Differential Fault Attack on Montgomery Ladder and in the Presence of Scalar Randomization

Andy Russon

Differential Fault Attack on Montgomery Ladder and in the Presence of Scalar Randomization

Andy Russon[1,2]

[1] Orange, Applied Crypto Group, Cesson-Sévigné, France
[2] Univ Rennes, CNRS, IRMAR - UMR 6625, 35000 Rennes, France
andy.russon@univ-rennes1.fr

Abstract. Differential fault attacks are powerful techniques to break a cryptographic primitive, where the adversary disrupts the execution of a calculation to find a secret key. Those attacks have been applied in Elliptic Curve Cryptography under various types of faults, and there exists several protection mechanisms to prevent them.

In this paper, we present a new differential fault attack on the Montgomery ladder algorithm for scalar multiplication. We further present that such attacks can be applied when specific point additions formulas are used and when different scalar blinding techniques to randomize the computation are present.

Keywords: Differential fault attack · Elliptic curve · Montgomery ladder · Scalar blinding · Scalar splitting

1 Introduction

Differential Fault Analysis (DFA) was first introduced on block ciphers [4] and RSA [7]. In these attacks, a fault is induced and modifies the behavior of the execution resulting in an erroneous output. The effects of the fault on the output are compared with the correct one to compromise the full secret key.

The efficiency of Elliptic Curve Cryptography (ECC) makes it popular for embedded devices due mainly to the small parameter size for high-security level, thus it is necessary to protect against physical attacks such as fault attacks. In this paper, we are interested in several unexplored paths for DFA on the scalar multiplication which is the main operation of ECC.

The first main contribution is a new attack on the Montgomery ladder algorithm [22]. Its most sensitive part as implemented in cryptographic libraries is a conditional swap, and we extend the analysis of DFA when a fault affects this operation. In particular, we look at the use of specific point addition formulas that do not use all point coordinates. Those are specific to the Montgomery ladder algorithm and often used in libraries outside of the classical formulas. Furthermore, we show that point validation or loop invariant verification are not sound measures to protect against our attack.

Our second main contribution is to show that scalar blinding methods do not prevent DFA if the randomizer is too small. Those are the first Coron countermeasure [13] that adds a random multiple of the group order to blind the secret scalar, and the others are methods that separate in several shares the scalar with a multiplicative or Euclidean splitting [12,38]. We consider the attack in the context of the ECDSA signature scheme, as it is well suited for DFA and can be exploited for key recovery using lattice techniques [25]. Finally, our attack was experimented on several simulations.

The paper is organized as follows. In Sect. 2 we introduce notations on elliptic curves and ECDSA. Then, Sect. 3 describes our DFA attack on the Montgomery ladder algorithm, followed in Sect. 4 on how the scalar blinding and scalar splitting methods can also be attacked. We give in Sect. 5 a practical evaluation of the implementation of the ladder algorithm in cryptographic libraries, how the attack can be achieved under the skip instruction fault model or with a random fault in a register, and results from several simulations. Finally, we discuss in Sect. 6 past proposed countermeasures to protect against DFA and the limitations of some against our attack, and we conclude in Sect. 7. The construction of the lattices adapted for each case considered in the paper is presented in Appendix A.

1.1 Related Works

The first report of DFA with elliptic curves was presented in [3]. The target of the fault injection is a point coordinate during a scalar multiplication resulting in a point that does not belong to the original elliptic curve. The algorithm is run backward with the correct and erroneous outputs by making guesses on the bits of the secret scalar processed after the fault was made. The comparison with the correct value is used to check which guess is the correct one. This attack makes the points leave the curve, and a classical countermeasure is to validate them using the curve equation before releasing an output.

Another DFA was proposed in [6], with the advantage that a point validation does not detect the fault. Indeed this *sign-change fault* attack only modifies the sign of a point, so it still satisfies the curve equation. An example is given to realize such an effect with a fault during the calculation of the NAF representation of the secret scalar, and the paper claims that it could be adapted to the Montgomery ladder algorithm in the case the y-coordinate is used.

In the same line of work, DFA where point validation cannot detect the fault were presented in [32] and [33] on the Montgomery ladder algorithm. The faults considered are a skip of one or several operations of the algorithm such that one bit is not processed in the first paper, or with a skip of one multiplication or one squaring when used with RSA in the second paper (but compatible with elliptic curves if one replaces the operations with point doublings and point additions). Then, it is possible to recover the bits of the scalar processed after or before the fault occurred.

Our attack is akin to the previous DFA on the Montgomery ladder algorithm, where the fault does not make the points leave the elliptic curve. But it shares

similarities to the *sign-change fault* attack, as the goal is to change the sign of the implicit loop invariant of the algorithm.

There are other recent fault attacks on ECC, but those are either a DFA against a wNAF algorithm for scalar multiplication [11], or target specifically deterministic signature scheme such as the determinist variant of ECDSA or EdDSA [1,26,29,30] that cover well the subject.

2 Preliminaries

In this section, we introduce notations of elliptic curves, followed by a description of ECDSA and why it is useful for a DFA attack.

2.1 Elliptic Curves over Prime Fields

An elliptic curve E defined over a field $\mathbf{F}_p$ with p a prime greater than 5 is the set of points $(x, y) \in (\mathbf{F}_p)^2$ that satisfy an equation of the form

$$y^2 = x^3 + Ax + B, \quad A, B \in \mathbf{F}_p, \tag{1}$$

with $\Delta = 4A^3 + 27B^2 \neq 0$, and an additional point $\mathcal{O}$, alongside an operation that makes the curve an abelian group. This operation is the point addition, where the identity is $\mathcal{O}$, and the inverse of a point $P = (x_P, y_P)$ is $-P = (x_P, -y_P)$.

For an integer k, the operation called scalar multiplication is the repeated addition of a point P that appears k times and is noted $[k]P$. For all points P, there exists a smallest positive integer k such that $[k]P = \mathcal{O}$ and is called the order of the point.

Given Q a point in the subgroup of prime order q generated by P, then there exists an integer k such that $Q = [k]P$ and is called the discrete logarithm of Q in base P. The security of ECC is based upon the hardness of finding the discrete logarithm, and the best algorithms are Baby Step-Giant Steps (BSGS) [35] and Pollard's *rho* algorithms with complexity $O(\sqrt{q})$. In the case k is known to lie in a relatively small interval $[a, b]$, then it can be found in complexity $O(\sqrt{b-a})$ with BSGS or Pollard's kangaroo algorithm [27].

2.2 ECDSA

This is an elliptic curve-based signature scheme [24]. Its domain parameters are an elliptic curve E and a base point P of prime order q that belongs to the curve.

Given a private key α in $[1, q-1]$ and a hashing function H, signing a message M is done according to Algorithm 1, and the pair (r, s) forms the signature.

The verification process consists of computing the point

$$\widetilde{Q} = [\mathrm{H}(M)s^{-1}]P + [rs^{-1}]P_{\mathrm{pub}} \tag{2}$$

where $P_{\mathrm{pub}} = [\alpha]P$ is the public key of the signer, and the signature is valid if r is equal to the x-coordinate of $\widetilde{Q}$ (lifted as an integer, then reduced modulo q).

Algorithm 1 ECDSA signature generation.

