A new weak curve fault attack on ECIES: embedded point validation is not enough during decryption

Weiqiong Cao[1,2], Hongsong Shi[1], Hua Chen[2], Yuhang Wang[1]

[1] China Information Technology Security Evaluation Center. Building 1, yard 8, Shangdi West Road, Haidian District, Beijing 100085, China.
[2] Trusted Computing and Information Assurance Laboratory, Institute of Software, Chinese Academy of Sciences, South Fourth Street 4#, ZhongGuanCun, Beijing 100190, China.
Email: caoweqion@163.com, hsshi@163.com, chenhua@tca.iscas.ac.cn

Abstract. ECIES has been widely used in many cryptographic devices and systems to ensure the confidentiality of communication data. Hence, researching its security of implementation is essential. It is generally considered that the embedded point validation towards the input point Q during decryption is enough to resist most of the existing fault attacks and small subgroup attacks. Even many open source algorithm libraries (e.g., OpenSSL and BouncyCastle) only employ the embedded point validation to resist fault attack. However, the proposed weak curve fault attack in this paper can break this situation because it can successfully pass the embedded point validation and the validation of the scalar multiplication about the input point Q and cofactor h(i.e., $hQ \neq \mathcal{O}$). Moreover, the proposed attack does not require that the instances of ECDLP on the weak curve derived by fault injection is computationally practical which could increase the availability of fault injection. The simulations demonstrate the feasibility of our attack. Finally, we also investigate the implementations of 14 open source algorithm libraries, and there are 10 algorithm libraries which can not block our attack. Hence, we also give some suggestions about countermeasures.

Keywords: ECIES, Weak Curve Attack, Fault Attack, Small Subgroup Attack

1 Introduction

1.1 Background

ECIES [1] is an integrated encryption scheme based on elliptic curve, which is an extension of ElGamal encryption algorithm. It is mainly used for ensuring confidentiality of data or transmitting session key. It has been included in ANSI X9.63, ISO/IEC 18033-2, IEEE1363A, SECG SEC 1 and the other standards, where SM2PKE encryption also belongs to ECIES-class algorithm. ECIES has been implemented in many open source algorithm libraries such as OpenSSL, Crypto++, BouncyCastle, Miracl and so on. Its security is similar with (EC)DH, in which the small subgroups attack (SSA) [2] is one of the biggest threats.

There have been some SSAs on (EC)DH by exploiting the flaws of no point validation. For example, in ESORICS'2015 [3], Tibor et al. carried on a SSA on ECDH algorithm in TLS protocol, which can reveal the private key of the server. A SSA on DH was also introduced in [4]. However, these attacks are infeasible when being faced with standard implementation of ECC algorithm libraries, since a point validation is added in ECIES and (EC)DH so as to block the SSAs. Generally, for input point G, the *point validation* is just checking whether G is on the original curve before the scalar multiplication $Q = kG$, which is also called *embedded point validation*, while the validation of $nG \neq \mathcal{O}$ may be replaced by that of $hG \neq \mathcal{O}$ with respect to performance, where n is the order of G, $\mathcal{O}$ is infinite point and h is the cofactor. Thereby, the SSA [5] exploiting 2^m-format cofactor to reveal m-bit secret also can be blocked.

Weak curve fault attack(WCFA) is another main attack towards ECC. The first WCFA on elliptic curve cryptosystem (ECC) was proposed by Biehl et al in Crypto'2000 [6]. Suppose that some bits of the basis point G are disturbed by fault injection before the calculation of scalar multiplication $Q = kG$. Since the parameter b is not involved in the calculation of scalar multiplication, the faulty G and the derived points Q, defined by Q' and G', are not on the original curve but on a new weak curve $E'(a, b')$. Hence, $Q = kG$ is changed into $Q' = kG'$ on $E'(a, b')$. If the maximum prime factor q of the order n' of the point G' is enough small, i.e., $O\left(\sqrt{q}\right)$ steps is a feasible amount of computation, then the elliptic curve discrete logarithm problem (ECDLP) $Q' = kG'$ can be solved by the traditional methods such as pohlig Hellman [7] and Pollard Rho [8] algorithms to recover the reduced value $k \bmod n'$ of k. After that, many similar WCFAs based on faulty parameter a, modulus p and basis point G were proposed in the publications [9,10,11]. Unfortunately, the embedded point validation could block all the WCFAs above. Moreover, the WCFAs must know the result of scalar multiplication on the weak curve, while this result is not output in ECIES encryption algorithm. In sum, it is usually considered that the WCFAs above could not been used to analyze ECIES. Besides, there exist some special WCFAs which can pass the embedded point validation, e.g., the twisted curve attack proposed in FDTC'2008 [12]. Moreover, Battistello introduced a concept of common point in COSADE'2014 [13], and proposed a new WCFA targeting parameter a when parameter b is quadratic residual in Weierstrass curve $E(a, b)$. The above two attacks can pass the embedded point validation, but they still have some limitations: 1)the derived weak curve must satisfy solving the instances of ECDLP on it is computationally feasible; 2) they are all under special attack conditions, such as quadratic residual b and Montgomery ladder multiplication.

