Boolean Exponent Splitting

Michael Tunstall[1], Louiza Papachristodoulou[2], and Kostas Papagiannopoulos[2]

[1] Rambus Cryptography Research Division,
425 Market Street, 11th Floor, San Francisco,
CA 94105, United States
michael.tunstall@cryptography.com
[2] Radboud University Nijmegen, Digital Security Group - ICIS
Mercator 1, Toernooiveld 212,
6525 EC Nijmegen, The Netherlands
louiza@cryptologio.org, kostaspap88@gmail.com

Abstract. A typical countermeasure against side-channel attacks consists of masking intermediate values with a random number. In symmetric cryptographic algorithms, Boolean shares of the secret are typically used, whereas in asymmetric algorithms the secret exponent/scalar is typically masked using algebraic properties. This paper presents a new exponent splitting technique with minimal impact on performance based on Boolean shares. More precisely, it is shown how an exponent can be efficiently split into two shares, where the exponent is the XOR sum of the two shares, typically requiring only an extra register and a few register copies per bit. Our novel exponentiation and scalar multiplication algorithms can be randomized for every execution and combined with other blinding techniques. In this way, both the exponent and the intermediate values can be protected against various types of side-channel attacks. We perform a security evaluation of our algorithms using the mutual information framework and provide proofs that they are secure against first-order side-channel attacks. The side-channel resistance of the proposed algorithms is also practically verified with test vector leakage assessment performed on Xilinx's Zynq zc702 evaluation board.

1 Introduction

Side-channel analysis as a method of extracting cryptographic keys was first presented by Kocher [Koc96], who noted that timing differences in the execution time of a modular exponentiation could be used to break instances of RSA [RSA78]. Subsequently, Kocher et al. [KJJ99] observed that the instantaneous power consumption could reveal information on intermediate states of any cryptographic algorithm, since the instantaneous power consumption has, in many cases, been shown to be proportional to the Hamming weight of the data being manipulated [BCO04]. Electromagnetic emanations around a device can also be exploited in the same way [GMO01,QS01].

In public key cryptography one typically uses countermeasures based on redundant representations to prevent side-channel leakage [Cor99,WMPW98] (referred to as blinding). To protect an exponent used in a group exponentiation one would typically add a random multiple of the order of the group to the exponent, providing a random bitwise representation of the exponent. These countermeasures can provide a strong resistance to differential power analysis (DPA), but are not convenient in some instances. As noted by Smart et al. [SOP08], the random value used to blind an exponent needs to have a bit length larger than the longest run of zeros or ones in the bitwise representation of the the order of the group. If we consider ECDSA [Nat09], for example, the bitwise representations of the orders of the groups used typically contain long runs of ones making this countermeasure undesirable. That is, to provide a randomized bitwise representation the random value used would have a bit length comparable to the exponent it is protecting, leading to a prohibitive impact on performance. While some have proposed alternative elliptic curves that avoid this problem [ML14], widely used curves will continue to have long runs of zeros and ones in the bitwise representation of the order of the group, since the modulus is typically chosen with this property to allow for the rapid computation of a modular reduction. See, for example, the elliptic curves proposed by Bos et al. [BCLN16], Hamburg [Ham15] and Bernstein [Ber06].

This paper improves upon those exponent blinding techniques by using a two-share Boolean split of the exponent. This is very surprising at first, because the arithmetic operations involved in exponentiation algorithms do not seem to combine well with bitwise XOR operations. The trick is to use exponentiation algorithms involving several (normally two) varying group elements in their main loop, such as the Montgomery powering ladder or Joye's double-and-add-always algorithm. The choice of which group element to update at each step of the algorithm can then be somewhat decoupled from the expression assigned to it. When the exponent is split into two Boolean shares, roughly speaking, one share can be used on one side of assignments while the other share is used on the other side. It is clearly not possible to achieve such a complete decoupling, but one can come close enough that the actual bits of the exponent do not need to be computed explicitly at any step of the algorithm.

1.1 Related Work.

Exponent splitting is a known countermeasure that comes in various forms (additive, multiplicative, Euclidean) [CJ01,CJ03]. It is generally avoided in practice, because it typically doubles the execution time. Negre and Plantard [NP16] proposed a regular modular exponentiation using multiplicative half-size splitting. This method, although performing 16% faster than related splitting algorithms, is still not very efficient since two exponentiations (of half-sized words) have to be performed. Moreover, it is only secure against SPA, but not against DPA, collision or template attacks. The authors propose randomizing the exponent to protect against those types of attacks, which should have additional performance overhead.

The countermeasures proposed in this paper are based on a novel exponent splitting technique with minimal performance overhead. Apart from randomizing the exponent, the algorithms can be used to hide the exponent length and prevent leakage from intermediate values. One could also consider our algorithms as an enhancement of algorithmic countermeasures to address-bit side-channel attacks (ADPA) [MDS99,MD99,IIT02], where one attempts to conduct a DPA on the addresses of registers or memory locations rather than the intermediate states of an algorithm.

May et al. [MMS01] proposed a hardware countermeasure to address-bit side-channel attacks where registers used to compute an exponentiation algorithm would be randomly renamed. This idea was extended to software by Itoh et al. [IIT03] and Izumi et al. [IISO10] who proposed exponentiation algorithms where accesses to memory locations, or registers, is randomized from one execution to another. Address-bit side-channel attacks, and the specific countermeasures, have received relatively little attention in the literature since blinding methods typically randomize the bitwise representation of an exponent, and therefore the accessed addresses [Cor99,CJ01].

1.2 Contribution.

In this paper, we present a *new countermeasure for exponent splitting*. We describe a method of splitting an exponent into *two Boolean shares*, analogous to the countermeasures that one would use for an implementation of a block cipher and similar to the countermeasures used to prevent address-bit side-channel attacks [MDS99,MD99,IIT02]. Having embedded devices as our targeted implementation, and an adversary able to get useful information from the length of the exponent or the intermediate values, we provide a number of secure algorithms against a broad range of side-channel attacks.

At the same time, the modifications that are required to a group exponentiation algorithm have negligible effect on the time required to compute the actual group exponentiation, which is a significant advantage over previous examples of exponent splitting [CJ01,CJ03].

In addition, our method can be efficiently combined with blinding techniques applied to the input to a group exponentiation algorithm, in order to prevent leakage of the intermediate values. We further demonstrate that the proposed algorithms can be used to compute an ECDSA signature where the nonce is generated in two shares that do not need to be combined. Hence, providing a convenient method for implementing a side-channel resistant instance of ECDSA with no significant impact on performance.

A further property that two of our algorithms have is that the exponent (or scalar) length can be hidden. The way the operations are handled by registers combining the Boolean splitting of the exponent can provide some algorithms with the useful property of tolerating leading zero bits.

Finally, the method of Boolean exponent splitting, cornerstone of the proposed algorithms, is evaluated in terms of security. An evaluation using the information-theoretic framework of Standaert et al. [SMY09] and a Test Vector Leakage Assessement (TVLA) by Goodwill et al. [GJJR11] are performed. We investigate the usual leakage models based on data or location leakage and show that an adversary would need either a second-order data attack or a third-order location attack to successfully break the security of our algorithms. In addition, we present for the first time a hybrid model, where data leakage is combined with location leakage, offering new exploitation opportunities. The rich interactions between data and location leakage corroborates the need for holistic countermeasures that encompass a wide spectrum of side-channel attacks.

1.3 Organization of the paper.

The rest of this paper is organized as follows: In Section 2, we describe previously proposed exponent splitting methods, their efficiency and known attacks against these countermeasures. Section 3 presents our proposed methods of exponent splitting based on an XOR operation. More precisely, Sections 3.1-3.2, present the splitting method on various regular exponentiation algorithms. We show that when XOR-splitting is applied, each algorithm is resistant against first-order side-channel attacks. In Section 3.3 we present our splitting method in the context of scalar multiplication for elliptic curves. In Section **??**, we demonstrate how this method can be used to generate an ECDSA signature where only shares of the random nonce are manipulated. An efficient algorithm to transform the Boolean share to multiplicative is also presented and shown to be first-order secure. We further provide details on how our countermeasure can be used with ECDH. Section 5 presents an extensive security evaluation of our algorithms based on formal methods and the mutual information framework. Data and location leakage diagrams are also presented here, together with the hybrid leakage attack. Section 6 discusses implementation considerations and shows the results of TVLA of a practical implementation on a Xilinx Zynq evaluation board. Finally, we conclude in Section 7.

2 Exponent Splitting Methods

The critical operation in public key cryptographic algorithms is exponentiation in a certain group $\mathbb{G}$ of order μ, where the input message $x \in \mathbb{G}$ is raised by a secret exponent κ and the result $y = x^\kappa$ is the public output of the algorithm. When implementing a group exponentiation algorithm the exponent is typically blinded by adding some random multiple of the order of the group to the exponent. Trivially, $(r\mu) + \kappa \equiv \kappa \pmod{\mu}$ for $r, \kappa \in \mathbb{Z}$ where r is random. Hence, computing $x^{\kappa + r\mu}$ is equivalent to computing x^κ. While this randomizes the bitwise representation of an exponent, the entire exponent is still equivalent to the exponent in a given group. Examples of attacks that have been proposed include analyzing a single trace (from SPA [KJJ99] to collisions in manipulated values [WvWM11,KKYH10,HKT15]) or attempting to find collisions in the random values used to then derive a (blinded) exponent [SI11].

One method that can hinder these attacks, is to split an exponent into two values whose bitwise representations are random. Then one would compute a group exponentiation where the combined effect of the two values is equivalent to that of the desired exponent. There are several methods of exponent splitting proposed by Clavier and Joye [CJ01]:

- **Additive Splitting.** For a random integer r with bit-length smaller or equal to the exponent κ, we can define $\kappa = r + (\kappa - r)$. The output of the modular exponentiation $y = x^\kappa$ in $\mathbb{G}$ can be computed by calculating $y = x^r \cdot x^{\kappa - r}$ in $\mathbb{G}$.
- **Multiplicative Splitting.** For some group $\mathbb{G}$ we can define $k' = k\, r^{-1} \bmod |\mathbb{G}|$ for some integer r. Then the exponentiation $y = x^k$ in $\mathbb{G}$ can be computed by using $y = (x^r)^{k'} \bmod |\mathbb{G}|$.