Require: message M, private key α, point P of order q on an elliptic curve
Ensure: signature (r, s) of the message M under the private key α

1: **repeat**
2: $k \leftarrow$ random integer in $[1, q-1]$
3: $Q \leftarrow [k]P$
4: $r \leftarrow x_Q \bmod q$
5: $s \leftarrow k^{-1}(\mathrm{H}(M) + \alpha r) \bmod q$
6: **until** $r \neq 0$ and $s \neq 0$
7: **return** (r, s)

Faulty Signature. The differential fault attack of this paper results in the production of a faulty signature. If a fault is made during the scalar multiplication such that the output is Q', then the resulting signature (r', s') is

$$\begin{cases} r' = x_{Q'} & \bmod q \\ s' = k^{-1}(\mathrm{H}(M) + \alpha r') & \bmod q. \end{cases}$$

It is not possible to recover the full signature (r, s) from the faulty signature, but the point Q from which r is derived can be reconstructed using the public point of the signer from the relation that is used for signature verification:

$$[\mathrm{H}(M)s'^{-1}]P + [r's'^{-1}]P_{\mathrm{pub}} = [k]P = Q.$$

The point Q' can also be obtained by lifting the integer r' as a point on the elliptic curve. However, the value $x_{Q'}$ has been reduced modulo the prime q. It has been shown that outside of Q' there are only a few possible points [2]. Since the prime q is generally the curve cardinality and is very close to the field order, there are likely only two possible points, Q' and $-Q'$.

Therefore, an attacker can obtain both Q and Q', which is a major part of a DFA attack.

2.3 Hidden Number Problem

The attack presented in this paper allows an attacker to retrieve partial knowledge of the nonce in an ECDSA signature. This can be turned into an instance of the Hidden Number Problem, and solve it using lattices to recover the private key [8,25].

By injecting the partial information of nonces in the linear equations of n signatures, it can be rewritten as a linear system of n equations and $(n+1)$ variables:

$$u_i X + v_i \equiv Y_i, \pmod q, \quad 1 \leq i \leq n. \tag{3}$$

The unknowns are X (the private key α) and $Y_1, \ldots, Y_n$ (the unknown parts of the nonces). The Hidden Number Problem is finding X when the variables Y_i are known to belong in a relatively small interval.

The simplest case is when the most significant bits (respectively least) of the nonces are known, thus the variables Y_i consist of their least significant bits (respectively most). The number of signatures to collect depends on the leak obtained on each nonce. We can get a rough idea with a rule of thumb: with a t-bit curve and ℓ bits leaked per nonce, we can expect around t/ℓ signatures for the lattice attack to succeed. For instance, with 5 least significant bits leaked on a 256-bit curve, an average of 54 signatures are generally sufficient. Therefore, this step in the attack has a negligible cost from a few milliseconds up to a few seconds.

Explanation and construction of lattices for each situation are detailed in Appendix A.

3 DFA on Montgomery Ladder

In this section, we present the Montgomery ladder algorithm, then we describe our attack.

3.1 The Montgomery Ladder Algorithm

One advantage of the Montgomery ladder algorithm for computing $Q = [k]P$ is that the same elliptic curve operations are executed for each bit processed: the algorithm has a regular behavior.

This is done by using two variable points R_0 and R_1 that satisfy the invariant $R_1 - R_0 = P$ in each loop. Let $k = (k_{n-1}, \ldots, k_0)_2$ the binary representation of the scalar k, and suppose the leading bits $\widehat{k}$ down to k_j are already processed, meaning that $R_0 = [\widehat{k}]P$ and $R_1 = [\widehat{k}+1]P$. The state of the Montgomery ladder algorithm is updated depending on the current bit k_{j-1} as follows:

$$(R_0, R_1) = \begin{cases} ([2]R_0, R_0 + R_1) & \text{if } k_{j-1} = 0, \\ (R_0 + R_1, [2]R_1) & \text{if } k_{j-1} = 1. \end{cases} \tag{4}$$

As a consequence, at the end of the step, we have $R_0 = [2\widehat{k} + k_{j-1}]P$, and the relation $R_1 - R_0 = P$ still holds. The process goes on until the last bit, and the final state gives $R_0 = [k]P$.

To avoid branch conditions, a conditional swap with bitwise masking techniques is commonly used in implementations so the point doubling is executed with the correct value, and a second time after the operations to restore R_0 and R_1 (see Algorithm 2).

Remark 1. There is also a padding method to avoid a leak of the bit length of its input given in [10], using the group order q of bit length t: the scalar k is replaced with $k+\varepsilon q$ with $\varepsilon \in \{1, 2\}$ that makes the new scalar exactly a $(t+1)$-bit integer. Since q is the order of the base point, the final result of the scalar multiplication is unchanged. We suppose in the following that this countermeasure is implicitly used.

Algorithm 2 Montgomery ladder

Require: $k = (k_{n-1}, \ldots, k_0)_2$, P, $k_{n-1} = 1$
Ensure: $Q = [k]P$

1: $R_0 \leftarrow P$
2: $R_1 \leftarrow [2]P$
3: **for** $i = n - 2$ **down to** 0 **do**
4: conditional_swap(k_i, R_0, R_1)
5: $R_1 \leftarrow R_0 + R_1$
6: $R_0 \leftarrow [2]R_0$
7: conditional_swap(k_i, R_0, R_1)
8: **return** R_0

3.2 New Attack: Invariant Sign-Change Fault

We consider a fault that inverts the state of the ladder algorithm after the processing of the bit k_j (see Sect. 5 for examples of how it can be achieved):

$$\begin{cases} R_0 = [\widehat{k}]P \\ R_1 = [\widehat{k}+1]P \end{cases} \xrightarrow[\text{fault}]{\lightning} \begin{cases} R_0 = [\widehat{k}+1]P \\ R_1 = [\widehat{k}]P \end{cases} \tag{5}$$

The value R_0 for processing the next bit is $R' = [\widehat{k}+1]P$ and the invariant for the remainder of the algorithm is the point $I = -P$. Thus, the resulting point of the scalar multiplication is

$$Q' = [2^j]R' + [\overline{k}]I = [(\widehat{k}+1)2^j - \overline{k}]P, \tag{6}$$

where $\overline{k} = (k_{j-1}, \ldots, k_0)_2$ are the least significant bits following the bit processed when the fault was made.

Then, the following difference is a point that depends only on the j least significant bits of k:

$$Q - Q' = [2\overline{k} - 2^j]P. \tag{7}$$

Those j bits can be found with an exhaustive search. An alternative is to calculate the sum

$$Q + Q' = [\widehat{k}2^{j+1} + 2^j]P, \tag{8}$$

that depends only on the most significant bits of k.

While this type of fault is undetectable with a point validation, a check of the invariant reveals that a wrong calculation occurred. This is true for classical formulas such as affine point addition or their projective equivalent (including the complete formulas of [28]). However, there are specific formulas that do not use all point coordinates which has an impact on the previous description and makes the fault undetectable by a check of the invariant, and it is covered below.

Remark 2. In the particular case of $\overline{k} = 2^{j-1}$ the points Q and Q' are equal, so it is impossible to distinguish with the cases where the fault has no impact on the swap operation.

***x*-only Formulas.** The particularity of those formulas is that the y-coordinates of the points are not used to compute either a point doubling or a point addition [9,18]. Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$. Given x_1 and x_2, and the auxiliary value x_P the x-coordinate of $P_1 - P_2$, then those formulas compute the x-coordinate of $P_1 + P_2$. No auxiliary data is needed for the point doubling (outside of the elliptic curve parameters). Those formulas are well adapted for the Montgomery ladder algorithm since the point addition occurs between two points whose difference is invariant and equal to the input of the scalar multiplication.

The invariant is replaced by $-P$ in our attack. Its x-coordinate is the same as P, so the point additions in the following steps are correctly calculated, and the differential analysis can be done.