In addition, a degenerate curve attack on Edward curve was proposed in PKC'2016 [14]. It maps the addition group on the original curve into another group on singular curve by fault injection, such that the calculation of scalar multiplication on the new singular curve is simple and the scalar can be deduced directly. Then Takahashi et al. combined this attack and fault attack to analyze ECDSA and SM2 encryption in Europe S&P'2019 [15]. The combined attack assumes that the instructions during the decompression of basis point in SECG can be skipped by fault injection. Thereby, a similar singular curve is constructed. However, this attack requires high-accuracy fault injection and can not attack ECIES decryption with point validation.

1.2 Our contributions

In this paper, we exploit the advantages of the weak fault curve attack and small subgroups attack, and propose a new weak curve fault attack towards standard ECIES (based on Weierstrass equation). The attack assumes that fault injection causes a continuous bit block of parameter a disturbed randomly. Then, some weak curves can be constructed, by which some reduced values of secret can be deduced by small subgroups attacks. Compared with the existing weak fault curve attacks, our attack is more feasible and practical. The detailed advantages are summarized as follows.

- Our attack is effective for all the standard ECIES algorithms with embedded point validation. The embedded point validation and $hG \neq \mathcal{O}$ (where h is cofactor, G is the input point of scalar multiplication and $\mathcal{O}$ is infinite point) during ECIES decryption can not block our attack, since the input point constructed by the adversary for decryption is just on the weak curve.
- Compared with the existing weak curve faults, our attack reduces the requirement of fault injection extremely. it is unnecessary for our attack that the instances of ECDLP on the constructed weak curves are solvable practically. As long as there are some small factors in the order of the weak curves, then the weak curves can be used for our attack.
- Our attack can pose a pratical threat to many widely used open source algorithm libraries with embedded point validation, such as OpenSSL, BouncyCastle, Botan, mbedtls and so on. It demonstrates that the embedded point validation is not enough for resisting fault attack. Moreover, we also give some effective countermeasures.

The remainder of this paper is organized as follows: Section 2 reviews the preliminaries that required to describe our approach. Section 3 describes the concrete weak curve fault attack. Section 4 shows the experimental facets of the validity. Section 5 summarizes the available countermeasures, investigates the open source algorithm libraries being vulnerable to our attack and gives some countermeasure suggestions. Finally, a conclusion is given in Section 6.

2 Preliminaries

In view of the universality of prime field $\mathbb{F}_p$ (p is a prime number), here we just introduce ECIES on Weierstrass elliptic curve $E(a, b)$ in $\mathbb{F}_p$.

2.1 Elliptic curve in $\mathbb{F}_p$

Weierstrass equation of elliptic curves in $\mathbb{F}_p$ is denoted by

$$E(a,b) : y^2 = x^3 + ax + b \mod p,$$

where parameters $a, b \in \mathbb{F}_p$ meet $4a^3 + 27b^2 \neq 0$.

The additive group of the rational points on elliptic curve $E(a, b)$ is defined as

$$E(\mathbb{F}_p) = \left\{(x,y) \in \mathbb{F}_p \times \mathbb{F}_p | y^2 = x^3 + ax + b \bmod p\right\} \cup \{\mathcal{O}\},$$

where $\mathcal{O}$ is the infinite point. The number of the points in $E(\mathbb{F}_p)$, also called the order of $E(a, b)$ in $\mathbb{F}_p$, is denoted as $\#E(\mathbb{F}_p)$ which is generally solved by SEA algorithm [16].

Let G be a point in $E(\mathbb{F}_p)$, and then the group $\langle G\rangle$ with generator G is the additive subgroup of $E(\mathbb{F}_p)$. If the number of the points in $\langle G\rangle$ is n, then n is the order of point G satisfying $nG = \mathcal{O}$. $\mathcal{O}$ is the identity element of $\langle G\rangle$. The inverse element for any point $P = (x, y) \in \langle G\rangle$ is $-P = (x, -y) \in \langle G\rangle$. For any integer $k \in \mathbb{Z}_n$ and point G, the scalar multiplication $kG = G + G + \ldots + G$ (k times) is calculated using the following point doubling and addition operations.