The same techniques can be applied to scalar multiplication algorithms for elliptic curves (ECs), in order to hide the secret scalar. The problem with these methods of exponent splitting is that they will typically

double the time required to compute a group exponentiation, because r is required to have a bit-length similar to the exponent. A practical attack by Feix et al. [FRV14] demonstrates that a blinded scalar can be determined if r is too small.

A further method described by Ciet and Joye [CJ03] is:

– **Euclidean Splitting.** By writing the exponent as $k = \lfloor k/r \rfloor r + k \bmod r$ and letting $s = x^r$ for some r, then $y = x^k$ can be computed by $y = s^{k'} \times x^{k \bmod r} = (x^r)^{k'} \times x^{k \bmod r}$, where $k' = \lfloor k/r \rfloor$.

The impact on the time required to compute an exponentiation is lower than the other splitting methods listed above. In fact, in [CJ03] the authors evaluated this variant applied to Shamir's double ladder to have the same cost as the 'double-and-add-always' algorithm (equivalent to the 'square-and-multiply-always' for exponentiation). Precomputation of powers of s can reduce the exponentiation cost compared to additive or multiplicative splitting. However, this method has the same constraints as adding a multiple of the group order. That is, r needs to have a bit length larger than the longest run of ones and zeros in k and will have a significant impact on performance in many cases [SOP08].

3 Boolean Exponent Splitting Methods

In this section, we propose methods of exponent splitting based on XOR operation, and how an XOR-split exponent can be applied to the Montgomery powering ladder.

3.1 Montgomery Powering Ladder

The Montgomery Powering Ladder (MPL) was originally proposed as a means of speeding up scalar multiplication over ECs and later shown to be applicable to multiplicative written Abelian groups [Mon87,JY02]. We recall the description of the MPL given by Joye and Yen [JY02]: We consider the problem of computing $y = x^\kappa$ in $\mathbb{G}$ for inputs x and κ. Let $\sum_{i=0}^{n-1} k_i\, 2^i$ be the binary expansion of κ with bit length n (for ease of expression we shall also denote this as $(k_{n-1}, \ldots, k_0)_2$ where convenient). Then, defining $L_j = \sum_{i=j}^{n-1} k_i\, 2^{i-j}$ and $H_j = L_j + 1$, we have

$$L_j = 2\, L_{j+1} + k_j = L_{j+1} + H_{j+1} + k_j - 1 = 2\, H_{j+1} + k_j - 2 \tag{1}$$

and so we obtain

$$(L_j, H_j) = \begin{cases} (2\, L_{j+1},\, L_{j+1} + H_{j+1}) & \text{if } k_j = 0\,, \\ (L_{j+1} + H_{j+1},\, 2\, H_{j+1}) & \text{if } k_j = 1\,. \end{cases} \tag{2}$$

If we consider one register containing x^{L_j} and another containing x^{H_j} then (2) implies that

$$(x^{L_j}, x^{H_j}) = \begin{cases} \left(\left(x^{L_{j+1}}\right)^2,\, x^{L_{j+1}} \cdot x^{H_{j+1}}\right) & \text{if } k_j = 0\,, \\ \left(x^{L_{j+1}} \cdot x^{H_{j+1}},\, \left(x^{H_{j+1}}\right)^2\right) & \text{if } k_j = 1\,. \end{cases}$$

Given that $L_0 = k$ one can build an exponentiation algorithm that requires two group operations per bit of the exponent. Joye and Yen give several different versions, one of which is shown in Algorithm 1. All these methods are highly regular, meaning that a deterministic sequence of operations is executed for an exponent of a given bit length.

In applying an XOR-split exponent to MPL we use one share to dictate the address accessed and the other to act as the exponent. That is, we consider (2), where the previous round may provide either (L_j, H_j) or (H_j, L_j) and the computation changed accordingly.

Let $S_{0,j} = L_j$ and $S_{1,j} = H_j$ and $\sum_{i=0}^{n-1} a_i\, 2^i$ be the binary expansion of A with bit length n (i.e. the same bit length as the exponent). Then we can use the values of a_i to dictate whether a pair of registers holds (L_j, H_j) or (H_j, L_j). Specifically, (2) can be rewritten as

$$(S_{a_j,j}, S_{\neg a_j,j}) = \begin{cases} (2\, S_{a_j,j+1},\, S_{a_j,j+1} + S_{\neg a_j,j+1}) & \text{if } k_j = 0\,, \\ (S_{a_j,j+1} + S_{\neg a_j,j+1},\, 2\, S_{\neg a_j,j+1}) & \text{if } k_j = 1\,. \end{cases} \tag{3}$$

Algorithm 1: Montgomery Ladder

Input: $x \in \mathbb{G}$, an n-bit integer $\kappa = \sum_{i=0}^{n-1} k_i \, 2^i$
Output: x^{κ}

1 $R_0 \leftarrow 1_{\mathbb{G}}$; $R_1 \leftarrow x$;
2 **for** $i = n-1$ **down to** 0 **do**
3 $\quad R_{\neg k_i} \leftarrow R_{k_i} \cdot R_{\neg k_i}$;
4 $\quad R_{k_i} \leftarrow (R_{k_i})^2$;
5 **end**
6 **return** R_0

In (3), the values of L_j and H_j are assigned to S in an order dictated by the binary expansion of A. Generating A as a random sequence of bits could provide some side-channel resistance, but does not protect the exponent.

We further consider $\sum_{i=0}^{n-1} a_i \, 2^i$ and $\sum_{i=0}^{n-1} b_i \, 2^i$ be the binary expansion of A and B, respectively, where $\kappa = A \oplus B$ of bit length n. We note that, as above, $\sum_{i=0}^{n-1} k_i \, 2^i$ is the binary expansion of κ and $k_i = a_i \oplus b_i$ for $0 \leq i < n$. Then (3) can be rewritten as

$$(S_{a_j,j}, S_{\neg a_j,j}) = \begin{cases} (2\,S_{b_j,j+1},\, S_{b_j,j+1} + S_{\neg b_j,j+1}) & \text{if } k_j = 0\,, \\ (S_{b_j,j+1} + S_{\neg b_j,j+1},\, 2\,S_{\neg b_j,j+1}) & \text{if } k_j = 1\,. \end{cases} \tag{4}$$

Rather than using the same value to control which order L_j and H_j are assigned and read, we use the bits of A to determine the order L_j and H_j are assigned, and the bits of B to determine the order they are read. The combined effect is that the order the L_j and H_j are assigned and read is dictated by the bits of κ.

We note that (4) implies

$$(x^{S_{a_j,j}}, x^{S_{\neg a_j,j}}) = \begin{cases} \left(\left(x^{S_{b_j,j+1}}\right)^2,\, x^{S_{b_j,j+1}} \cdot x^{S_{\neg b_j,j+1}} \right) & \text{if } k_j = 0\,, \\ \left(x^{S_{b_j,j+1}} \cdot x^{S_{\neg b_j,j+1}},\, \left(x^{S_{\neg b_j,j+1}}\right)^2 \right) & \text{if } k_j = 1\,. \end{cases}$$

From which we can define Algorithm 2, which operates in much the same way as the MPL, as it produces a regular sequence of multiplications and squaring operations. However, one more register is required to allow the assignment in line 5 to affect R_0 or R_1. This algorithm is the basis that we use to present the essence of Boolean-split exponent. Algorithm 2 is largely equivalent to an algorithm proposed by Izumi et al. [IISO10] where we set the multiplication in line 4 to operate in a random order as it provides a better resistance to collision attacks, as demonstrated by Kim et al. [KKYH10]. We discuss this further in Section 3.3.

The intermediate states of the registers are not randomized in Algorithm 2 and would require additional countermeasures to provide a secure implementation. For example, inexpensive solutions such as randomizing projective points [WMPW98] or Ebeid and Lambert's blinding method for RSA [EL10] can be used (see Section 6). If we assume that the values held in registers $\{R_0, R_1, R_2\}$ do not leak we can state the following:

Lemma 1. *An implementation of Algorithm 2 is resistant to first-order side-channel analysis.*

Proof. It suffices to consider each intermediate state and verify that at least one random mask is applied. Verifying this for an entire group exponentiation would be tedious, but can be simplified if we consider two rounds of Algorithm 2. That is, if we consider round m, where $0 \leq m \leq n-2$, then the following operations are performed:

Algorithm 2: Montgomery Ladder with XOR-Split Exponent I

Input: $x \in \mathbb{G}$, n-bit integers $A = \sum_{i=0}^{n-1} a_i\, 2^i$ and $B = \sum_{i=0}^{n-1} b_i\, 2^i$
Output: x^{κ} where $\kappa = A \oplus B$

```
1 R_0 ← 1_G ; R_1 ← 1_G ; R_2 ← 1_G ;
2 b' ←R {0,1} ; R_{¬b'} ← x ;
3 for i = n − 1 down to 0 do
4 |   R_2 ← R_{a_i} · R_{¬a_i} ;
5 |   R_{a_i} ← (R_{(b_i ⊕ b') ⊕ a_i})^2 ;
6 |   R_{¬a_i} ← R_2 ;
7 |   b' ← b_i ;
8 end
9 return R_{b'}
```

1. $R_2 \leftarrow R_{a_m} \cdot R_{\neg a_m}$	6. $R_2 \leftarrow R_{a_{m+1}} \cdot R_{\neg a_{m+1}}$
2. $\alpha \leftarrow b_m \oplus b'$	7. $\alpha \leftarrow b_{m+1} \oplus b_m$
3. $\beta \leftarrow \alpha \oplus a_m$	8. $\beta \leftarrow \alpha \oplus a_{m+1}$
4. $R_{a_m} \leftarrow {R_\beta}^2$	9. $R_{a_{m+1}} \leftarrow {R_\beta}^2$
5. $R_{\neg a_m} \leftarrow R_2$	10. $R_{\neg a_{m+1}} \leftarrow R_2$

Let the proposition $\mathcal{P}(n)$ be that round $n > 0$ is resistant to first-order side-channel analysis for the n-th treated bit of the exponent. If we consider the first round, we wish to show $\mathcal{P}(1)$ is true and, in the above code fragment, b' is set to a random value from $\{0, 1\}$. Then, it is easy to see that:

- the results of the operations in lines 1, 4, 5, 6, 9 and 10 are dependent on the random values $\{R_0, R_1, R_2\}$.
- the results of the operations in lines 2, 3, 7 and 8 are uniformly distributed on $\{0, 1\}$.