The interesting side-effect happens for the reconstruction of the missing coordinate y of the resulting faulty point, since it uses formulas involving the two coordinates of the invariant (see Appendix B for the formula). In this case, the invariant has changed from (x_P, y_P) to $(x_P, -y_P)$, and the code might use the original invariant directly, stored in registers and not modified by the execution. The sign difference only impacts the y-coordinate of the output Q' which is the same as in Eq. (6) with a sign change (so it passes a point validation test). However, the attack on ECDSA needs to construct candidates for Q' which also includes $-Q'$ so it makes no difference in the analysis.

Furthermore, a check of the invariant would not detect the fault. Indeed, the points R_0 and R_1 will be reconstructed as $R'_0 = -R_0$ and $R'_1 = -R_1$ as explained above, so the difference

$$R'_1 - R'_0 = -(R_1 - R_0) = P$$

would yield the invariant P as if no fault occurred.

Co-*Z* Formulas. We look now at the co-Z formulas based on Jacobian projective representation of the points: a point (x, y) is represented by $(X : Y : Z)$ with $x = X/Z^2$ and $y = Y/Z^3$. The particularity of those formulas is the requirement that the two points share the third projective coordinate Z. We consider the variant that does not use this coordinate in the formulas [17].

Instead of a regular point doubling and point addition in a ladder step, it is composed of two additions, `XYcoZ-ADDC` and `XYcoZ-ADD`, such that the two inputs share the same Z-coordinate, and give two outputs with the same property:

$$\begin{aligned} \texttt{XYcoZ-ADDC} &: (P_1, P_2) \mapsto (P_1 + P_2, P_1 - P_2) \\ \texttt{XYcoZ-ADD} &: (P_1, P_2) \mapsto (P_1 + P_2, P_1). \end{aligned}$$

A formula for the recovery of the missing coordinate Z is necessary at the end of the scalar multiplication during the processing of the last bit to get the affine form.

The formulas are correct as long as the points share the Z-coordinate, and this property is not impacted by the attack. What remains to observe is the effect on the Z-coordinate recovery. The original invariant (x_P, y_P) might be

used instead of the new one $(x_P, -y_P)$ following the fault for the same reason as with the x-only formulas, and the consequence is the appearance of a factor -1 in the reconstructed coordinate Z of the points R_0 and R_1 (see Appendix B for details). But it only changes the sign of the affine coordinate y due to the Jacobian coordinates. So the erroneous output Q' is a valid point of the elliptic curve.

Finally, as with the x-only formulas, a check of the invariant would not detect the fault. Indeed, we have $R_1 - R_0 = -P$ after the fault, and the points are reconstructed as $R'_0 = -R_0$ and $R'_1 = -R_1$ so the difference $R'_1 - R'_0$ would yield the correct invariant.

Remark 3. An alternative view for the x-only and co-Z formulas is that the invariant of the algorithm is not the full point P anymore, but only its x-coordinate which stays intact during the attack.

Unknown Step. The differential points in Eq. (7) and (8) do not depend only on the least or most significant bits of the secret scalar, but also on the step where the fault was made. This could result in several candidates if several steps j are considered during the analysis.

Conservative choices can be made to lift this indeterminacy, at the cost of losing a few bits of the scalar. Suppose we retrieved the discrete logarithm $d = 2\overline{k} - 2^j$ of the differential point of Eq. (7), but the step j is unknown. We can compute $d/2 \bmod 2^i$ for an integer i that we expect to be smaller than j (say 5 for i against 10 for j), then the i least significant bits are retrieved.

The loss of precision is not impactful as the lattice attack on ECDSA can still be successful from a few bits per nonce.

4 DFA with Scalar Randomization

In this section, we present how differential fault attacks might still be applicable when the scalar is randomized with scalar blinding methods in the context of ECDSA.

4.1 Scalar Blinding with Group Order

This is the most classical measure proposed in [13]. The secret scalar k is replaced with

$$k^\star = k + mq,$$

where m is a random integer of λ bits. Since q is the order of the base point P, then we have

$$Q = [k^\star]P = [k]P + [mq]P = [k]P.$$

Write $k^\star = \widehat{k}^\star 2^j + \overline{k}^\star$ where $\overline{k}^\star$ are the j least significant bits, and $\widehat{k}^\star$ the most significant bits. Suppose that a DFA reveals $\overline{k}^\star$ (as in our attack on Montgomery

ladder), then the unknown part satisfies the inequality

$$\frac{q - \overline{k}^\star}{2^j} \leq \widehat{k}^\star < \frac{q + 2^\lambda q - \overline{k}^\star}{2^j}, \tag{9}$$

which is an interval of width $q/2^{j-\lambda}$. Then, it is necessary to have $j > \lambda$ for the unknown to be in an interval of width less than q, a necessity for exploitation in a lattice attack on ECDSA as described in Appendix A.

Cost. Suppose that we get a point whose discrete logarithm depends on the $j = \lambda + \varepsilon$ least significant bits of the blinded scalar for a nonnegative ε. An exhaustive search on those bits is expensive in this case, so a discrete logarithm algorithm such as BSGS and Pollard's kangaroo might be used to find the bits more efficiently in complexity $O(2^{(\lambda+\varepsilon)/2})$. For example, if λ is 20 (as was originally suggested in [13]), then a fault on the step $j = 24$ would make the discrete logarithm easy to find, and a small ε is sufficient to attack ECDSA as can be attested in our simulation tools. Therefore, the cost depends essentially on λ which has to be chosen quite large to prevent the attack or to make it impractical.

4.2 Euclidean Splitting

This method was proposed in [12] to protect against side-channel attacks as an alternative for scalar blinding. The secret scalar k is rewritten as

$$k = am + b,$$

where m is a random integer of λ bits with $a = \lfloor k/m \rfloor$ and $b = k \bmod m$. Then, the scalar multiplication $Q = [k]P$ can be computed as $Q = [m]([a]P) + [b]P$ using three individual scalar multiplications and a point addition.

We show here how to recover the random divisor (or one of its factors) and the remainder of the Euclidean division of a secret scalar k from a single fault. This can be used in ECDSA for a lattice attack, and on a fixed scalar with the Chinese Remainder Theorem.

We start by giving the general principle. Let $R = [a]P$ the scalar multiplication with the quotient, so the output is given by

$$Q = [m]R + [b]P.$$

If the point R is known, then the BSGS algorithm can be applied to find m and b. It consists of computing a first list of possible values for $[b]P$ (the *baby steps*), then a second list of possible points for $Q - [m]R$ (the *giant steps*) until a collision with the first list occurs, revealing the values m and b.

The first list depends only on the base point, so it can be computed once and stored for reuse. Since both m and b are less than 2^λ, both the time and space complexities of the algorithm are $O(2^\lambda)$.

The proposed target is the scalar multiplication $[m]R$ with the random divisor. We suppose a fault has been made such that the effective calculation is $[m']R$ where the difference $\delta = m - m'$ belongs to a set of size T. Then the result of the whole scalar multiplication $Q = [m]R + [b]P$ is altered in a point $Q' = [m']R + [b]P$, and their difference is

$$Q - Q' = [\delta]R.$$

A candidate for R is constructed from each candidate $\tilde{\delta}$ for δ:

$$\widetilde{R} = [1/\tilde{\delta}](Q - Q').$$

The BSGS strategy is applied to get candidates $(\tilde{m}, \tilde{b})$ that satisfy the equality

$$Q - [\tilde{m}]\widetilde{R} = [\tilde{b}]P.$$

Cost. There are T possible values for δ and the BSGS algorithm runs in $O(2^\lambda)$ steps, so the overall cost is $O(2^\lambda T)$. So it is practical only for a small parameter λ (the only library implementing this technique that we found uses a parameter λ of 32 bits so the time and memory constraints are low enough). In particular, the memory constraints of BSGS should make the attack infeasible for $\lambda = 64$.