Point Addition

If $P = (x_1, y_1) \in \langle G\rangle$, $Q = (x_2, y_2) \in \langle G\rangle$, and $P \neq \pm Q$, then $(x_3, y_3) = P+Q$ satisfies

$$\begin{array}{l} x_3 = \lambda^2 - x_2 - x_1 \\ y_3 = \lambda(x_1 - x_3) - y_1 \end{array}, \text{ where } \lambda = \frac{y_2-y_1}{x_2-x_1}.$$

Point doubling

If $P = (x_1, y_1) \in \langle G\rangle$ and $P \neq -P$, then $(x_3, y_3) = 2P$ satisfies

$$\begin{array}{l} x_3 = \lambda^2 - 2x_1 \\ y_3 = \lambda(x_1 - x_3) - y_1 \end{array}, \text{ where } \lambda = \frac{3x_1^2+a}{2y_1}.$$

From the rules above, parameter b does not participate in the calculation of point doubling and addition, which is also the key point of most of the existing fault attacks.

2.2 ECIES encryption algorithm

We recap briefly ECIES encryption algorithm as introduced in [17]. The whole encryption process is shown in Algorithm 1. User **A** encrypts the plain m with the public key P_B of User **B** and outputs the ciphertext (Q, C, t). h is the cofactor satisfying $hn = \#E(\mathbb{F}_p)$. $KDF(.)$ is the key derivation function based on hash function. $ENC_{k_1}(.)$ and $DEC_{k_1}(.)$ are generally the functions of symmetric algorithms, and can also be simple XOR operations, where k_1 is the secret key of ENC and DEC. $MAC_{k_2}(.)$ is the message authentication code with key k_2, such as HMAC. There are two scalar multiplication operations during encryption, among which the result Q of kG is transmitted to **B** as a part of ciphertext.

B uses its own private key d_B to decrypt the ciphertext sent from **A**. As shown in algorithm 2, in order to resist the small subgroup attack, the validation of the input point Q before the scalar multiplication (Step 1) is required.

Generally, as in [17], not the *point validation* but the *embedded point validation* without Step 4 (see Algorithm 3 for details) is employed to check the input point of scalar multiplication. Our attack just captures this advantage to analyze decryption.

Algorithm 1 Encryption of ECIES

```
Require: The definition of a specific elliptic curve E$(a, b)$ in $\mathbb{F}_p$, a base point $G$ of the curve with order $n$, plaintext $m$, B's public key $P_B$.
Ensure: Ciphertext $(Q, C, t)$.
1: Generate $k$ with random generator;
2: Calculate $Q = kG$ and $S = hkP_B$, where $Q = (x_Q, y_Q)$ and $S = (x_S, y_S)$;
3: If $S = \mathcal{O}$ then goto step 1;
4: Calculate $(k_1, k_2) = KDF(x_S, Q)$, $C = ENC_{k_1}(m)$ and $t = MAC_{k_2}(C)$;
5: return $(Q, C, t)$
```

Algorithm 2 Decryption of ECIES

```
Require: The definition of a specific elliptic curve E$(a, b)$ in $\mathbb{F}_p$, basis point $G$ of the curve with order $n$, ciphertext $(Q, C, t)$ and B's private key $d_B$.
Ensure: Plaintext $m$.
1: Check point $Q$ with embedded point validation (Algorithm 3). If the validation is failed, return rejecting this ciphertext;
2: Calculate $S = (hd_B)Q$, where $S = (x_S, y_S)$. If $S = \mathcal{O}$, return rejecting this ciphertext;
3: Calculate $(k_1, k_2) = KDF(x_S, Q)$ and $t' = MAC_{k_2}(C)$. If $t' = t$, return rejecting this ciphertext;
4: Calculate $m = DEC_{k_1}(C)$;
5: return $(m)$
```

2.3 Definitions of weak curve

To better describe our approach, we introduce the following definitions based on the foundation in Section 2.1.

Definition 1. ***Elliptic curve discrete logarithm problem (ECDLP)****: given $G \in E(\mathbb{F}_p)$ with order n and an element $Q \in \langle G \rangle$, find the value $k \in \mathbb{Z}_n$ such that $Q = kG$.*

To our knowledge, the combination of Pohlig-Hellman algorithm [7] and Pollard's rho algorithm [8] is the most common approach in classical computer for solving ECDLP in arbitrary elliptic curves. Thereby, we have the following definitions.