If we assume that $\mathcal{P}(m)$ is true for all $m \in \{1, \ldots, n\}$, then we consider $\mathcal{P}(n+1)$ where b' is set to b_n. As b_n is one share of a previously treated exponent bit, it is indistinguishable from a random value from $\{0, 1\}$. The above statements regarding the results of the operations apply. Hence, by induction we have shown $\mathcal{P}(n)$ is true for all $n > 0$. To complete the proof, we simply note that only half of the code fragment above will need to be considered in the last round. □

Remark 1. In [IIT03], the authors present the randomized addressing method (RA), in order to provide protection against ADPA and eliminate the correlation between an exponent bit and the register where the result of an operation is stored. In this work, we do not limit our countermeasure to work only against ADPA. Our goal is to perform operations on different exponent shares, in a way that an adversary would need a combination of leakages (like Higher Order DPA combined with template attacks) in order to recover the exponent.

3.2 Using Inverses

In this section we propose an algorithm more suited to groups where inversions can be readily computed. Le Duc et al. [LTT15] propose a straightforward variant of the Montgomery powering ladder that requires the computation of inverses. They note that (2) can be rewritten as

$$(L_j, H_j) = \begin{cases} (H_j - 1, L_{j+1} + H_{j+1}) & \text{if } k_j = 0\,, \\ (L_{j+1} + H_{j+1}, L_j + 1) & \text{if } k_j = 1\,. \end{cases} \tag{5}$$

From which we can define Algorithm 3. If we let $T_{0,j} = L_j$ and $T_{1,j} = H_j$, or $T_{0,j} = H_j$ and $T_{1,j} = L_j$ and store the ordering in another variable we can rewrite (5) as

$$(T_{0,j}, T_{1,j}) = \begin{cases} (L_j, H_j) & \text{if } k_j = 0 \\ (H_j, L_j) & \text{if } k_j = 1 \end{cases} = \begin{cases} (L_{j+1} + H_{j+1}, L_j - 1) & \text{if } k_j = 0\,, \\ (L_{j+1} + H_{j+1}, L_j + 1) & \text{if } k_j = 1\,. \end{cases} \tag{6}$$

From which we can define Algorithm 4.

Algorithm 3: Variant with Inverses I

Input: $x \in \mathbb{G}$, an n-bit integer $\kappa = \sum_{i=0}^{n-1} k_i\, 2^i$
Output: x^κ

$R_0 \leftarrow 1_{\mathbb{G}}$; $R_1 \leftarrow x$;
$U_0 \leftarrow x^{-1}$; $U_1 \leftarrow x$;
for $i = n-1$ **down to** 0 **do**
  $R_{\neg k_i} \leftarrow R_{k_i} \cdot R_{\neg k_i}$;
  $R_{k_i} \leftarrow R_{\neg k_i} \cdot U_{k_i}$;
end
return R_0

Algorithm 4: Variant with Inverses II

Input: $x \in \mathbb{G}$, an n-bit integer $\kappa = \sum_{i=0}^{n-1} k_i\, 2^i$
Output: x^κ

$R_0 \leftarrow 1_{\mathbb{G}}$; $R_1 \leftarrow x$;
$U_0 \leftarrow x^{-1}$; $U_1 \leftarrow x$;
for $i = n-1$ **down to** 0 **do**
  $R_0 \leftarrow R_0 \cdot R_1$;
  $R_1 \leftarrow R_0 \cdot U_{k_i}$;
end
return $R_{\neg k_0}$

Following the previous notation, we notice that $T_{0,j}$ should contain the sum of the registers in the previous round[3]. Therefore, (6) can be rewritten as follows:

$$(T_{0,j}, T_{1,j}) = \begin{cases} (T_{b',j+1} + T_{\neg b',j+1}, T_{0,j} - 1) & \text{if } k_j = b' = 0\,, \\ (T_{b',j+1} + T_{\neg b',j+1}, T_{0,j} + 1) & \text{if } k_j = b' = 1\,. \end{cases} \tag{7}$$

We note that to treat k_{j+1}, $b' = k_j$. However, if we let $k_j = a_j \oplus b_j$, for $a_j, b_j \in \{0,1\}$ and $h = a_j \oplus b_j \oplus b_{j-1}$, we can modify (7) as follows:

$$(T_{0,j}, T_{1,j}) = \begin{cases} (T_{\neg h,j+1} + T_{h,j+1}, T_{0,j} - 1) & \text{if } a_j = b_j\,, \\ (T_{\neg h,j+1} + T_{h,j+1}, T_{0,j} + 1) & \text{if } a_j = \neg b_j\,. \end{cases} \tag{8}$$

By using the above equations as exponents of x, we can define Algorithm 5.

Algorithm 5: Montgomery Ladder with XOR-Split Exponent II

Input: $x \in \mathbb{G}$, n-bit integers $A = \sum_{i=0}^{n-1} a_i\, 2^i$ and $B = \sum_{i=0}^{n-1} b_i\, 2^i$, $r \in_R \mathbb{Z}$
Output: x^κ where $\kappa = A \oplus B$

$R_0 \leftarrow 1_{\mathbb{G}}$; $R_1 \leftarrow 1_{\mathbb{G}}$; $U_0 \leftarrow x$; $U_1 \leftarrow x^{-1}$;
$b' \xleftarrow{R} \{0,1\}$; $R_{\neg b'} \leftarrow x$;
for $i = n-1$ **down to** 0 **do**
  $R_0 \leftarrow R_{b_i \oplus b'} \cdot R_{(b_i \oplus b') \oplus a_i}$;
  $R_1 \leftarrow R_0 \cdot U_{b_i}$;
  $b' \leftarrow b_i$;
end
return $R_{b'}$

[3] The algorithms are left-to-right, so $j+1$ indicates the round preceding j.

Algorithm 5 follows the same sequence of instructions with the MPL. Its correctness can be verified by the fact that at every round the difference $R_0/R_1 = x$ or $R_1/R_0 = x$, as for the usual ladder step. The advantage of Algorithm 5 compared to Algorithm 2, and consequently previously proposed algorithms by Izumi et al. [IISO10], is the elimination of the auxiliary register R_2. Instead, the auxiliary registers U_0, U_1 manipulate the known fixed value x or x^{-1} for computational purposes, and they do not require additional computational power or updates when the algorithm is executed.

As previously, if we assume that the values held in registers $\{R_0, R_1\}$ do not leak we can state the following:

Lemma 2. *An implementation of Algorithm 5 is resistant to first-order side-channel analysis.*

Proof. It suffices to consider each intermediate state and verify that at least one random mask is applied. Verifying this for an entire group exponentiation would be tedious, but can be simplified if we consider two rounds of Algorithm 5. That is, if we consider round m, where $0 \leq m \leq n-2$, then the following operations are performed:

1. $\alpha \leftarrow b_m \oplus b'$	5. $\alpha \leftarrow b_{m+1} \oplus b_m$
2. $\beta \leftarrow \alpha \oplus a_m$	6. $\beta \leftarrow \alpha \oplus a_{m+1}$
3. $R_0 \leftarrow R_\alpha \cdot R_\beta$	7. $R_0 \leftarrow R_\alpha \cdot R_\beta$
4. $R_1 \leftarrow R_0 \cdot U_{b_m}$	8. $R_1 \leftarrow R_0 \cdot U_{b_{m+1}}$

Let the proposition $\mathcal{P}(n)$ be that round $n > 0$ is resistant to first-order side-channel analysis for the n-th treated bit of the exponent. If consider the first round, we wish to show $\mathcal{P}(1)$ is true and, in the above code fragment, b' is set to a random value from $\{0, 1\}$. Then, it is easy to see that:

- the results of the operations in lines 3, 4, 7 and 8 are dependent on the random values $\{R_0, R_1\}$.
- the results of the operations in lines 1, 2, 5 and 6 are uniformly distributed on $\{0, 1\}$.

If we assume that all $\mathcal{P}(m)$ is true for $m \in \{1, \ldots, n\}$, then we consider $\mathcal{P}(n+1)$ where b' is set to b_n. As b_n is one share of a previously treated exponent bit, it is indistinguishable from a random value from $\{0, 1\}$. The above statements regarding the results of the operations apply. Hence, by induction we have shown $\mathcal{P}(n)$ is true for all $n > 0$. To complete the proof, we simply note that only half of the code fragment above will need to be considered in the last round. □

3.3 Boolean Scalar Splitting

In the above, we define group exponentiations applicable to any multiplicatively written group $\mathbb{G}$. However, specific groups may have particular characteristics that means the algorithms above are not suitable as described. In this section, we discuss the algorithms in the context of a group formed from the points on an elliptic curve (EC). We define the EC $\mathcal{E}$ over a finite field $\mathbb{F}_q$, for a large prime q. $\mathcal{E}$ consists of points (x, y), with x, y in $\mathbb{F}_q$, that satisfy, for example, the short Weierstraß equation

$$\mathcal{E} : y^2 = x^3 + a\,x + b$$

with $a, b \in \mathbb{F}_q$, and the point at infinity denoted $\boldsymbol{O}$. The set $\mathcal{E}(\mathbb{F}_q)$ is defined as $\mathcal{E}(\mathbb{F}_q) = \{(x, y) \in \mathcal{E} \,|\, x, y \in \mathbb{F}_q\} \cup \{\boldsymbol{O}\}$, where $\mathcal{E}(\mathbb{F}_q)$ forms an Abelian group under the chord-and-tangent rule and $\boldsymbol{O}$ is the identity element. Alternative equations with different representations of a neutral element are also used in cryptographic algorithms, such as Edwards curves [Edw07,BL09] and Montgomery curves [Mon87]. The scalar multiplication of a given point is a group exponentiation in $\mathcal{E}$ that uses elliptic curve arithmetic, i.e. addition between points or scalar multiplication $[\kappa]\boldsymbol{P}$ for some integer $\kappa < |\mathcal{E}|$, and is an important part of many cryptographic algorithms.