Several Candidates. Eventually, several candidates for (m, b) can be found, but we can still salvage valuable information on the scalar k. Let $(\tilde{m}, \tilde{b})$ a candidate alongside the corresponding value $\tilde{\delta}$. The correct values (m, b) and δ are also amongst the candidates.

We start with the case $b \neq \tilde{b}$. Since $(\tilde{m}, \tilde{b})$ is a candidate, we have

$$[\tilde{m}]\widetilde{R} + [\tilde{b}]P = [m]R + [b]P, \tag{10}$$

from which we derive the relation

$$a \equiv \tilde{\delta}(\tilde{b} - b)(m\tilde{\delta} - \tilde{m}\delta)^{-1} \pmod{q}. \tag{11}$$

The quotient a is recovered, so the scalar k can be fully reconstructed and verified with the relation $Q = [k]P$. This case seems unlikely to happen.

In the case the candidates are $(\tilde{m}_1, b)$, …, $(\tilde{m}_N, b)$, then we can pose $d = \gcd(\tilde{m}_1, \ldots, \tilde{m}_N)$, and we get the relation $k \equiv b \pmod{d}$.

Example with the Invariant Sign-Change Fault. We apply the attack of Sect. 3.2 when no specific formulas are used. After a fault on the scalar multiplication $[m]R$, then the result is $[m']R$ with $m' = (\widehat{m} + 1)2^j - \overline{m}$. The difference with the correct output Q is

$$Q - Q' = [2\overline{m} - 2^j]R,$$

and the value $\delta = 2\overline{m} - 2^j$ depends only on the least significant bits of m. Therefore, the BSGS part of the attack only needs to run an exhaustive search on the most significant bits of m. If the *baby steps* are precomputed, then the complexity is $O(2^\lambda)$.

4.3 Multiplicative Splitting

This technique was proposed in [38]. A random value m of λ bits is randomly generated, and γ is defined such that we have the relation

$$k \equiv m\gamma \pmod{q}.$$

The scalar multiplication $Q = [k]P$ is computed in two successive scalar multiplications as $R = [m]P$ and $Q = [\gamma]R$.

The differential fault attack can be applied with a fault in the second scalar multiplication. We suppose a fault has been made such that the effective calculation is $[\gamma']R$ where the difference $\delta = \gamma - \gamma'$ belongs to a set of size T. Then the result of the whole scalar multiplication $Q = [\gamma m]P$ is altered in a point $Q' = [\gamma' m]P$, and their difference is

$$Q - Q' = [\delta m]P.$$

The value δm can be found by running through all possible values for δ, and then computing the discrete logarithm of the point $[m]P$ in base P with BSGS or Pollard's kangaroo algorithms.

Cost. Since m is a positive integer less than 2^λ the overall cost is $O(2^{\lambda/2}T)$. The cost is similar to the attack on the blinding with the group order, so it is tractable for small λ (such as 20 or 32).

Example with the Invariant Sign-Change Fault. In certain cases, a single discrete logarithm is sufficient when δ represents a small value. For example, if we consider the fault of Sect. 3.2 during the processing of the bit γ_j, then the difference with the result of the whole scalar multiplication is

$$Q - Q' = [(2\overline{\gamma} - 2^j)m]P.$$

We obtain a point whose discrete logarithm in base P is less than $2^{\lambda+j}$ (in absolute value), so the complexity to find it is $O(2^{(\lambda+j)/2})$. Again, this is practical when λ is relatively small (as was suggested in the paper that proposed this method), then the discrete logarithm can be found in a matter of seconds or minutes.

This discrete logarithm is useful for lattice attacks on ECDSA. Indeed, we have the relation

$$\frac{k - m\overline{\gamma}}{m2^j} \equiv \widehat{\gamma} \pmod{q}, \tag{12}$$

where $\widehat{\gamma}$ is a relatively small integer compared to the order q (at least j bits less). When it is possible to distinguish m from $2\overline{\gamma} - 2^j$ in the discrete logarithm (if m is prime for instance), then a lattice attack can be applied.

Table 1. Overview of Montgomery ladder in several cryptographic libraries.

Library	Init.	Swap variant	Formulas	Remarks
Weierstrass curves				
OpenSSL 1.1.1k	$(P, 2P)$	Algorithm 3*	x-only	
LibreSSL 3.2.4	$(P, 2P)$	Algorithm 3*	Jacobian	
CoreCrypto (Apple)	$(P, 2P)$	Algorithm 3	Co-Z	Point valid., Eucl. split
Montgomery curves				
SymCrypt	$(\mathcal{O}, P)$	Algorithm 3	x-only	
Mbed TLS	$(\mathcal{O}, P)$	Algorithm 2	x-only	
libsodium †	$(\mathcal{O}, P)$	Algorithm 3	x-only	

*The source code is slightly different but the compiled code corresponds to Algorithm 3.
†`ref10` implementation of Curve25519 present in other libraries.

Algorithm 3 Processing of the bit k_i in the Montgomery ladder variants with merged swaps.

1: $\mathsf{pbit} \leftarrow \mathsf{pbit} \oplus k_i$
2: $\mathsf{conditional\_swap}(\mathsf{pbit}, R_0, R_1)$
3: $R_1 \leftarrow R_0 + R_1$
4: $R_0 \leftarrow [2]R_0$
5: $\mathsf{pbit} \leftarrow k_i$

5 Practical Evaluation

In this section, we consider the practicality of the attack and present evidence on how it can be achieved on several cryptographic libraries, with simulated experiments to validate our claims that are publicly available.[1]

5.1 Montgomery Ladder in Libraries

In most cryptographic libraries, the second swap in the loop is merged with the first swap of the next step to avoid an unnecessary swap: the swap is effective only if the scalar bit differs from the previous one. This variant is presented in Algorithm 3.

We list in Table 1 the variant used for several libraries that implement the Montgomery ladder algorithm for the elliptic curve scalar multiplication. Elliptic curves in Montgomery form such as Curve25519 are also present in the table, though those are not used for ECDSA, the attack might still be applicable in situations where the attacker can obtain the correct and erroneous outputs.

It shall be noted that in some cases side-channel attacks could be sufficient, but those are inherent to how the actual swap is implemented. There is the binary masking technique, where the swap is made using a mask with its bits

[1] https://github.com/orangecertcc/dfa-ladder.

all set to 0 or 1, and a template attack was applied where the leak comes from the `AND` binary operator [23]. In the case of Mbed TLS, the multiplication by 0 or 1 is used to swap the values, and is also vulnerable to a template attack [21]. In both cases, a single trace could reveal the whole scalar.

Assuming that an implementation is protected against these attacks, then a fault attack becomes relevant. In the following we present a strategy to perform our attack with the variant of Algorithm 3.

5.2 Realization of the Fault Attack

Physical access to the device is necessary, and the attacker must be able to disturb the calculation at a specific point in time and location.

Skip Instruction. The first model considered is the skip instruction that was applied successfully in practice on RSA exponentiation with a spike injection on a microcontroller to skip a squaring [31]. It was also recently applied in the elliptic curve point decompression algorithm to make a point lie on weak curve [5,36].

The effects described in Sect. 3.2 can be achieved if line 1 of Algorithm 3 is omitted during one iteration of the algorithm. Indeed, the variable pbit at the beginning of the loop refers to the previous scalar bit, and keeps track of the current state of the couple (R_0, R_1) such that the loop invariant is

$$R_{1-\mathsf{pbit}} - R_{\mathsf{pbit}} = P.$$

So, if the line "$\mathsf{pbit} \leftarrow \mathsf{pbit} \oplus k_i$" is not executed and the bit k_i is 1, then the variable pbit is not updated:

- If pbit was 0, then we have $R_1 - R_0 = P$, the points are not switched, so we still have $R_1 - R_0 = P$;
- If pbit was 1, then we have $R_0 - R_1 = P$, the points are switched, so we have now $R_1 - R_0 = P$.