Algorithm 3 (Embedded) Point Validation

```
Require: The definition of a specific elliptic curve $\mathrm{E}(a,b)$ in $\mathbb{F}_p$, order $n$ and input
    point $Q$.
Ensure: Whether $Q$ is valid or not.
1: Check that $Q = \mathcal{O}$;
2: Check that the $x_Q$ and $y_Q$ ($x$ and $y$ coordinates) of $Q$ belong to $\mathbb{F}_p$;
3: Check that $Q$ is on the curve $\mathrm{E}(a,b)$;
4: Check that $nQ = \mathcal{O}$ (This step would be omitted in embedded point validation
    due to efficiency);
5: If any checking is failed, then return (Invalid); Otherwise, return (Valid).
```

Definition 2. *It is assumed that n is the order of weak curve $E(a,b)$ and meets the factorization $n = \prod_{i=1}^{u} q_i^{e_i}$, where $q_i \in \mathbb{N}$ is a prime factor of n, $e_i > 0$ denotes the degree of q_i in the factorization and $q_i < q_j$ for $1 \leq i < j \leq u$. If the biggest prime factor q_u of order n meets $O(\sqrt{q_u})$ operations is feasible in computation, i.e., $O(\sqrt{q_u})$ is not bigger than a predefined constant* PRAC_COMP*, an ECDLP instance, the order n on $E(a,b)$ is called* practically solvable*. Correspondingly, $E(a,b)$ is called as practically solvable weak curve.*

In this paper, we set PRAC_COMP$= 2^{64}$ group operations by considering currently computing power of classical computers, which can be redefined with the development of computer technology.

3 Weak curve attack on ECIES

In this section, we first introduce the attack scenarios and fault model. Then, we describe the concrete weak curve attack on ECIES with 4 steps, where step 2 and step 3 are the most important two steps and consist of two algorithms.

3.1 Attack Scenarios

Generally, for weak curve fault attack, fault injection mainly targets the physical objects including RAM, EEPROM or CPU register/memory so as to perturb some bits of base point G, parameter a or modulus p. There are usually three fault types, including temporary fault (i.e., the fault is updated after every invocation), semi-permanent fault (i.e., the fault is kept unless the device restores the correct one) and permanent fault (i.e., the fault is always kept once induced). In this paper, we consider the fault type is permanent fault or semi-permanent fault. In order to obtain the private key d_B of **B**, the adversary in our attack impersonates **A** to send ciphertext to **B**. Moreover, it is assumed that parameter a is involved in the calculation of scalar multiplication in decryption and there is a continuous l-bit block of a disturbed randomly by fault injection. Hence, a is changed into a' which has l bits different from a. The faulty a' will keep unchanged during the attack process until the device reset or reinducing fault.

Consequently, the embedded point validation during decryption (step 1 in Algorithm 2) is not on the curve $E(a, b)$ but on $E(a', b)$, and the scalar multiplication $S = (hd_B)Q$ during decryption (step 2 in Algorithm 2) is on $E(a', b)$: $x^3 + a'x + b \bmod p$.

3.2 Proposed fault attack on weak curves

Before the attack, some notations should be defined firstly.

T: an integer which is set to be not greater than PRAC_COMP
$SET(T)$: the set of the small factors of n' (derived from valid a') which are not greater than 1
$Red(d_B)$: the set of the reduced values of d_B
N: the number of $SET(T)$ (which is also the number of $Red(d_B)$)
E_j (& R_j): the j-th element of $SET(T)$ (&$Red(d_B)$)

The sets $SET(T)$ and $Red(d_B)$ are initialized to be two empty sets, i.e., $SET(T) = \emptyset$, $Red(d_B) = \emptyset$ and $N = 0$.

Step 1. Fault injection: Perturb a to be changed into a'

It is assumed that fault injection is induced to perturb l-bit continuous segment of a randomly before decryption such that a is changed into a'. As mentioned as above, a' can be represented with $a' = a \oplus \beta 2^\lambda$, where β is a l-bit random number, $\lambda \in \mathbb{N}$ belongs to $[0, l_a - l](l_a$ is the bit length of $a)$, and $\lambda + l \leq l_a$. Moreover, it is usually considered that l is valued from 8, 16, 24, 32, 64 in view of the bit width of processor, which still meets 2^l is not greater than PRAC_COMP.

If the faulty a' is unknown, we utilize the following **Step 2** (i.e., Algorithm ALG-GUESS-PARA) to reveal the faulty a'. Otherwise, go to **Step 3** (i.e., Algorithm ALG-OBTAIN-PRIKEYINFO).

Step 2. Algorithm ALG-GUESS-PARA: Guess and determine a'

Step 2-1. Guess all the possible values a'_j of $a'(= a \oplus \beta 2^\lambda)$, i.e., guess l-bit $\beta \in \mathbb{Z}_{2^l}$ and $\lambda \in [0, l_a - l]$. The number of the possible values of a' is $(l_a - l + 1)2^l$ at most and $j \in \{0, 1, \ldots, (l_a - l + 1)2^l\}$.
Step 2-2. Based on the guessed value a'_j, construct a curve $E(a'_j, b)$, and select a arbitrary point Q_j on it. Forge the ciphertext (Q_j, C_j, t_j) (See Algorithm 1), where C_j and t_j is not calcuted by Q_j but randomly selected, i.e., C_j and t_j could definitely be rejected by **B**. Then, send the ciphertext to inquiry **B**.
Step 2-3. If the ciphertext is rejected by **B** just after the embedded point validation (i.e., Q_j is the not on the curve $E(a', b)$), then the corresponding guessed a'_j is not the valid value of a'. Then, go to **Step 2-1**. Otherwise, if **B** rejects the ciphertext after the scalar multiplication $(hd_B)Q_j$ (i.e., Q_j has passed the embedded point validation and is on $E(a', b)$), then a'_j is just the faulty parameter a'. Finally, end the algorithm.