The algorithms presented above cannot be securely implemented as described because of the neutral element. In the short Weierstraß example, the neutral element $1_{\mathbb{G}}$ is represented in $\mathcal{E}$ as the point at infinity

$\boldsymbol{O}$ and cannot be manipulated in a regular way. That is, one would typically be obliged to test for a numerical representation of $\boldsymbol{O}$ and conduct a different operation if it is detected. In practice, one would implement the algorithm such that the most significant bit (assumed to be set to one) is already treated by the pre-processing. For example, Algorithm 2 can be implemented as shown in Algorithm 6, and Algorithm 5 as shown in Algorithm 7.

Algorithm 6: Montgomery Ladder with XOR-Split Scalar on an EC

Input: $\mathcal{E}$, $\mathbb{F}_q$, $\boldsymbol{P} \in \mathcal{E}$, n-bit integers $A = \sum_{i=0}^{n-1} a_i\, 2^i$ and $B = \sum_{i=0}^{n-1} b_i\, 2^i$
Output: $\boldsymbol{Q} = [\kappa]\boldsymbol{P}$ where $\kappa = A \oplus B$

$\boldsymbol{R}_0 \leftarrow \boldsymbol{P}$; $\boldsymbol{R}_1 \leftarrow \boldsymbol{P}$; $\boldsymbol{R}_2 \leftarrow \boldsymbol{P}$;
$b' \xleftarrow{R} \{0,1\}$;
$\boldsymbol{R}_{\neg b'} \leftarrow 2\,\boldsymbol{P}$;
for $i = n-2$ **down to** 0 **do**
  $\boldsymbol{R}_2 \leftarrow \boldsymbol{R}_{a_i} + \boldsymbol{R}_{\neg a_i}$;
  $\boldsymbol{R}_{a_i} \leftarrow 2\,\boldsymbol{R}_{(b_i \oplus b') \oplus a_i}$;
  $\boldsymbol{R}_{\neg a_i} \leftarrow \boldsymbol{R}_2$;
  $b' \leftarrow b_i$;
end
return $\boldsymbol{R}_{b'}$

Algorithm 7: Montgomery Ladder with XOR-Split Scalar II on an EC

Input: $\mathcal{E}$, $\mathbb{F}_q$, $\boldsymbol{P} \in \mathcal{E}$, n-bit integers $A = \sum_{i=0}^{n-1} a_i\, 2^i$ and $B = \sum_{i=0}^{n-1} b_i\, 2^i$
Output: $\boldsymbol{Q} = [\kappa]\boldsymbol{P}$ where $\kappa = A \oplus B$

$\boldsymbol{R}_0 \leftarrow \boldsymbol{P}$; $\boldsymbol{R}_1 \leftarrow \boldsymbol{P}$;
$\boldsymbol{U}_0 \leftarrow \boldsymbol{P}$; $\boldsymbol{U}_1 \leftarrow -\boldsymbol{P}$;
$b' \xleftarrow{R} \{0,1\}$;
$\boldsymbol{R}_{\neg b'} \leftarrow 2\,\boldsymbol{P}$;
for $i = n-2$ **down to** 0 **do**
  $\boldsymbol{R}_0 \leftarrow \boldsymbol{R}_{b_i \oplus b'} + \boldsymbol{R}_{(b_i \oplus b') \oplus a_i}$;
  $\boldsymbol{R}_1 \leftarrow \boldsymbol{R}_0 + \boldsymbol{U}_{b_i}$;
  $b' \leftarrow b_i$;
end
return $\boldsymbol{R}_{b'}$

As previously, if we assume that the values held in registers $\{R_0, R_1, R_2\}$ do not leak we can state the following:

Corollary 1. *Lemma 1 implies that an implementation of Algorithm 6 is resistant to first-order side-channel analysis.*

Corollary 2. *Lemma 2 implies that an implementation of Algorithm 7 is resistant to first-order side-channel analysis.*

4 Applying Exponent Splitting to Elliptic Curve Cryptography

In this section, we describe how one could apply the above algorithms to ECDSA and ECDH, as the particular security requirements of these algorithms merit further discussion.

4.1 Applying Exponent Splitting to ECDSA

For ECDSA, a given base point $\boldsymbol{P} = (x, y)$ for an EC $\mathcal{E}$ over $\mathbb{F}_q$, with private key d and hash function h, the signer that wants to sign a message m picks a random $\kappa < |\mathcal{E}| = n$ (where $n = |\mathcal{E}|$ is the order of the curve) and computes

$$r \xleftarrow{x} [\kappa]\boldsymbol{P} \quad \text{and} \quad s \leftarrow \kappa^{-1}\,(h(m) + d\,r) \bmod |\mathcal{E}|\,.$$

We denote the extraction of the x-coordinate of a point and its assignment to a variable by $\xleftarrow{x}$. The signature of m is the pair: $\{r, s\}$. We note that the security of this signature scheme relies on the random value κ remaining unknown to an attacker. Moreover, the nonce κ should be used only once and randomly generated for every new signature, otherwise the private key d can be trivially derived [Nat09].

Securing the Scalar Multiplication: The scalar multiplication $[\kappa]\boldsymbol{P}$ during the computation of r is a critical operation in ECDSA from a side-channel point of view, because an attacker only needs to derive a small number of bits of κ to obtain the secret key d. For example, a lattice-based analysis can reveal the private key from the knowledge of some bits of the scalar κ [HS01]. In practice, one could use the knowledge of one bit from around 30 signatures or eleven bits from one signature [NNTW05]. Similarly, one needs to prevent an adversary from determining the bit length of the scalar used to prevent a lattice attack.

When implementing a scalar multiplication, one could use any of the algorithms described in the previous section, with extra care required to prevent the bit length of the scalar from leaking. Algorithms 6 and 7 require that the most-significant bit of the scalar is set to one and need further modification to ensure that the bit length of the scalar is not revealed.

In implementing Algorithms 6 or 7, one could add a multiple of the order of the group to the scalar to hide its length [Cor99]. While this would seem to make the algorithms redundant, one could choose a small multiple of the order of the group without considering the longest run of ones or zeros in the bitwise representation of the order. Indeed, one could multiply the order of the group by a power of two such that increasing the bit length is impossible or fixed to a one bit increase. That is, ensure that the carries produced by adding the multiple of the order of the group do not extend the bit length or always extend the bit length. Given the long runs of ones or zeros typically seen in the bitwise representation of the order of many groups formed from the points of an EC, the increase in bit length should be very small [Nat09,BCLN16,Ham15,Ber06]. For example, if we consider P192 defined by NIST [Nat09], where the elliptic curve is defined over $\mathbb{F}_p$, with $p = 2^{192} - 2^{64} - 1$, then the 128 most-significant bits of p are set to one. If twice the order is added to a nonce then a carry from the most-significant bits will be produced with an overwhelming probability, thus always producing a nonce of 194 bits. However, in some cases, one may wish to ensure that the carry has been produced to prevent a theoretical reduction in the security of a system. That is, one could design attacks around an oracle that indicated the value of the carry, but such an oracle is unlikely to exist.

Alternatively, one could add some more logical functions to allow for the most significant bit to be set to zero. In Algorithms 8 and 9 we show how this could be done for Algorithms 6 and 7, respectively. In both cases we use the variable m to be set to three while the most-significant bits of the scalar, $A \oplus B$, are set to zero, allowing the output of operations to be diverted into $\boldsymbol{R}_2$. In Algorithm 9, we do this for both operations in the loop and keep the initial state of $\boldsymbol{R}_0$ and $\boldsymbol{R}_1$ until after the first bit set to one is treated. Then Algorithm 9 proceeds in the same way as Algorithm 7. Algorithm 8 starts in the same way, using m to divert the output of the operations into $\boldsymbol{R}_2$. When the most-significant bit of the scalar, $A \oplus B$, set to one is treated the output stored in $\boldsymbol{R}_2$ is copied to either $\boldsymbol{R}_0$ or $\boldsymbol{R}_1$. Then Algorithm 8 proceeds in the same way as Algorithm 6.

As previously, if we assume that the values held in registers $\{R_0, R_1, R_2\}$ do not leak we can state the following:

Proposition 1. *An implementation of Algorithm 8 is resistant to first-order side-channel analysis.*

Proof. We note that the proof is almost identical to that given for Lemma 1. In addition, we can note that h and m are set initially and remain unchanged until the first bit set to one is encountered. Then h and m are set to zero and have no further impact on the computation. Given that h and m only change at one point that will be fixed for a given exponent it cannot be attacked via a first-order side-channel analysis. □

Proposition 2. *An implementation of Algorithm 9 is resistant to first-order side-channel analysis.*

Proof. We note that the proof is almost identical to that given for Lemma 2. In addition, we can note that s and m are set initially and remain unchanged until the first bit set to one is encountered. Then s and m are set to zero and have no further impact on the computation. Given that s and m only change at one point that will be fixed for a given exponent it cannot be attacked via a first-order side-channel analysis. □

We note that Algorithms 8 and 9 require a careful implementation. The computation of h in line 6 of Algorithm 8 and s in line 9 of Algorithm 9 combine the shares of the exponent. However, once the most-significant bit of the exponent is treated each shared is zeroed with a logical-AND operation before the

Algorithm 8: Montgomery Ladder with Fixed-Length XOR-Split Scalar on an EC

Input: $\mathcal{E}$, $\mathbb{F}_q$, $\boldsymbol{P} \in \mathcal{E}$, ℓ-bit integers $A = \sum_{i=0}^{n-1} a_i\, 2^i$ and $B = \sum_{i=0}^{n-1} b_i\, 2^i$ where $n \geq \ell$
Output: $\boldsymbol{Q} = [\kappa]\boldsymbol{P}$ where $\kappa = A \oplus B$