In both cases, the variable pbit gets the value 1 at the end of the loop, so starting from the next iteration we have

$$R_{1-\mathsf{pbit}} - R_{\mathsf{pbit}} = R_0 - R_1 = -P,$$

and the sign of the loop invariant has changed for the remainder of the algorithm.

One alternative is to target the line "$\mathsf{pbit} \leftarrow k_i$". If this line is skipped and the bit k_i differs from the value in the variable pbit, then it will not be consistent with the current state or (R_0, R_1), but starting from the processing of the next loop iteration.

Remark 4. Of course, in half of the cases, skipping one of these instructions will not have an effect and result in a correct output (and it is discarded in our attack).

Algorithm 4 Constant-time conditional swap of two values with binary operators (comments: alternative version)

Require: (w_0, w_1), bit b
Ensure: (w_b, w_{1-b})
  $\mathsf{mask} \leftarrow (b, \ldots, b)_2$
  $\mathsf{tmp} \leftarrow \mathsf{mask} \wedge (w_0 \oplus w_1)$ ▷ $\mathsf{tmp} \leftarrow w_0$
  $w_0 \leftarrow w_0 \oplus \mathsf{tmp}$ ▷ $w_0 \leftarrow (w_0 \wedge \neg\mathsf{mask}) \vee (w_1 \wedge \mathsf{mask})$
  $w_1 \leftarrow w_1 \oplus \mathsf{tmp}$ ▷ $w_1 \leftarrow (w_1 \wedge \neg\mathsf{mask}) \vee (\mathsf{tmp} \wedge \mathsf{mask})$
  return (w_0, w_1)

Fault in a Register. A common method for the conditional swap is to use binary masks as presented in Algorithm 4. The interesting part is the construction of the binary mask. It is generally done using the binary representation of -1 in a machine-word with all bits set to 1. So the value $-b$ gives a null mask if b is 0, and a binary mask with all bits set to 1 if b is 1.

However, there are other ways to construct such masks, where any nonzero b ends up with a binary mask with bits set to 1. Let N the bit length of machine-words, then the two equivalent following formulas give an example of such construction (the first one is present in Mbed TLS and the second in OpenSSL) where "$\gg$" is the bitwise shift right operator:

$$-\big((b \vee (-b)) \gg (N-1)\big) \quad \text{or} \quad \big((\neg b \wedge (b-1)) \gg (N-1)\big) - 1.$$

A fault that randomly modifies the register that contains the bit b will have the desired effect and swaps the points if the original value of b is 0. This can be achieved with a random fault on a register.

5.3 Simulations

Simulations were used to put in practice our attack and evaluate the other different cases of the paper. The first one uses the GNU Debugger GDB to simulate faults according to the fault models presented above in the OpenSSL implementation of the Montgomery ladder algorithm. The second is based on the Unicorn engine[2] to test the effect of faults wrongly injected with the skip instruction fault model. Finally, other cases with the randomization methods were also simulated in Python and the lattice attack used the `fpylll` library [37].

GDB Simulation. We give in Listing 1.1 part of the assembly code related to the loop of the Montgomery ladder algorithm in OpenSSL version 1.1.1k (compiled on a Raspberry Pi device model 4B).

The instruction on address `0xdd208` corresponds to the line "$\mathsf{pbit} \leftarrow \mathsf{pbit} \oplus k_i$" that needs to be ignored in the skip instruction fault model. The second fault model can be achieved with a modification of register `r6` after this same

[2] https://www.unicorn-engine.org/.

```
dd1e8:  mov     r1, r8
dd1ec:  ldr     r0, [sp, #8]
dd1f0:  bl      8b440 <BN_is_bit_set>       ; r0 <- current bit k_i
dd1f4:  ldr     r6, [sp, #12]               ; r6 <- pbit
dd1f8:  mov     r3, r9
dd1fc:  ldr     r2, [fp, #8]
dd200:  ldr     r1, [r7, #8]
dd204:  sub     r8, r8, #1
dd208:  eor     r6, r6, r0                  ; r6 <- pbit XOR k_i
dd20c:  mov     sl, r0
dd210:  mov     r0, r6
dd214:  bl      8b554 <BN_consttime_swap>   ; swap X if r0 = 1
dd218:  mov     r0, r6
dd21c:  mov     r3, r9
dd220:  ldr     r2, [fp, #12]
dd224:  ldr     r1, [r7, #12]
dd228:  str     sl, [sp, #12]               ; pbit <- k_i
dd22c:  bl      8b554 <BN_consttime_swap>   ; swap Y if r0 = 1
dd230:  mov     r0, r6
dd234:  mov     r3, r9
dd238:  ldr     r2, [fp, #16]
dd23c:  ldr     r1, [r7, #16]
dd240:  bl      8b554 <BN_consttime_swap>   ; swap Z if r0 = 1
```

Listing 1.1. Excerpt of assembly code of the function `ec_scalar_mul_ladder` in OpenSSL 1.1.1k.

instruction. Indeed, this variable is only used thereafter for the conditional swap on each point coordinates.

It is easy to instrument these faults with GDB, and has been automatized with two scripts. In both cases the analysis on the signatures followed by the lattice attack resulted in a successful private key recovery.

Unicorn Simulation. Unicorn is CPU framework emulator and we used it through the Rainbow tool[3] that makes it easy to trace the execution of all instructions of a binary. It can be stopped at any moment and the next instruction can be read. Then the skip instruction fault model can be instrumented as follows: we read the next instruction, and it is skipped by resuming the execution at the following instruction using the size of the skipped instruction.

The constant-time big integer modular arithmetic of the `secp256r1` curve written in assembly was chosen (taken from the OpenSSL project). It has no external dependency which makes it easier to work with the emulator. Two binaries were created to implement the Montgomery ladder variant of Algorithm 3: the first with Jacobian projective coordinates, and the second with co-Z formulas.

The instructions related to the lines "$\mathsf{pbit} \leftarrow \mathsf{pbit} \oplus k_i$" and "$\mathsf{pbit} \leftarrow k_i$" are present in the assembly code of both binaries. When one of those is skipped during an iteration of the main loop, then the analysis of Sect. 3.2 is successful and the least significant bits of the scalar are recovered.

[3] https://github.com/Ledger-Donjon/rainbow.

However, we found a false positive situation with both binaries when the fault skips one instruction in the function that extracts one bit of the scalar. What happens is that the extracted bit is incorrect: the bit k_j is replaced with $1 - k_j$, but the remaining of the scalar multiplication is done correctly. This is equivalent to a bitflip of the scalar, and as a consequence we have

$$Q - Q' = \begin{cases} [-2^j]P & \text{if } k_j = 0, \\ [2^j]P & \text{if } k_j = 1. \end{cases}$$

If we look at Eq. (7), this might wrongly reveal that the least significant bits of the scalar are only composed of zero bits. Therefore, to avoid a wrong signature in the lattice attack against ECDSA, it might be better to discard this case (say j is 16, then there would be one out 65536 scalars on average where the 16 least significant bit are indeed set to 0 so discarding a correct result would be rare).

Another false positive was observed with the Jacobian projective formulas: after a specific instruction skip in the point addition function, one of the points is not loaded correctly and the addition happens with the same point: the doubling function is called instead, and the analysis catches a wrong value.