Note that it is straightforward to tell when **B** rejects the ciphertext in **Step 2-3**. The valid point Q_j (which is on $E(a', b)$) can pass the point validation in decryption and be involved in the calculation of $(hd_B)Q_j$ which *is the main time (or power) consumption in decryption*, while the invalid point on the incorrect guessed $E(a'_j, b)$ would be rejected during the embedded point validation. Hence, the time lengths (or the feature of power consumption) of the two rejections are obviously different, from which we can distinguish the correct guessed a'_j. Moreover, the above algorithm of determining a' will cost $O((l_a - l + 1)2^l)$ computational complexity.

Then, with the known faulty a', we can obtain some reduced values about the private key of **B** by the following algorithm.

Step 3. Algorithm ALG-OBTAIN-PRIKEYINFO: Obtain the reduce values of private key d_B

Step 3-1. Calculate the order n' of a' with SEA algorithm [16], and factorize n' into $n' = \prod_{i=1}^{u} q_i^{e_i}$ (which is feasible when the bit length of n' is less than 512), where $q_i \in \mathbb{N}$ is the prime factor and $e_i \in \mathbb{N}(e_i > 0)$ is the degree of $q_i (q_i < q_j$ for $i < j)$.

Step 3-2. For each factor $q_i^{e_i} (1 \le i \le u)$ of n', if $q_i^{e_i}$ satisfies

$$q_i^{e_i} < T \text{ and } gcd(q_i, E) = 1, \forall E \in SET(T),$$

then add $q_i^{e_i}$ into $SET(T)$ (i.e., $SET(T)$ is updated); if $q_i^{e_i}$ satisfies

$$q_i^{e_i} < T \ , \ E | q_i^{e_i} \text{ and } q_i^{e_i} > E, \exists E \in SET(T),$$

then delete E from $SET(T)$ and add $q_i^{e_i}$ into $SET(T)$ (i.e., $SET(T)$ is updated); otherwise, $SET(T)$ is not updated.

Step 3-3. If $SET(T)$ is not updated (i.e., the faulty a is a invalid one), go to **Step 1** and regenerate a new faulty a'.

Step 3-4. For every factor $q_i^{e_i}$ above which is added into $SET(T)$, being assumed to be E_j (i.e., $E_j = q_i^{e_i}$), select a point Q with order E_j on $E(a', b)$. Guess all the reduced values $\tilde{d}_B \in [0, \frac{E_j}{\gcd(E_j, h)} - 1] \cap \mathbb{Z}$ of d_B, by which forge the corresponding ciphtertext with randomly selected message m (according to Algorithm 1)

$$\begin{array}{l} \tilde{S}(x_{\tilde{S}}, y_{\tilde{S}}) = h\tilde{d}_B Q, \\ (\tilde{k}_1, \tilde{k}_2) = KDF(x_{\tilde{S}}, Q), \\ \tilde{C} = ENC_{\tilde{k}_1}(m), \\ \text{and } \tilde{t} = MAC_{\tilde{k}_2}(\tilde{C}). \end{array}.$$

Finally, send the forged ciphtertext $(Q, \tilde{C}, \tilde{t})$ to inquiry user **B**. If **B** accepts the ciphtertext and outputs the plaintext, then the guessed $\tilde{d}_B$ is the correct reduced value R_j of d_B, where $R_j = d_B \bmod \frac{E_j}{\gcd(E_j, h)}$. Then, add R_j into $Red(d_B)$. Otherwise, continue to guess and inquiry.

Note that, except for being required to have small factors $q_i^{e_i}$, the orders n' of the valid a' is not required to be practically solvable. Hence, most of faulty a's based on random fault model are valid for our attack. Moreover, for each guessed value R_j and order E_j $(j = 1, ..., N)$, since all the E_js are less than T, the maximum computational complexity for the above guess-determine approach is $O(NT)$. Hence, if only T is set to be less than PRAC_COMP (N is usually a small integer), the approach is feasible.