$\boldsymbol{R}_0 \leftarrow \boldsymbol{P}$; $\boldsymbol{R}_1 \leftarrow \boldsymbol{P}$; $\boldsymbol{R}_2 \leftarrow \boldsymbol{P}$;
$b' \xleftarrow{R} \{0,1\}$; $h \leftarrow 1$; $m \leftarrow 3$;
for $i = n-1$ **down to** 0 **do**
  $\boldsymbol{R}_2 \leftarrow \boldsymbol{R}_{a_i} + \boldsymbol{R}_{\neg a_i}$;
  $\boldsymbol{R}_{((1+a_i)\vee m)-1} \leftarrow 2\,\boldsymbol{R}_{(h\oplus b_i \oplus b')\oplus a_i}$;
  $h = m \wedge 1$;
  $m = 3\,((h \wedge \neg a_i) \oplus (h \wedge b_i))$;
  $\boldsymbol{R}_{((1+\neg a_i)\vee m)-1} \leftarrow \boldsymbol{R}_2$;
  $b' \leftarrow b_i$;
end
return $\boldsymbol{R}_{b'}$

Algorithm 9: Montgomery Ladder with Fixed-Length XOR-Split Scalar II on an EC

Input: $\mathcal{E}$, $\mathbb{F}_q$, $\boldsymbol{P} \in \mathcal{E}$, ℓ-bit integers $A = \sum_{i=0}^{n-1} a_i\, 2^i$ and $B = \sum_{i=0}^{n-1} b_i\, 2^i$ where $n \geq \ell$
Output: $\boldsymbol{Q} = [\kappa]\boldsymbol{P}$ where $\kappa = A \oplus B$

$\boldsymbol{R}_0 \leftarrow \boldsymbol{P}$; $\boldsymbol{R}_1 \leftarrow \boldsymbol{P}$;
$\boldsymbol{U}_0 \leftarrow \boldsymbol{P}$; $\boldsymbol{U}_1 \leftarrow -\boldsymbol{P}$;
$b' \xleftarrow{R} \{0,1\}$; $h \leftarrow 1$; $m \leftarrow 3$;
$\boldsymbol{R}_{\neg b'} \leftarrow 2\,\boldsymbol{P}$;
for $i = n-1$ **down to** 0 **do**
  $\boldsymbol{R}_{(1\vee m)-1} \leftarrow \boldsymbol{R}_{b_i \oplus b'} + \boldsymbol{R}_{(b_i \oplus b')\oplus a_i}$;
  $\boldsymbol{R}_{(2\vee m)-1} \leftarrow \boldsymbol{R}_0 + \boldsymbol{U}_{b_i}$;
  $b' \leftarrow (b_i \wedge \neg n) \oplus (b' \wedge n)$;
  $s = ((m \wedge \neg a_i) \oplus (m \wedge b_i))$;
  $m = 3\,s$;
end
return $\boldsymbol{R}_{b'}$

shares are combined (prior to this any treated bits are always zero). A detailed discussion is beyond the scope of this paper, but one needs to ensure that an implementation does not inadvertently combine these values [BGG+14].

Using an ECDSA Nonce as Two Shares: In this section we describe how one could generate an ECDSA signature where the nonce is only present as two shares. That is, one could generate two random values to represent the two shares of the nonce, and the actual value of the nonce would never appear in the signing device.

We assume that the first half of the signature r has been generated using one of the scalar multiplication algorithms given above, using shares A and B where the nonce $\kappa = A \oplus B$. To compute the inverse of κ modulo $|\mathcal{E}|$ we need to change the way that κ is shared between two variables without revealing any information on κ.

Goubin defined a secure means of modifying two shares such that the secret value is no longer given by the XOR of the shares, but by subtracting one from the other [Gou01]. That is, the relationship $\kappa = A \oplus B$ becomes $\kappa \equiv A' - B' \pmod{|\mathcal{E}|}$. The essential observation of Goubin was that the function

$$\Phi(a,b) : \mathbb{Z}^2 \longrightarrow \mathbb{Z} : a, b \longmapsto (a \oplus b) + b \tag{9}$$

is affine over $\mathbb{F}_2$. If we consider α, β as Boolean shares of x, where $x = \alpha \oplus \beta$, then, as defined by Goubin,

$$x + \alpha = \beta \oplus \Phi(\beta, \gamma) \oplus \Phi(\beta, \gamma \oplus \alpha)\ , \tag{10}$$

for a random variable γ. This is not side-channel resistant in $\mathbb{Z}$ because of the carries produced by the addition operations. One needs to use a group $\mathbb{Z}_{2^k}$, for some $k \in \mathbb{Z}_+$, and have γ be a random value from $\mathbb{Z}_{2^k}$. However, we wish to change a Boolean share into an additive share in the group $\mathbb{Z}_{|\mathcal{E}|}$ where (10) is not valid.

we can modify the shares such that Goubin's mask conversion can be used adding a random multiple of the order of the group as proposed by Coron for other algorithms [Cor99]. One can generate a random integer $\ell \in \mathbb{Z}_{2^n}$ for some convenient bit length n. The Boolean shares $(A = \kappa \oplus B, B)$ can be modified to (A', B') where

$$(A', B') = (A \oplus \ell|\mathcal{E}|, \ell|\mathcal{E}| + B)\ . \tag{11}$$

We note that this gives $A' = \kappa \oplus (\ell|\mathcal{E}| + B)$. If we consider the bit length of $|\mathcal{E}|$ is m then the above modified shares can be used with (10) using group $\mathbb{Z}_{2^{n+m}}$ to change the shares (A', B') to (A'', B'') where

$$(A'', B'') = (\kappa + \ell|\mathcal{E}| + B, \ell|\mathcal{E}| + B) \ . \tag{12}$$

Then A'' and B'' can be reduced modulo $|\mathcal{E}|$ to provide additively split shares of κ in $\mathbb{Z}_{|\mathcal{E}|}$. We can generate a random integer $\omega < |\mathcal{E}|$ and change the shares to a multiplicative split (A''', B''') where

$$(A''', B''') = (\omega\, A'' - \omega\, B'' \bmod |\mathcal{E}|, \omega) \ . \tag{13}$$

We note that this gives $A''' \equiv \omega\kappa \pmod{|\mathcal{E}|}$. The second part of the signature can be generated from r by computing

$$s \leftarrow B''' \left(A'''^{-1} \left(h(m) + d\, r \right) \right) \bmod |\mathcal{E}| \, .$$

Thus, using the above, one can generate an ECDSA signature where the nonce κ only exists as two shares. An explicit algorithm requiring 16 operations is given in Algorithm 10.

Algorithm 10: Secure Boolean-to-Multiplicative Masking

Input: $A, B \in \mathbb{Z}_{2^n}$, where $\kappa = A \oplus B$, $\kappa \in Z_{|\mathcal{E}|}$ and n is the bit length of $|\mathcal{E}|$, random value $\gamma \in \{0, \ldots, 2^n - 1\}$, random value $\ell \in \{0, \ldots, 2^m - 1\}$ where m is the bit length of one computer word and random value $\omega \in \{1, \ldots, |\mathcal{E}|\}$.

Output: $\kappa\, \omega \bmod |\mathcal{E}|$.

1 $x_1 \leftarrow \ell \times |\mathcal{E}|$;
2 $B' \leftarrow x_1 + B$;
3 $x_2 \leftarrow B' \oplus B$;
4 $A' \leftarrow A \oplus x_2$;
5 $x_3 \leftarrow A' \oplus \gamma$;
6 $x_4 \leftarrow x_3 + \gamma$;
7 $x_5 \leftarrow B' \oplus \gamma$;
8 $x_6 \leftarrow A' \oplus x_5$;
9 $x_7 \leftarrow x_6 + x_5$;
10 $x_8 \leftarrow A' \oplus x_4$;
11 $x_9 \leftarrow x_7 \oplus x_8$;
12 $B'' \leftarrow B' \bmod |\mathcal{E}|$;
13 $A'' \leftarrow x_9 \bmod |\mathcal{E}|$;
14 $x_{10} \leftarrow B'' \times \omega \bmod |\mathcal{E}|$;
15 $x_{11} \leftarrow A'' \times \omega \bmod |\mathcal{E}|$;
16 $x_{12} \leftarrow x_{11} - x_{10} \bmod |\mathcal{E}|$;
17 **return** x_{12} ;

Lemma 3. *An implementation of Algorithm 10 is resistant to first-order side-channel analysis.*

Proof. It suffices to consider each intermediate state and verify that at least one random mask is applied. We consider the intermediate states as:

1. $\ell \times |\mathcal{E}|$
2. $\ell \times |\mathcal{E}| + B$
3. $(\ell \times |\mathcal{E}| + B) \oplus B$
4. $(\ell \times |\mathcal{E}| + B) \oplus \kappa$
5. $(\ell \times |\mathcal{E}| + B) \oplus \kappa \oplus \gamma$
6. $((\ell \times |\mathcal{E}| + B) \oplus \kappa \oplus \gamma) + \gamma$
7. $(\ell \times |\mathcal{E}| + B) \oplus \gamma$
8. $(\ell \times |\mathcal{E}| + B) \oplus \kappa \oplus (\ell \times |\mathcal{E}| + B) \oplus \gamma$
9. $((\ell \times |\mathcal{E}| + B) \oplus \kappa \oplus (\ell \times |\mathcal{E}| + B) \oplus \gamma) + ((\ell \times |\mathcal{E}| + B) \oplus \gamma)$
10. $(((\ell \times |\mathcal{E}| + B) \oplus \kappa \oplus (\ell \times |\mathcal{E}| + B) \oplus \gamma) + ((\ell \times |\mathcal{E}| + B) \oplus \gamma)) \oplus (\ell \times |\mathcal{E}| + B) \oplus \kappa$
11. $\kappa + \ell \times |\mathcal{E}| + B$
12. $B \bmod |\mathcal{E}|$
13. $\kappa + B \bmod |\mathcal{E}|$
14. $B\, \omega \bmod |\mathcal{E}|$
15. $\kappa\, \omega + B\, \omega \bmod |\mathcal{E}|$
16. $\kappa\, \omega \bmod |\mathcal{E}|$

Then, it is easy to see that:

– the results of the operations in lines 1, 2, 3, 7, 12 and 14 are not dependent on the secret κ and cannot be attacked by a first-order side-channel analysis.