For the co-Z binary, we included the invariant check at the end of the scalar multiplication as a countermeasure [39]. We adapted the `XYcoZ-ADD` function such that it computes the difference of the inputs (the invariant) instead. Once the missing Z-coordinate is recovered and the points are converted to their affine representation, the calculated invariant I is XORed with P, the correct invariant, and the output Q:

$$Q \oplus I \oplus P.$$

If the calculated invariant is correct, it should be canceled by P. As was expected from Sect. 3.2, I is indeed correctly calculated as P. The output is a valid point and it does not prevent our attack.

6 Countermeasures

As with other works where the fault does not make the elliptic curve point leave the curve, a point validation cannot detect the fault, even in the case of x-only or co-Z formulas (for the former it was suggested in [15] to recover the missing coordinate and perform the verification, but in the context of the attack of [16], and would not be able to prevent our attack).

Verification of the Montgomery ladder invariant was proposed in [14,39] against fault attacks. As we have seen in Sect. 3.2, it should work in general because the invariant is changed, except in the cases of the x-only and co-Z formulas (the reconstructed invariant would be the correct one) as was experimented in Sect. 5.3.

There is another idea from [14] to prevent our attack with the x-only and co-Z formulas. It is a variant of the point blinding countermeasure from [13] adapted to Montgomery ladder: the algorithm is initialized as $R_0 = P + R$ and $R_1 = [2]P + R$ for a random point R, and the invariant $R_1 - R_0$ is still the

input P. At the end of the scalar multiplication we have $R_0 = [k]P + [2^{n-1}]R$, so a subtraction by $S = [2^{n-1}]P$ is needed to get the correct output. In our attack, the points R_0 and R_1 are inverted and the invariant becomes $-P$:

$$\begin{cases} R_0 = [k']P + [2^{n-1}]R \\ R_1 = R_0 - P. \end{cases}$$

We have seen that it changes the sign of the reconstructed point after the recovery of the missing coordinate. Therefore, subtracting the blinded point to get the output would give

$$Q' = -R_0 - [2^{n-1}]R = -[k']P - [2^n]R,$$

and without the knowledge of the point R, the output is useless for an attacker. However, this is true as long as R is randomly selected at each new execution, and it was originally proposed to update R by replacing it with the point $[(-1^{\beta})2]R$ with $\beta \in \{0, 1\}$ chosen randomly. With faults successfully injected in consecutive runs, it might be possible to deduce the point R, and then the differential analysis could be done.

Classic countermeasures such as repeating the operations twice and check consistency can be applied against our attack. To reduce the cost, it was proposed in [6] to make the second computation on an elliptic curve $E_{p'}$ over a smaller prime field $\mathbf{F}_{p'}$, and the first on an elliptic curve $E_{pp'}$ over the integer ring $\mathbf{Z}/pp'\mathbf{Z}$. On one hand, the reduction modulo p of the result gives the expected calculation, and on the other hand, the reduction modulo p' is checked with the calculation on $E_{p'}$.

A variant of the previous method was proposed in [19,32] where the second computation is done on an auxiliary group glued together with the elliptic curve operation. For instance, it can be done by adding an integer to point coordinates which keeps track of the current discrete logarithm of the points using the following rules:

$$(P_1, \ell_1) + (P_2, \ell_2) = (P_1 + P_2, \ell_1 + \ell_2), \quad [2](P, \ell) = ([2]P, 2\ell).$$

If no fault occurred the resulting point should be $([k]P, k)$ with k the secret scalar. This method should detect the fault in our attack since the auxiliary value is consistent with the point, so a change in the point affects the value too.

Finally, in the case of an attack on ECDSA, it is always possible to verify the signature at the end of the calculation.

7 Conclusion

In this paper, we presented a new differential fault attack on Elliptic Curve Cryptography with the Montgomery ladder algorithm. We showed that an attacker can switch two points with either a skip instruction or a random fault in a

register. With this modification in the program flow, the least (or most) significant bits of the secret scalar can be determined from the difference between the correct and erroneous outputs.

Furthermore, this attack bypasses some of past countermeasures against fault attacks on ECC. A consequence is that particular care is necessary to choose the right measures to protect an implementation when protection against fault attacks is part of the threat model.

Finally, we presented evidence that scalar randomization with common methods is not enough to thwart differential fault attacks. It requires that the randomizer is small enough for the attack to be practical. However, it is generally suggested in the papers that proposed such methods to choose them small to reduce the extra cost.

Future work could explore further ways to achieve the effects of our attack using other fault models or targeting other instructions, or investigate other randomization methods.

A Lattice Attack

In this appendix, we present the lattice construction to solve the Hidden Number Problem, then we give the values for the particular cases met in the paper.

A.1 Lattice Construction

First we introduce the notation $|\cdot|_q$ defined as

$$|z|_q = \min_{y \in \mathbf{Z}} |z - yq|,$$

for any real z, which is a reduction modulo q in the range $[-q/2, q/2]$ followed by an absolute value.

Let $uX + v \equiv Y \bmod q$ a linear equation in the variables X and Y, where an approximation viewed as an integer of Y is known:

$$B_1 \leq Y < B_2,$$

where B_1 and B_2 are two integers with $(B_2 - B_1) < q/L$ for a positive integer L. The width of the interval can be reduced by centering around 0. Let $C = (B_1 + B_2)/2$ be the center of the interval, and we get the bound

$$|Y - C| < q/(2L).$$

Therefore, noting $v' = C - v$, we have $|uX - v'|_q = |Y - C|_q = |Y - C|$, since we know that $(Y - C)$ is in $[-q/2, q/2]$. Then, using the bound on it, we obtain the inequality

$$|uX - v'|_q < q/(2L), \tag{13}$$

whose meaning is that when seen modulo q, the value v' is close to a multiple of the hidden number X.

Given n inequalities of the form $|u_i X - v'_i|_q < q/(2L_i)$ derived from equations where X is a common variable, we construct the lattice generated by this integer matrix:

$$\begin{bmatrix} 2L_1 q & & & & & \\ & 2L_2 q & & & & \\ & & \ddots & & & \\ & & & 2L_n q & & \\ 2L_1 u_1 & 2L_2 u_2 & \cdots & 2L_n u_n & 1 & 0 \\ 2L_1 v'_1 & 2L_2 v'_2 & \cdots & 2L_n v'_n & 0 & q \end{bmatrix}.$$

Denoting $\mathbf{m}_i$ the i-th line of the matrix, and

$$\begin{cases} \mathbf{u} = (2L_1 u_1, \ldots, 2L_n u_n, 1, 0) \\ \mathbf{v} = (2L_1 v'_1, \ldots, 2L_n v'_n, 0, q), \end{cases}$$

the vectors from the last two lines of the matrix, there exists integers λ_i such that the vector $X\mathbf{u} - \mathbf{v} + \sum_{i=1}^{n} \lambda_i \mathbf{m}_i$ is a short vector of the lattice according to the inequalities, since each coordinate is bounded by q.

Applying a lattice basis reduction algorithm such as LLL [20] or BKZ [34], there is a possibility that one of the vectors of the reduced basis is the one we are looking for. By construction, this short vector has the hidden number X as its penultimate coordinate. In the different settings, it corresponds with a private key and can be easily checked with the public key.

A.2 Application to ECDSA

First, we recall that a signature (r, s) can be rewritten:

$$k \equiv \alpha r/s + \mathrm{H}(M)/s \pmod{q}.$$

The hidden number X is the private key α, while the unknown variable Y corresponds partially to the ephemeral value k in various ways depending on each of the following cases.