Step 4. Recover the private key d_B

If the N elements $E_j(j = 1, ..., N)$ of $SET(T)$ satisfy

$$\prod_{i=1}^{N} \frac{E_i}{\gcd(E_i, h)} < n,$$

then go to **Step 1** to generate a new faulty a'. Meanwhile, repeat **Algorithm** ALG-GUESS-PARA and **Algorithm** ALG-OBTAIN-PRIKEYINFO to add some new elements into $SET(T)$ and $Red(d_B)$. Otherwise, using Chinese remainder theorem(CRT), calculate

$$\begin{cases} R_1 = d_B \bmod \frac{E_1}{\gcd(E_1,h)} \\ \vdots \\ R_N = d_B \bmod \frac{E_N}{\gcd(E_N,h)} \end{cases},$$

and recover the private d_B.

4 Experimental analysis

In this section, we demonstrate the feasibility of our attack by simulations. The emphasis is on checking Algorithm ALG-OBTAIN-PRIINFO and the recovery of d_B. Therefore, we do not carry on the practical fault injection in experiments.

We perform the experiments in a general computer with 8-core CPU 3.4GHz, 8G memory and Windows7 OS. We employ the SEA algorithm algorithm and the CRT algorithm implemented in (C/C++) miracl library to calculate the weak curve order n' based on faulty a' and solve the private key, respectively.

We choose two 256-bit curves over finite prime field $\mathbb{F}_p$ to analyze in experiments respectively, including NIST P-256 curve [18] (hereafter called P-256) and the curve recommended in SM2 digital signature algorithm (hereafter called SM2-curve which still can be employed as the curve of ECDSA) [19]. For each curve, there are two simulated faults in experiments, the single-bit flipped fault and 16-bit random fault respectively. The single-bit flipped fault assumes to flip a bit-by-bit. Then there are 256 cases in total. The 16-bit random fault assumes that there are 16 bits of a disturbed randomly, i.e., $a' = a \oplus (\beta 2^{\lambda})$, where $\beta \in \{0, 1\}^{16}$ is a 16-bit random integer, and $\lambda \in [0, 240] \cap \mathbb{Z}$ is also a small random integer. The fault is also simulated for 256 times (where each simulation employs randomly different β and λ). To sum up, we do four types of different simulations, and each one is done for 256 times.

For each type of fault simulations, there are 256 different a's. For $i = 0, ..., 15$, pick the $16i$-th a' (which is known) as the faulty parameter of the first fault injection in turn (i.e., run Step 1 and Step 2 of Section 3.2). Then, run the Algorithm ALG-OBTAIN-PRIKEYINFO (i.e., Step 3) and update the sets $SET(T)$ and $Red(d_B)$. If all the elements of $SET(T)$ satisfy $\prod_{j=1}^{N} \frac{E_j}{\gcd(E_j,h)} < n$ (i.e., Step 4), then add $(16i+1)\%256$(or $(16i+2)\%256, ..., (16i+r_i-1)\%256$)-th a' one-by-one as the parameters of $2(, 3, ..., r_i)$-th fault injection (Steps 1 and 2) until the attack terminates and outputs the private key d_B (Steps 3 and 4), where r_i is the required number of fault injection to recover d_B when $i = 0, ..., 15$ in turn. Finally, check $d_B G = P_B$.

Finally, for the four types of fault simulations and analysis, we summrize the range of r_i for $i = 0, ..., 15$ required in our attack when T is equal to $2^8, 2^{16}, 2^{24}, 2^{32}$ and 2^{40} respectively. As shown in Table 1, the required number r_i of fault injection in the 16-time attacks (corresponding to 16 groups of r_is) is 219 at most and 174 at least when T is 2^8 for single-bit flipped fault of SM2-curve, which is still practically feasible for fault injection. Moreover, the computational complicity of determining d_A when $T = 2^8$ is lowest than the other cases. When $T = 40$, there even exist 3 times of fault injection to recover the private key successfully. Moreover, the required r_i for the four types of fault simulations are approximately close. Thereby, our attack could be applied to a number of ECIES algorithms only with embedded point validation based on standard curves.

Table 1. The range of the needed number r_i of fault injection for $i = 0, ..., 15$

Fault type	$T = 2^8$	$T = 2^{16}$	$T = 2^{24}$	$T = 2^{32}$	$T = 2^{40}$
	Range of r_i				
single-bit flipped fault(P-256)	$126 - 174$	$15 - 27$	$8 - 18$	$7 - 16$	$5 - 12$
16-bits random fault(P-256)	$121 - 176$	$16 - 33$	$9 - 26$	$7 - 19$	$4 - 12$
single-bit flipped fault(SM2-curve)	$148 - 219$	$14 - 30$	$9 - 20$	$4 - 14$	$3 - 12$
16-bits random fault(SM2-curve)	$98 - 176$	$14 - 31$	$6 - 21$	$6 - 14$	$4 - 12$

5 Countermeasure discussion and practical threat

As mentioned above, the complete point validation with $nQ = \mathcal{O}$ towards input point Q is the most intuitive countermeasure to block our attack. However, introducing the calculation of scalar multiplication nQ almost doubles the performance time of each ECIES decryption, since the scalar multiplications nQ and $(hd_B)Q$ take the main time consumption. This is also the main reason why most of algorithm libraries do not employ the complete point validation.

