III lists $P[Z|(X=x\cap Y=y)]$ for all three attack states on 3-bit inputs. In AS0, we can always detect the first input bit correctly. So $P[Z|(X=\text{AS0}\cap Y=y)]=1$ for all y's. In AS1, if the input bit is 0, we can always detect it successfully. Therefore, $P[Z|(X=\text{AS1}\cap Y=\text{0xx})]=1$. If the first input bit is 1, then we have 4 cases: 111, 100, 101 and 101. The possibility, $P[Z|(X=\text{AS1}\cap Y=y)]$, where y = 111, 100, 101 or 101, is computed by dividing the number of paths that generate a unique sequence by the total number of possible paths y may take. If, y is the last string after all others are excluded, $P[Z|(X=x\cap Y=y)] = 1$ because y is never misidentified as other strings. For example, $P[Z|(X=\text{AS1}\cap Y=101)]=1$ because in AS1, 101 is only identified after 111, 110, and 100 are excluded. The possibilities in AS1/11 can be calculated in a similar way. The differences are that there are only two entries in AS1/11 and the unique sequence can be found in the power traces that are in either State 1 or State 11.

In Table III, we can calculate the maximal error rate of AS1 and AS1/11. In AS1, we make mistakes when the input string is 100, 110, or 111 and the unique sequences do not appear because the execution path is determined by random variables and we may not have enough power traces. With a single power trace, the probability of wrong detections is 1/2, 3/4, and 3/4 for input strings 100, 110, and 111, respectively. In AS1/11, the probability of wrong detections is 3/8 for input strings 111. As the number of power traces increases, the probability decreases exponentially. With N power traces, the overall maximum error rate in AS1 or AS1/11 can be calculated as:

$$MAX\{\frac{1}{8}\times(\frac{1}{2})^N+\frac{1}{8}\times(\frac{3}{4})^N+\frac{1}{8}\times(\frac{3}{4})^N,\frac{1}{2}\times(\frac{5}{8})^N\}$$

When N is 100, the maximum error rate is about 2^{-43}. When N is 200, it is about 2^{-85}. If the key length is 163, the overall error rate is about 2^{-77} for 200 power traces. So our attacks do not require a large number of power traces to achieve a very high success rate.

TABLE III: SUCCESS RATE WITH ONE POWER TRACE

Y \ X	AS0	AS1	AS1/11
0xx	1	1	None
100	1	1/2	None
101	1	1	None
110	1	1/4	1
111	1	1/4	3/8

C. *Related Work*

Okeya and Takagi also proposed a method to attack Randomized Automaton 1 [9]. Their method has three weaknesses [10]. First, they only focused on Randomized Automaton 1. Second, they assumed that the attacker knows the bit length of the secret value. Third, they overlook the case that one of the points in addition is the point at infinity.

Compared with Okeya and Takagi's algorithms, our attacks do not need to know the length of the secret key and are not limited to Randomized Automaton 1.

V. CONCLUSIONS

The randomized scalar point multiplication algorithms were proposed to thwart power analysis attacks. They combine several addition-subtraction chains and select a chain to follow with random variables at run-time. In this paper, we proposed an effective attack on the randomized algorithms. Our attack is based on unique sequences, the power outputs that can be generated only by specific key bits. Our attack does not require a large number of power traces to achieve a very high success rate. With 200 power traces, the error rate of a single key bit is smaller than 2^{-85}.

REFERENCES

[1] V. S. Miller, "Use of elliptic curves in cryptography," *Advances in Cryptology, CRYPTO '85*, pp. 417-426, August 1985.

[2] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of Computation*, vol. 48, no. 177, pp. 203-209, January 1987.

[3] S. Vanstone, "ECC holds key to next-gen cryptography," Available: http://www.us.design-reuse.com/articles/article7409.html, [Mar 2004].

[4] E. Oswald and M. Aigner, "Randomized addition-subtraction chains as a countermeasure against power attacks," *Proceedings of CHES 2001*, pp. 39-50, May 2001.

[5] J. S. Coron, "Resistance against differential power analysis for elliptic curve cryptosystems," *Proceedings of CHES 1999*, pp. 292-302, August 1999.

[6] A. J. Menezes, "Elliptic curve public key cryptosystems," Kluwer Academic Publishers, 1993.

[7] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," *Proceedings of CRYPTO'99*, pp. 388-397, August 1999.

[8] F. Morain, and J. Olivos, "Speeding up the computations on an elliptic curve using addition-subtraction chains," *Theoretical Informatics and Applications*, vol. 24, pp. 531-543, 1990.

[9] K. Okeya, and K. Sakurai, "On insecurity of the side channel attack countermeasure using addition-subtraction chains under distinguishability between addition and doubling," *Proceedings of 7th Australasian Conference on Information Security and Privacy, ACISP 2002*, pp. 420-435, July 2002.

[10] D. Han, N. S. Chang, S. W. Jung, Y. H. Park, C. H. Kim, and H. Ryu, "Cryptanalysis of the full version randomized addition-subtraction chains", *Proceedings of 8th Australasian Conference on Information Security and Privacy, ACISP 2003*, pp. 67 – 78, July 2003.