Case 1. Let $k = am + b$ the Euclidean division of k by an integer m. This case concerns the attack on the Euclidean splitting in Sect. 4.2, but also cases when the ℓ least significant bits are known by setting $m = 2^\ell$. If m and b are known, then we have

$$\begin{cases} Y = a, \\ u = r/(sm) \bmod q, \\ v = \mathrm{H}(M)/(sm) - b/m \bmod q. \end{cases}$$

With the padding method applied on k beforehand, we have $2^t \leq k < 2^t + q$ so the unknown a is bounded by

$$\frac{2^t - b}{m} \leq a < \frac{2^t + q - b}{m},$$

of width q/m. When the blinding method of Sect. 4.1 is used instead of a padding, we have the inequality in Eq. (9) when the ℓ least significant bits are known.

Case 2. Instead, if a and m are known in the Euclidean division of k by m, then we have

$$\begin{cases} Y = b, \\ u = r/s \bmod q, \\ v = \mathrm{H}(M)/s - am \mod q. \end{cases}$$

The unknown b is a non-negative integer less than m. This case corresponds to the most significant bits of k known. Suppose that we know the $(\ell + 1)$ most significant bits (including one from the padding), then $m = 2^{t-\ell}$ (which can be approximated to $q/2^\ell$ when q is very close to 2^t for some standardized elliptic curves).

Case 3. This is the situation of Sect. 4.3 where k is randomized by an integer m of at most λ bits, and is rewritten as $k \equiv m\gamma \bmod q$. If m and the ℓ least significant bits $\overline{\gamma}$ of γ are known, we can write

$$k \equiv m\widehat{\gamma}2^\ell + m\overline{\gamma} \pmod q,$$

where the unknown is $\widehat{\gamma}$ (the most significant bits of γ). We have

$$\begin{cases} Y = \widehat{\gamma}, \\ u = r/(ms2^\ell) \bmod q, \\ v = \mathrm{H}(M)/(sm2^\ell) - \overline{\gamma}/2^\ell \bmod q. \end{cases}$$

If no padding was applied on the scalar multiplication with γ, then the unknown is bounded by $0 \leq \widehat{\gamma} < q/2^\ell$, and if a padding is applied then the bound on the unknown is

$$\frac{2^t - \overline{\gamma}}{2^\ell} \leq \widehat{\gamma} < \frac{2^t + q - \overline{\gamma}}{2^\ell},$$

both of them of width $q/2^\ell$.

B Coordinates Recovery

In this appendix, we give more details on the recovery of the missing coordinates for the x-only and co-Z formulas.

B.1 x-only Formulas

Those are based on the homogeneous projective coordinates: a point (x, y) is represented as $[X : Z]$ where $x = X/Z$ for a nonzero Z.

The missing coordinate can be recovered in the case of the Montgomery ladder algorithm. Let $[X_0 : Z_0]$ and $[X_1 : Z_1]$ the representations of two points R_0 and R_1, and $P = (x_P, y_P)$ the point such that $R_1 - R_0 = P$. The formula to recover the affine y-coordinate of R_0 is

$$y_0 = \frac{2BZ_0^2Z_1 + Z_1(AZ_0 + x_PX_0)(x_PZ_0 + X_0) - X_1(x_PZ_0 - X_0)^2}{2y_PZ_0^2Z_1} \tag{14}$$

Algorithm 5 Recovery of the missing coordinate with co-Z formulas in the last step of the Montgomery ladder algorithm.

1: $R_{1-k_0}, R_{k_0} \leftarrow$ `XYcoZ-ADDC`(R_{k_0}, R_{1-k_0})
2: $Z \leftarrow (X(R_1) - X(R_0))x_P Y(R_{k_0})/y_P X(R_{k_0})$
3: $R_{k_0}, R_{1-k_0} \leftarrow$ `XYcoZ-ADD`(R_{1-k_0}, R_{k_0})
4: $Q = (X(R_0)/Z^2, Y(R_0)/Z^3)$

If the invariant sign-change fault attack of Sect. 3.2 is successful, then we have $R_1 - R_0 = -P = (x_P, -y_P)$. A correct reconstruction of R_0 should use $-y_P$ instead of y_P in Eq. (14). On the contrary, it will introduce a factor -1 in the computation of y_0, so the reconstructed point will be $R'_0 = -R_0$. A similar formula can be derived for R_1 and the reconstruction would give $R'_1 = -R_1$. So the difference $R'_1 - R'_0$ would still be equal to the original invariant.

B.2 Co-Z Formulas

Those formulas are based on the Jacobian projective coordinates: a point (x, y) is represented by $(X : Y : Z)$ with $x = X/Z^2$ and $y = Y/Z^3$. The variant considered does not use the third coordinate Z in calculation.

The missing coordinate is mandatory to get the affine representation. Let $P = (X : Y : Z)$ with the coordinate Z unknown, and (x, y) its known affine representation, then we have

$$Z = \frac{x \cdot Y}{y \cdot X}. \tag{15}$$

On the last step of the Montgomery ladder algorithm, the `XYcoZ-ADDC` operation computes the difference of the points R_0 and R_1, so the invariant P appears in Jacobian projective form. The above formula allows the reconstruction of its missing Z-coordinate which is common to R_0 and R_1. This is given in Algorithm 5 (another factor is present in line 2 to take into account the final `XYcoZ-ADD` operation).

If the invariant sign-change fault attack of Sect. 3.2 is successful, then the invariant becomes $-P$ which introduces a factor -1 in Eq. (15) since the affine coordinates of P will be used. As a consequence, the missing coordinate reconstructed in line 2 of Algorithm 5 will be $-Z$. Let $R_0 = (X_0 : Y_0 : Z)$ and $R_1 = (X_1 : Y_1 : Z)$ the two points at the end of the Montgomery ladder algorithm. Then using $-Z$ instead of Z to get the affine form will result in

$$R'_i = \left(\frac{X_i}{(-Z)^2}, \frac{Y_i}{(-Z)^3}\right) = (x_i, -y_i) = -R_i, \quad i \in \{0, 1\}.$$

So the difference $R'_1 - R'_0$ would still be equal to the original invariant.

References

1. Ambrose, C., Bos, J.W., Fay, B., Joye, M., Lochter, M., Murray, B.: Differential attacks on deterministic signatures. In: Smart, N.P. (ed.) CT-RSA 2018. LNCS, vol. 10808, pp. 339–353. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-76953-0_18
2. Barenghi, A., Bertoni, G.M., Breveglieri, L., Pelosi, G., Sanfilippo, S., Susella, R.: A fault-based secret key retrieval method for ECDSA: analysis and countermeasure. ACM J. Emerg. Technol. Comput. Syst. **13**(1), 8:1–8:26 (2016). https://doi.org/10.1145/2767132
3. Biehl, I., Meyer, B., Müller, V.: Differential fault attacks on elliptic curve cryptosystems. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 131–146. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44598-6_8
4. Biham, E., Shamir, A.: Differential fault analysis of secret key cryptosystems. In: Jr., B.S.K. (ed.) CRYPTO '97. LNCS, vol. 1294, pp. 513–525. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0052259
5. Blömer, J., Günther, P.: Singular curve point decompression attack. In: Homma, N., Lomné, V. (eds.) FDTC 2015. pp. 71–84. IEEE Computer Society (2015). https://doi.org/10.1109/FDTC.2015.17
6. Blömer, J., Otto, M., Seifert, J.P.: Sign change fault attacks on elliptic curve cryptosystems. In: Breveglieri, L., Koren, I., Naccache, D., Seifert, J. (eds.) FDTC 2006. LNCS, vol. 4236, pp. 36–52. Springer, Heidelberg (2006). https://doi.org/10.1007/11889700_4
7. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the importance of checking cryptographic protocols for faults. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 37–51. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-69053-0_4
8. Boneh, D., Venkatesan, R.: Hardness of computing the most significant bits of secret keys in Diffie-Hellman and related schemes. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 129–142. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_11
9. Brier, E., Joye, M.: Weierstraß elliptic curves and side-channel attacks. In: Naccache, D., Paillier, P. (eds.) PKC 2002. LNCS, vol. 2274, pp. 335–345. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45664-3_24
10. Brumley, B.B., Tuveri, N.: Remote timing attacks are still practical. In: Atluri, V., Díaz, C. (eds.) ESORICS 2011. LNCS, vol. 6879, pp. 355–371. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23822-2_20
11. Cao, W., Feng, J., Chen, H., Zhu, S., Wu, W., Han, X., Zheng, X.: Two lattice-based differential fault attacks against ECDSA with wNAF algorithm. In: Kwon, S., Yun, A. (eds.) ICISC 2015. LNCS, vol. 9558, pp. 297–313. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-30840-1_19
12. Ciet, M., Joye, M.: (Virtually) free randomization techniques for elliptic curve cryptography. In: Qing, S., Gollmann, D., Zhou, J. (eds.) ICICS 2003. LNCS, vol. 2836, pp. 348–359. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-39927-8_32
13. Coron, J.S.: Resistance against differential power analysis for elliptic curve cryptosystems. In: Koç, Ç.K., Paar, C. (eds.) CHES'99. LNCS, vol. 1717, pp. 292–302. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48059-5_25
14. Dominguez-Oviedo, A., Hasan, M.A.: Algorithm-level error detection for Montgomery ladder-based ECSM. J. Cryptogr. Eng. **1**(1), 57–69 (2011). https://doi.org/10.1007/s13389-011-0003-1