In addition, the general scalar masking resisting side channel attack (SCA) also could block our attack. For example, the scalar d_B is masked with a small random number γ to become a new value d'_B, i.e., $d'_B = d_B + \gamma n$. After that, d'_B as the new scalar participates in the following calculation of scalar multiplication $(hd'_B)Q$. For this case, since Q is not on the original curve and $nQ \neq \mathcal{O}$, it is necessary to guess the reduced value $\tilde{\gamma} \in [0, d-1]$ of γ in addition to the reduce value of d_B in our attack, where $d = \frac{E_j}{\gcd(E_j, h)}$. Then, calculate $\tilde{S}(x_{\tilde{S}}, y_{\tilde{S}}) = h(\tilde{d}_B + \tilde{\gamma} n)Q$ (see Step 3-4 of Algorithm ALG-OBTAIN-PRIKEYINFO in Section Section 3.2). However, γ is changed along with each decryption and thereby is determined with probability $\frac{1}{\gamma \bmod d \times d_B \bmod d}$. If the selected d is slightly large, the probability of determining $\gamma \bmod d$ is very low, which would block our attack. Similarly, the scalar masking $d'_B = d_B + \gamma n$ also introduces additional time consumption due to the increasing bit length of d'_B. Therefore, in real implementation, many open source algorithm libraries do not employ the scalar masking and replace it with other countermeasures, such as randomizing the coordinates of the point, $d'_B = d_B + 2n$ (or $d'_B = d_B + n$) (making constant time of the scalar multiplication) and so on. However, for these countermeasures, our attack is immune.

To sum up, although the two countermeasures above (complete point validation and scalar masking) can block our attack, there still exist many security bugs in practical implementation. In view of this, we summarize the potential security risks of many open source libraries under our attack.

5.1 Practical threat to open source algorithm libraries

We investigated the current commonly used open source algorithm libraries such as OpenSSL, CYPTO++, BouncyCastle and so on, and check whether they have the point (or embed point) validation and scalar masking. Finally, we found many libraries do not or only implement the embedded point validation which is still vulnerable to our attacks. We summarized the security risks of the algorithm libraries in Table 2. Let EPV denote the embedded point validation (see Algorithm 3), and PV denote the complete point validation (with the additional scalar multiplication of $nQ = \mathcal{O}$) which could block our attack. On the one hand, there are only 4 algorithm libraries (The last 4 lines) implementing PV during scalar multiplication in Table 2. Moreover, only CYPTO++ and libtomcrypt set PV as the default validation, and can block our attack. For MIRACL, although there is PV implemented in it, EPV is employed during the real implementation of ECIES. In addition, employing PV in BouncyCastle depends on the fact that the cofactor h is not equal to 1, which is uncertain whether it will block our attack. On the other hand, except the algorithm library Botan 2.17.3, the other libraries do not employ the countermeasure of scalar masking (which can block our attack). Moreover, in Botan 2.17.3, the scalar masking is implemented only when there is random number generator (RNG). Otherwise, d_B is masked with $d'_B = d_B + 2n$ (or $d'_B = d_B + n$). Hence, its vulnerability on resisting our attack depends on the concrete resource of the algorithm library. In addition, Tinycrypt, OpenSSL and GmSSL also employ the scalar countermeasure of $d'_B = d_B + 2n$

(or $d'_B = d_B + n$), but it is ineffective to block our attack. In a word, our attack could cause real threat to most of current common open source algorithm libraries(10 out of 14).

Table 2. The range of the needed number r_i of fault injection for $i = 0, ..., 15$

Item	Algorithm libraries	Method of validation	Scalar masking	Whether block our attack or not
1	Tinycrypt	none	✗	No
2	Wolfssl	none	✗	No
3	matrixssl	none	✗	No
4	OpenSSL	EPV	✗	No
5	OpenSSL-andrio	EPV	✗	No
6	Botan 2.17.3	EPV	✓ (if existing RNG)	Uncertain
7	GmSSL	EPV	✗	No
8	cryptlib 3.4.6	EPV	✗	No
9	libgcrypt-1.9.2	EPV	✗	No
10	mbedtls-2.25	EPV	✗	No
1^2	MIRACL	EPV(default) and PV	✗	No
11	BouncyCastle 1.8.9	EPV($h = 1$) and PV	✗	Uncertain
13	CYPTO++ 8.4.0	EPV and PV(default)	✗	Yes
14	libtomcrypt	PV	✗	Yes

5.2 Our suggestion on countermeasure

To balance the implementation efficiency and the security resisting our attack, we recommend adding an embedded point validation of the base point G during the ECIES decryption (see Algorithm 2) after the embedded point validation of the input point Q (Step 1 in Algorithm 2). That is, before the scalar multiplication $(hd_B)Q$ (Step 2 in Algorithm 2), check whether G is on the curve with parameters a and b. Since parameter a has been replaced by a', G is not on the curve $E(a', b)$ definitely and the ciphertext is naturally rejected. Note that this countermeasure is just valid when there is only one fault (i.e., faulty parameter a). For more faults, e.g., the faulty instruction flow that all the embedded point validations are skipped by fault injection in each decryption, we recommend that the scalar masking with a small random number should be employed.