– the results of the operations in lines 4, 5, 6, 8, 9, 10, 11, 13, 15 and 16 are dependent on the secret κ but are masked by at least one random value and, therefore, cannot be attacked by a first-order side-channel analysis.

Hence, Algorithm 10 is resistant to first-order side-channel analysis. □

We note that in the above proof the security of the output of line 10 may not be immediately apparent. We refer the reader to Goubin [Gou01] for a thorough explanation.

4.2 Applying Exponent Splitting to ECDH

For ECDH, the basic primitive is to take a public key $\boldsymbol{Q}$ and compute a scalar multiplication using a private key. This operation has different security requirements to ECDSA. The bit length of the private key does not need to be protected but the input point $\boldsymbol{Q}$ could be controlled by an adversary. The exponent splitting methods detailed in this paper do not modify the intermediate states generated and one would expect that randomizing projective points would be adequate to provide a secure solution [WMPW98]. However, such multiplicative masking can be problematic if an attacker can choose and input that could produce a point with a coordinate set to zero, which cannot be blinded using a multiplication [Gou03]. Hence, one would need to combine our algorithms with Coron's countermeasures [Cor99] and add a small multiple of the order of the group to the private key before it is used. The bit length of the multiplier needs to be chosen such that an attacker cannot predict the location of a zero-coordinate with sufficient reliability to make is visible in a side-channel attack. Something like 16 bit may be sufficient, depending on the signal-to-noise ratio of the platform. The advantage of combining these countermeasures is that one does not need to consider the longest runs of ones or zeros in the order of the group.

5 Security Evaluation

In this section, we discuss the security of the algorithms presented previously, first by making a comparison with the state-of-the-art algorithms and then by showing the resistance of our algorithms against template attacks. Furthermore, we provide a security evaluation of Algorithm 2, proposed in this paper, in terms of noisy leakage security and its resistance to multiple forms of side-channel attack. We note that similar analyses can be carried out for all exponent splitting variants presented in this work.

5.1 Theoretical Comparison with the State-of-the-Art Algorithms

In this subsection, we compare our proposed algorithms with a selection of algorithms discussed in the previous sections and summarize our observations in Table 1.

The first block of algorithms in Table 1, contain exponentiation algorithms using the Montgomery power ladder without splitting the exponent (Algorithm 1), with additive splitting or with variations of XOR-splitting (Algorithms 2, 3, 4). Multiplicative or Euclidean splitting are not included in this table, because in terms of security they have the same side-channel resistance as an algorithm with additive splitting. In terms of performance, the number of operations is the similar, unless the values $s^{k\prime}$ are precomputed and stored in memory. The second block of algorithms summarizes the behavior of the corresponding scalar multiplication algorithms [4].

We note that none of the algorithms in their current form can prevent leakage from observing the intermediate values. However, intermediate values can be blinded with a random value at the cost of an inversion (or subtraction for elliptic curves).

[4] We do not count XORs, which can be implemented almost for "free" compared to the cost of multiplications (M), squaring operations (S) and modular inversions (I) in the chosen field or point additions (A) and doubling operations (D) on an elliptic curve. The subtraction of points on an elliptic curve has the same cost as an addition, so we do not count them separately.

Algorithm	♯operations	♯registers	Hide length	ADPA	Interm. Values
Algorithm 1	$n \cdot M + n \cdot S$	2	✗	✗	✗
Clavier-Joye [CJ01]	$2(n \cdot M + n \cdot S)$	2	✗	✗	✗
Algorithm 2	$n \cdot M + n \cdot S$	3	✓	✓	✗
Algorithms 3–4	$2n \cdot M$	4	✓	✓	✗
Algorithm 5	$2n \cdot M$	4	✓	✓	✗
Algorithm 6	$(n-1) \cdot A + (n-1) \cdot D$	3	✓	✓	✗
Algorithm 7	$2 \cdot (n-1) \cdot A$	4	✓	✓	✗
Itoh et al. [IIT03] Alg. 8	$(n-1) \cdot D + (n-1) \cdot A + 1 \cdot I$	3	✗	✓	✗
Izumi et al. [IISO10] Alg. 2	$(n-1) \cdot D + (n-1) \cdot A$	3	✗	✓	✗
Algorithm 8	$n \cdot D + n \cdot A$	3	✓	✓	✗
Algorithm 9	$2 \cdot n \cdot A$	4	✓	✓	✗

Table 1. Comparison Table

5.2 Security against Template Attacks

Template Attacks in general are a powerful attack technique because they take advantage of most of the information available in the traces that are measured [CRR02]. An attacker is assumed to possess a device similar to the device under attack, and can build templates for some instructions knowing the power consumption characteristics on a device.

In a template-based DPA, an attacker matches the templates based on different key hypotheses with the recorded power traces [OM07]. The templates that have the best match indicate the key. This type of attack is the best attack in an information theoretic sense and, in a masking scheme like ours, it would work if one could build templates for each share a_i and b_i. Simultaneous knowledge of the two shares would be necessary because, for every possible value of the key bit, there are two different possibilities for a_i, b_i that could give the same result, as follows:

$$k_i = 1 \Rightarrow (a_i = 0 \quad \text{and} \quad b_i = 1) \quad \text{or} \quad (a_i = 1 \quad \text{and} \quad b_i = 0)$$

$$k_i = 0 \Rightarrow a_i = b_i = 0 \quad \text{or} \quad a_i = b_i = 1$$

In operations where $b\prime$ is also involved, as in line 5 of Algorithm 2, line 4 of Algorithm 5 etc., we need to store the b_{i-1} value from the templates of the previous round. This procedure seems quite complex and unlikely to succeed in devices with a small signal-to-noise ratio.

Another powerful template attack technique is Online Template Attacks (OTA) [BCP+14]. OTA uses templates of multiples of a known base point $k\boldsymbol{P}$, for some $k \in \mathbb{Z}$, and compares them with the result of a specific operation, a doubling operation for example, for every round of the algorithm. Every unprotected implementation of scalar multiplication is vulnerable to OTA, unless the coordinates of the base point are randomized[5]. The algorithms presented in this paper are secure against OTA, because the doubling (or not) of a specific point, does not reveal the bit of the scalar k_i. We demonstrate this claim, using two algorithms as case studies, one for the case of general group exponentiation and one for elliptic curves. The extension to other algorithms presented in this paper is straightforward.

Algorithm 2: In line 4 we have

$$R_{a_i} \leftarrow \left(R_{(b_i \oplus b') \oplus a_i}\right)^2 n\,,$$

where a squaring is performed and we could make templates for the potential values R_0^2 and R_1^2, in order to find out which operation is most probable. Let us assume that there is a pattern match of 99% with the value R_0^2. Even if we know with high probability which value is calculated at this step, we do not know which register is going to be used in the next step using OTA. Moreover, the index 0 can be derived from several combinations of a_i, b_i, b'. Namely, if $a_i = b_i = 0, b' = 0$, or $a_i = b_i = 1, b' = 0$ or $a_i = 0, b_i = 1, b' = 1$ or $a_i = 1, b_i = 0, b' = 1$. If the previous shares are known, therefore b' is known, there are still two possible

[5] Although OTA is initially presented for the case of elliptic curves, the attack can be used against any exponentiation algorithm, as long as templates for certain exponent values can be created.

values for a_i and b_i that can give the same result with equal probability. The success probability of OTA is then 1/2 which is no better than a random guess.

Algorithm 6: Let us assume that templates on a doubling operation show that in line 6 of Algorithm 8, we have $2\boldsymbol{R}_1$. The possible values of a_i, b_i, b' that could result in 1 are: $(a_i = b_i = 0, b' = 1)$, or $(a_i = b_i = 1, b' = 1)$ or $(a_i = 0, b_i = 1, b' = 0)$ or $(a_i = 1, b_i = 0, b' = 0)$. The success probability of OTA is again 1/2 if the previous shares are known. We note here, that in both cases, knowledge of the previous bit is not enough to give a success probability of 1/2, but knowledge of at least one of the shares is required.

5.3 Mutual Information-based Evaluation of Boolean Exponent Splitting

Having established that the proposed exponent splitting algorithms are probing-secure against first-order side-channel attacks, we proceed to analyze the noise amplification stage of the proposed countermeasure. Analytically, we perform an evaluation of Boolean exponent splitting (as described by Algorithm 2) using the information-theoretic framework of Standaert et al. [SMY09]. Analogous approaches can be conducted for all exponent splitting algorithms, yielding very similar results. Our analysis considers two sources of leakage, namely data-based leakage and location-based leakage (also known as address leakage). Using these two leakage sources, we demonstrate three possible attack paths against Algorithm 5, covering all possible combinations between leakage sources. Thus we show the noise amplification stage when only data-based leakage is exploited (data attack), when only location-based leakage is exploited (location attack) and finally the noise amplification stage when the adversary combines data and location leakage (hybrid attack).

Notation & MI Metric. In this subsection, random variables are denoted with capital letters. Instances of random variables and constant values are denoted with lowercase letters. Capital bold letters are used for random variable vectors and matrices and calligraphic font denotes sets. All simulations in this section are carried out with the identity leakage function. Observable data-based leakages of a certain intermediate value v are denoted using subscript L_v. Likewise, observable location-based leakages caused by accessing register R_i (where i the index) are denoted using subscript $L_{R\text{-}i}$. To distinguish between data-based leakage and location-based leakage we use superscript L^{data} and L^{loc}. In addition, we assume that different sources of leakage (data, location) have different noise levels i.e. we assume homoscedastic data noise $N^{data} \sim \mathcal{N}(0, \sigma^2_{data})$ and homoscedastic location noise $N^{loc} \sim \mathcal{N}(0, \sigma^2_{loc})$. We use the following formula to compute the MI metric.

$$MI(S; \mathbf{L}) = H[S] + \sum_{s \in \mathcal{S}} Pr[s] \cdot \sum_{\mathbf{m} \in \mathcal{M}^d} Pr[\mathbf{m}] \cdot \int_{\mathbf{l} \in \mathcal{L}^{(d+1)}} Pr[\mathbf{l}|s, \mathbf{m}] \cdot log_2 Pr[s|\mathbf{l}] \;\; d\mathbf{l}$$

$$\text{where } Pr[s|\mathbf{l}] = \frac{\sum_{\mathbf{m}^* \in \mathcal{R}} Pr[\mathbf{l}|s, \mathbf{m}^*]}{\sum_{s^* \in \mathcal{S}} \sum_{\mathbf{m}^* \in \mathcal{R}} Pr[\mathbf{l}|s^*, \mathbf{m}^*]}$$

Random variable S denotes the secret exponent bit, $\mathbf{L}$ denotes the leakage vector and $\mathbf{M}$ is a d-dimensional randomness vector that we need to sum over when randomization is in place, i.e. d is the attack order.