15. Ebeid, N.M., Lambert, R.: Securing the elliptic curve Montgomery ladder against fault attacks. In: Breveglieri, L., Koren, I., Naccache, D., Oswald, E., Seifert, J. (eds.) FDTC 2009. pp. 46–50. IEEE Computer Society (2009). https://doi.org/10.1109/FDTC.2009.35
16. Fouque, P., Lercier, R., Réal, D., Valette, F.: Fault attack on elliptic curve Montgomery ladder implementation. In: Breveglieri, L., Gueron, S., Koren, I., Naccache, D., Seifert, J. (eds.) FDTC 2008. pp. 92–98. IEEE Computer Society (2008). https://doi.org/10.1109/FDTC.2008.15
17. Goundar, R.R., Joye, M., Miyaji, A., Rivain, M., Venelli, A.: Scalar multiplication on Weierstraß elliptic curves from co-Z arithmetic. J. Cryptogr. Eng. **1**(2), 161–176 (2011). https://doi.org/10.1007/s13389-011-0012-0
18. Izu, T., Takagi, T.: A fast parallel elliptic curve multiplication resistant against side channel attacks. In: Naccache, D., Paillier, P. (eds.) PKC 2002. LNCS, vol. 2274, pp. 280–296. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45664-3_20
19. Joye, M.: A method for preventing "skipping" attacks. In: IEEE Symposium on Security and Privacy Workshops. pp. 12–15. IEEE Computer Society (2012). https://doi.org/10.1109/SPW.2012.14
20. Lenstra, A.K., Lenstra, H.W.J., Lovász, L.: Factoring polynomials with rational coefficients. Mathematische Annalen **261**, 515–534 (1982). https://doi.org/10.1007/BF01457454
21. Loiseau, A., Lecomte, M., Fournier, J.J.A.: Template attacks against ECC: practical implementation against Curve25519. In: HOST 2020. pp. 13–22. IEEE (2020). https://doi.org/10.1109/HOST45689.2020.9300261
22. Montgomery, P.L.: Speeding the Pollard and elliptic curve methods of factorization. Math. Comput. **48**, 243–264 (1987). https://doi.org/10.1090/S0025-5718-1987-0866113-7
23. Nascimento, E., Chmielewski, L., Oswald, D.F., Schwabe, P.: Attacking embedded ECC implementations through cmov side channels. In: Avanzi, R., Heys, H.M. (eds.) SAC 2016. LNCS, vol. 10532, pp. 99–119. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-69453-5_6
24. National Institute of Standards and Technology: FIPS PUB 186-4 Digital Signature Standard (DSS) (2013)
25. Nguyen, P.Q., Shparlinski, I.E.: The insecurity of the elliptic curve digital signature algorithm with partially known nonces. Des. Codes Cryptogr. **30**(2), 201–217 (2003). https://doi.org/10.1023/A:1025436905711
26. Poddebniak, D., Somorovsky, J., Schinzel, S., Lochter, M., Rösler, P.: Attacking deterministic signature schemes using fault attacks. In: EuroS&P 2018. pp. 338–352. IEEE (2018). https://doi.org/10.1109/EuroSP.2018.00031
27. Pollard, J.M.: Monte Carlo methods for index computation (mod p). Math. Comput. **32**, 918–924 (1978)
28. Renes, J., Costello, C., Batina, L.: Complete addition formulas for prime order elliptic curves. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016. LNCS, vol. 9665, pp. 403–428. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49890-3_16
29. Romailler, Y., Pelissier, S.: Practical fault attack against the Ed25519 and EdDSA signature schemes. In: FDTC 2017. pp. 17–24. IEEE Computer Society (2017). https://doi.org/10.1109/FDTC.2017.12
30. Samwel, N., Batina, L.: Practical fault injection on deterministic signatures: The case of EdDSA. In: Joux, A., Nitaj, A., Rachidi, T. (eds.)

AFRICACRYPT 2018. LNCS, vol. 10831, pp. 306–321. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89339-6_17
31. Schmidt, J., Herbst, C.: A practical fault attack on square and multiply. In: Breveglieri, L., Gueron, S., Koren, I., Naccache, D., Seifert, J. (eds.) FDTC 2008. pp. 53–58. IEEE Computer Society (2008). https://doi.org/10.1109/FDTC.2008.10
32. Schmidt, J., Medwed, M.: A fault attack on ECDSA. In: Breveglieri, L., Koren, I., Naccache, D., Oswald, E., Seifert, J. (eds.) FDTC 2009. pp. 93–99. IEEE Computer Society (2009). https://doi.org/10.1109/FDTC.2009.38
33. Schmidt, J., Medwed, M.: Fault attacks on the Montgomery powering ladder. In: Rhee, K.H., Nyang, D. (eds.) ICISC 2010. LNCS, vol. 6829, pp. 396–406. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-24209-0_26
34. Schnorr, C., Euchner, M.: Lattice basis reduction: Improved practical algorithms and solving subset sum problems. Math. Program. **66**, 181–199 (1994). https://doi.org/10.1007/BF01581144
35. Shanks, D.: Class number, a theory of factorization, and genera. In: Proceedings of Symposium Mathematical Society. vol. 20, pp. 415–440 (1971). https://doi.org/10.1090/pspum/020
36. Takahashi, A., Tibouchi, M.: Degenerate fault attacks on elliptic curve parameters in OpenSSL. In: EuroS&P 2019. pp. 371–386. IEEE (2019). https://doi.org/10.1109/EuroSP.2019.00035
37. The FPLLL development team: fpylll, a Python wraper for the fplll lattice reduction library, Version: 0.5.6 (2021), https://github.com/fplll/fpylll
38. Trichina, E., Bellezza, A.: Implementation of elliptic curve cryptography with built-in counter measures against side channel attacks. In: Kaliski, B.S., Koç, K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 98–113. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-36400-5_9
39. Vasyltsov, I., Saldamli, G.: Fault detection and a differential fault analysis countermeasure for the Montgomery power ladder in elliptic curve cryptography. Math. Comput. Model. **55**(1-2), 256–267 (2012). https://doi.org/10.1016/j.mcm.2011.06.017