6 Conclusion

We combines the advantages of fault attack and small subgroup attack to propose a new weak curve fault attack on ECIES with embedded point validation. Based on the fault of parameter a, i.e., faulty a', our attack can pass the embedded point validation and recover the private key in ECIES decryption successfully. Moreover, the order n' of the weak curve generated by faulty a' is not required to be practically solvable, which can increasing the availability of fault injection. The experiments demonstrate that our attack is practically feasible and the investigation shows that our attack could pose real threat to many open source algorithm libraries. Therefore, we also give some suggestion about the countermeasures which can block our attack.

References

1. Brown, D.: Standards for efficient cryptography, sec 1: elliptic curve cryptography. Released Standard Version **1** (2009)
2. Zuccherato, R.: Methods for avoiding the small-subgroup attacks on the diffie-hellman key agreement method for s/mime. Network Working Group, RFC2785. The Internet Society (2000)
3. Jager, T., Schwenk, J., Somorovsky, J.: Practical invalid curve attacks on tls-ecdh. In: European Symposium on research in computer security, Springer (2015) 407–425
4. Valenta, L., Adrian, D., Sanso, A., Cohney, S., Fried, J., Hastings, M., Halderman, J.A., Heninger, N.: Measuring small subgroup attacks against diffie-hellman. In: NDSS. (2017)
5. Wroński, M.: Combined small subgroups and side-channel attack on elliptic curves with cofactor divisible by 2m. International Journal of Electronics and Telecommunications **65** (2019)
6. Biehl, I., Meyer, B., Müller, V.: Differential Fault Attacks on Elliptic Curve Cryptosystems. In: Advances in Cryptology-CRYPTO 2000, Springer (2000) 131–146
7. Pohlig, S., Hellman, M.: An improved algorithm for computing logarithms over gf (p) and its cryptographic significance (corresp.). IEEE Transactions on information Theory **24**(1) (1978) 106–110
8. Van Oorschot, P.C., Wiener, M.J.: Parallel collision search with cryptanalytic applications. Journal of cryptology **12**(1) (1999) 1–28
9. Ciet, M., Joye, M.: Elliptic curve cryptosystems in the presence of permanent and transient faults. Designs Codes Cryptography **36**(1) (2005) 33–43
10. Kim, T., Tibouchi, M.: Bit-flip faults on elliptic curve base fields, revisited. In: International Conference on Applied Cryptography and Network Security, Springer (2014) 163–180
11. Serraj, T., Ismaili, M.C., Azizi, A.: On the security of some elliptic curve standards in the presence of random fault analysis attacks. In: 2017 International Conference on Wireless Technologies, Embedded and Intelligent Systems (WITS), IEEE (2017) 1–5
12. Fouque, P.A., Lercier, R., Réal, D., Valette, F.: Fault attack on elliptic curve montgomery ladder implementation. In: 2008 5th Workshop on Fault Diagnosis and Tolerance in Cryptography, IEEE (2008) 92–98

13. Battistello, A.: Common points on elliptic curves: The achilles heel of fault attack countermeasures. In: International Workshop on Constructive Side-Channel Analysis and Secure Design, Springer (2014) 69–81
14. Neves, S., Tibouchi, M.: Degenerate curve attacks. In: Public-Key Cryptography–PKC 2016, Springer (2016) 19–35
15. Takahashi, A., Tibouchi, M.: Degenerate fault attacks on elliptic curve parameters in openssl. In: 2019 IEEE European Symposium on Security and Privacy (EuroS&P), IEEE (2019) 371–386
16. Elkies, N.D., et al.: Elliptic and modular curves over finite fields and related computational issues. AMS IP STUDIES IN ADVANCED MATHEMATICS **7** (1998) 21–76
17. Hankerson, D., Menezes, A.J., Vanstone, S.: Guide to elliptic curve cryptography. Springer (2004)
18. Certicom Research: Recommended Elliptic Curve Domain Parameters Standards for Efficient Cryptography (SEC) 2. https://www.iso.org/standard/76382.html (2000)
19. International Standard ISO/IEC 14888-3:2006(E): IT Security techniques Digital signatures with appendix Part 3: Discrete logarithm based mechanisms. https://www.iso.org/standard/76382.html (2018)