Data leakage attack. The first obvious way to recover k_{n-1} is by observing the data leakage of the values b_{n-1} and a_{n-1} at the same time. We run the algorithm for the first two rounds and note the intermediate values that can leak information. We let b' be a random value from R$\{0, 1\}$, then:

$i = n - 1$	$i = n - 2$
1. $b_m = b_{n-1} \oplus b'$	6. $b_m = b_{n-2} \oplus b'$
2. $a_m = b_m \oplus a_{n-1}$	7. $a_m = b_m \oplus a_{n-2}$
3. $R_0 = R_{b_m} \cdot R_{a_m}$	8. $R_0 = R_{b_m} \cdot R_{a_m}$
4. $R_1 = R_0 \cdot U_{b_{n-1}}$	9. $R_1 = R_0 \cdot U_{b_{n-2}}$
5. $b' = b_{n-1}$	10. $b' = b_{n-2}$

As can be observed in above, the value b_{n-1} is accessed in the first iteration ($i = n - 1$) three times, once when b_m is calculated (line 1), once implicitly for the index of $U_{b_{n-1}}$ (line 4) and finally for b' (line 5). The

value a_{n-1} is accessed once during the first iteration ($i = n-1$) and it is not used in the second iteration ($i = n-2$). We notice that the value b_{n-1} is used implicitly again in the second iteration, since it is equal to b'. An attacker observing the power leakage of this algorithm should be able to probe at two different points in time, in order to observe both leakages $L^{data}_{a_{n-1}}, L^{data}_{b_{n-1}}$ and eventually the key, i.e. we conclude that a second-order attack is possible for this scheme. Note also that the an adversary with ability to conduct horizontal side-channel attacks [BCPZ16] could observe the leakage of b_{n-1} multiple times, average them by computing $\bar{L}^{data}_{b_{n-1}} = \frac{1}{4} * \sum_{j=1}^{4} L^{data}_{b_{n-1}}$ in order to reduce the noise level and finally perform a second-order attack. The results of the MI evaluation are visible in Figure 1. As expected, the exponent splitting scheme performs noise amplification and has a different slope compared to an unprotected exponentiation (Algorithm 3.1). In addition, we observe the curve's horizontal shift to the right caused by the horizontal exploitation of the available leakage, i.e. we can quantify the effect of multiple leaky points for b_{n-1}.

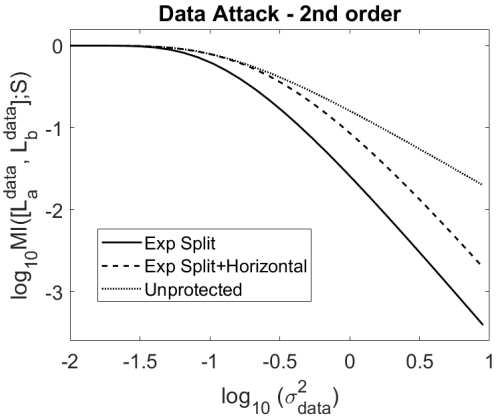

Fig. 1. MI evaluation for Algorithm 2, using a data leakage attack, with and without horizontal exploitation. Observed leakage vector $\mathbf{L} = [L^{data}_{a_{n-1}}, L^{data}_{b_{n-1}}]$.

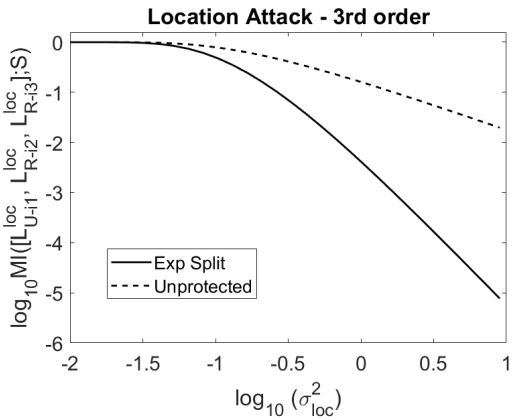

Fig. 2. MI evaluation for Algorithm 2, using a location leakage attack. Observed leakage vector $\mathbf{L} = [L^{loc}_{U\text{-}i_1}, L^{loc}_{R\text{-}i_2}, L^{loc}_{R\text{-}i_3}]$.

Location leakage attack. Let us assume that the adversary can distinguish between the manipulation of registers according to which address is accessed, similar to the address-bit DPA attack described in [IISO10]. If the adversary can distinguish between accesses to, U_0 and U_1 for example, a direct consequence is recovery of value b_{n-1}. To mount a successful attack against Algorithm 5 using solely location-based leakage, we need the simultaneous observation of the address of U_{i_1} and R_{i_2} and R_{i_3}, for indexes $i_1 = b_{n-1}$ (line 4) and $i_2 = b_m$ (line 3) and $i_3 = a_m$ (line 3). Thus, in order to recover k_{n-1}, we need to observe leakage vector $L^{loc} = [L^{loc}_{U\text{-}i_1}, L^{loc}_{R\text{-}i_2}, L^{loc}_{R\text{-}i_3}]$, i.e. perform a third-order attack. The results are visible in Figure 2, where we can observe the noise amplification effect that increases the curve's slope. Naturally, a third-order attack using only location-based leakage tends to be less effective compared to a second-order attack using only data-based leakage. However, depending on the device, exploiting the address dependency may be more effective than exploiting the data dependency. That is, the third-order attack can become more efficient if $\sigma_{data} > \sigma_{loc}$.

Hybrid leakage attack. Lastly, we analyze the scenario in which an adversary can observe both data-based and location-based leakage. Using this information the adversary can use leakage vector $\mathbf{L} = [L^{data}_{a_{n-1}}, L^{loc}_{U\text{-}b_{n-1}}]$ to carry out a second-order attack that uses data leakage to recover bit a_{n-1} and location leakage with regard to register U to recover bit b_{n-1}. Since data and location leakage imply different noise levels, i.e. ($\sigma_{data} \neq \sigma_{loc}$), we need to represent the available information as a three-dimensional plot, as in Figure 3. The wave-like plot quantifies the attainable information with regard to a particular data and location noise level. Thus, it assists the side-channel evaluator to analyze the scheme's security in a more holistic way that factors in location leakage and demonstrates the tradeoff between data noise and location noise. If for

instance $\sigma_{loc} \ll \sigma_{data}$ in the target device, the adversary can directly opt for the hybrid attack, instead of pursing a data-only attack route.

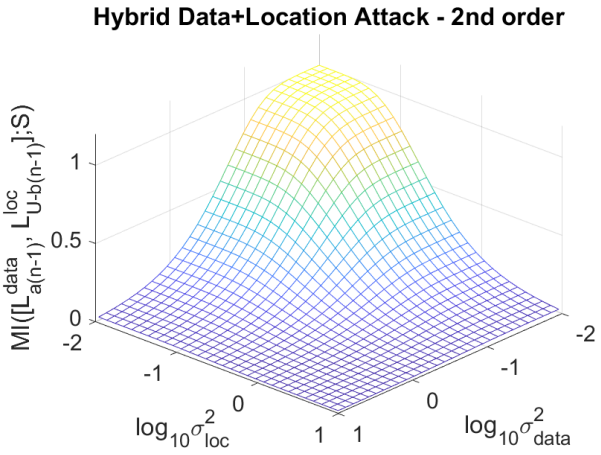

Fig. 3. MI evaluation for Algorithm 5 exponent splitting, using a hybrid leakage attack. Observed leakage vector $\mathbf{L} = [L^{data}_{a_{n-1}}, L^{loc}_{U\text{-}b_{n-1}}]$.

6 Implementation Considerations

In this section, we describe the results of applying Test Vector Leakage Assessment (TVLA) [GJJR11] to implementations of some of the algorithms above. We further describe modifications required to achieve a secure implementation where the hardware architecture can mean that variables that should be independent leak at the same time, potentially unmasking a secret value [BGG+14].

Our implementations were developed using Xilinx's Zynq zc702 evaluation board. The Zynq zc702 microprocessor contains two ARM7 cores and an FPGA fabric. We used one ARM7 core for our implementations, clocked at 667 MHz, and the FPGA provided a means of triggering an oscilloscope at a convenient point in our implementations. We acquired a trace of the electromagnetic emanations around one of the coupling capacitors.

The test that we used from TVLA is to determine whether there are statistically significant differences in the mean traces of two sets of traces, one acquired with a fixed scalar and the other with random scalar. One would typically randomly interleave acquisitions so that environmental effects are the same for both sets and there are no erroneous indications of leakage, caused, for example, by the least significant bit of a variable used to count the number of acquisitions. In applying this, one would take two sets of data, and conduct Welch's t-test point-by-point to determine whether there is evidence against the null hypothesis that the sets are the same. We determine that leakage is present if we observe values above $6.63\,\sigma$ which gives the probability of indicating leakage where no leakage is present, often referred to as a Type I error, of approximately 1×10^5 when using traces containing 3×10^5 samples. The interested reader is referred to Goodwill et al. [GJJR11] and Schneider and Moradi [SM15] for a thorough description.

We made a straightforward implementation of Algorithm 6 using NIST's P192 curve and conducted a test where we compared a set of traces with a fixed scalar compared to a set of traces with a random scalar. The elliptic curve points were implemented as homogeneous projective points. We use the x and z-coordinates in conjunction with so-called x-only algorithms for point arithmetic [BJ02], as one would for an implementation of ECDH. The instantaneous electromagnetic emanations around the targeted capacitor were measured during the execution of the first 20 rounds of the implementation. The top-left trace in

Figure 4 shows the result of a TVLA analysis with 1×10^3 traces where leakage can be seen in numerous places.

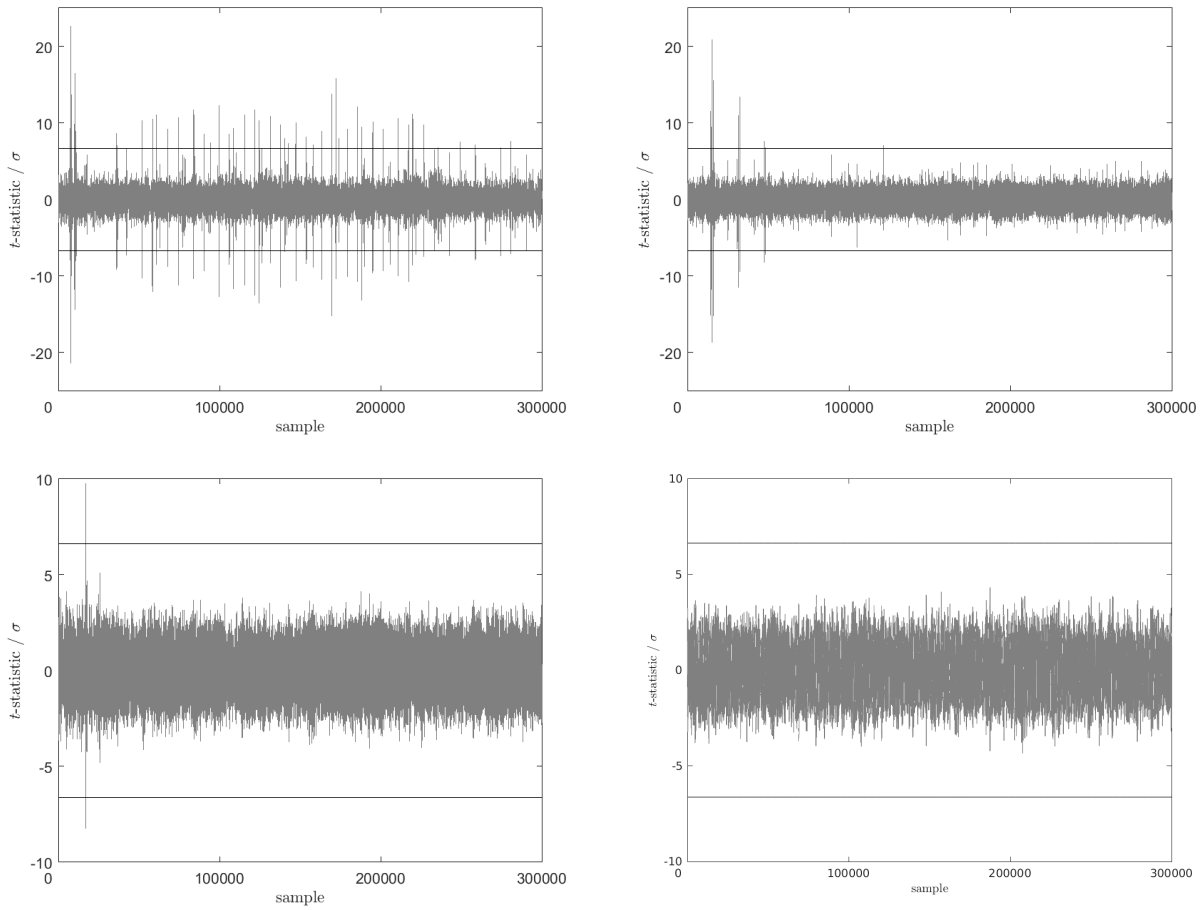

Fig. 4. From top left to bottom right we show: an unmasked implementation showing leakage after 1×10^3 traces, a naïve implementation of Algorithm 6 and a more secure variant both showing leakage after 1×10^6 traces, and an implementation of Algorithm 11 that does not show leakage after 1×10^6 traces

A straightforward implementation of Algorithm 6 was tested in the same way. The algorithm is similar to that proposed by Izumi et al. [IIT03] but with masking conducted before the execution of the scalar multiplication, rather than on-the-fly. The resulting TVLA traces is shown in the top-right of Figure 4, where we note that significant leakage is present with 1×10^6 traces. This is caused by the microprocessor combining values held in registers because of the architecture chosen by the designers [BGG+14].

A more secure implementation can be made by computing some of the required indices before the execution of the main loop of the scalar multiplication, as shown in Algorithm 11. We set C to $B \oplus \left\lfloor \frac{B}{2} \right\rfloor$ such that individual bits of B are masked by adjacent bits. The resulting TVLA trace is shown in the bottom-left of Figure 4, where we observe that there is only one place where we see significant leakage with 1×10^6 traces. This leakage occurs because the initial state of $\{\boldsymbol{R}_0, \boldsymbol{R}_1\}$ contain $\{\boldsymbol{P}, 2\,\boldsymbol{P}\}$ in some random order. In the first loop of the scalar multiplication $\{\boldsymbol{R}_0, \boldsymbol{R}_1\}$ is overwritten with $\{2\,\boldsymbol{P}, 3\,\boldsymbol{P}\}$ or $\{3\,\boldsymbol{P}, 4\,\boldsymbol{P}\}$, in some random order, depending on whether the second most-significant bit of κ is set to 0 or 1, respectively. When

$2\boldsymbol{P}$ overwrites $2\boldsymbol{P}$ the side-channel leakage will be significantly different to any other possible combination, since the Hamming distance will be zero.

Algorithm 11: Montgomery Ladder with XOR-Split Scalar on an EC

Input: $\mathcal{E}$, $\mathbb{F}_q$, $\boldsymbol{P} \in \mathcal{E}$, n-bit integers
$A = \sum_{i=0}^{n-1} a_i\, 2^i, \quad B = \sum_{i=0}^{n-1} b_i\, 2^i$
Output: $\boldsymbol{Q} = [\kappa]\boldsymbol{P}$ where $\kappa = A \oplus B$
Uses: $C = \sum_{i=0}^{n-1} c_i\, 2^i$

1 $\boldsymbol{R}_0 \leftarrow \boldsymbol{P}$; $\boldsymbol{R}_1 \leftarrow \boldsymbol{P}$; $\boldsymbol{R}_2 \leftarrow \boldsymbol{P}$;
2 $C \leftarrow B \oplus \left\lfloor \frac{B}{2} \right\rfloor$;
3 $b' \leftarrow b_{n-1}$;
4 $\boldsymbol{R}_{\neg b'} \leftarrow 2\boldsymbol{P}$;
5 **for** $i = n-2$ **down to** 0 **do**
6 $\quad \boldsymbol{R}_2 \leftarrow \boldsymbol{R}_{a_i} + \boldsymbol{R}_{\neg a_i}$;
7 $\quad \boldsymbol{R}_{a_i} \leftarrow 2\,\boldsymbol{R}_{a_i \oplus c_i}$;
8 $\quad \boldsymbol{R}_{\neg a_i} \leftarrow \boldsymbol{R}_2$;
9 **end**
10 **return** $\boldsymbol{R}_{b_0}$

Algorithm 12: Montgomery Ladder with XOR-Split Scalar II on an EC

Input: $\mathcal{E}$, $\mathbb{F}_q$, $\boldsymbol{P} \in \mathcal{E}$, n-bit integers
$A = \sum_{i=0}^{n-1} a_i\, 2^i, \quad B = \sum_{i=0}^{n-1} b_i\, 2^i$
Output: $\boldsymbol{Q} = [\kappa]\boldsymbol{P}$ where $\kappa = A \oplus B$
Uses: $C = \sum_{i=0}^{n-1} c_i\, 2^i$ and $D = \sum_{i=0}^{n-1} d_i\, 2^i$

1 $\boldsymbol{R}_0 \leftarrow \boldsymbol{P}$; $\boldsymbol{R}_1 \leftarrow \boldsymbol{P}$;
2 $\boldsymbol{U}_0 \leftarrow \boldsymbol{P}$; $\boldsymbol{U}_1 \leftarrow -\boldsymbol{P}$;
3 $C \leftarrow B \oplus \left\lfloor \frac{B}{2} \right\rfloor$; $D \leftarrow C \oplus A$;
4 $b' \leftarrow b_{n-1}$;
5 $\boldsymbol{R}_{\neg b'} \leftarrow 2\boldsymbol{P}$;
6 **for** $i = n-2$ **down to** 0 **do**
7 $\quad \boldsymbol{R}_0 \leftarrow \boldsymbol{R}_{c_i} + \boldsymbol{R}_{d_i}$;
8 $\quad \boldsymbol{R}_1 \leftarrow \boldsymbol{R}_0 + \boldsymbol{U}_{b_i}$;
9 $\quad b' \leftarrow b_i$;
10 **end**
11 **return** $\boldsymbol{R}_{b_0}$

A fully secure implementation can be achieved by randomizing the point produced by the doubling operation, by multiplying the x and z-coordinate of the resulting point by a random value. In implementing Algorithm 11, this was achieved by randomizing $\boldsymbol{R}_0$ and $\boldsymbol{R}_1$ before the main loop of the scalar multiplication. The resulting TVLA trace is shown in the bottom-right of Figure 4, where we observe that there is no significant leakage with 1×10^6 traces. An alternative would be to set the coordinates of $\boldsymbol{R}_{a_i}$ to zero before setting $\boldsymbol{R}_{a_i}$ to $\boldsymbol{R}_2$.

Algorithm 12 shows the same arguments applied to Algorithm 7. However there is no need to randomize any points during the loops of scalar multiplication. If the redundant representation of the point assigned to $\boldsymbol{R}_0$ and $\boldsymbol{R}_1$ is randomized separately to that applied to $\boldsymbol{U}_0$ an overwrite with a Hamming distance of zero cannot occur.

7 Conclusion

In this paper, we show how an exponent can be split into two shares, where the exponent is the XOR sum of the two shares and the cost is typically an extra register and some register copies per bit. A significant advantage over previously proposed exponent splitting methods, which have a prohibitive impact on performance [CJ01]. Our method can also be applied to groups whose order contains long runs of bits set to 0 or 1 without any penalty on performance or security. We showed that our algorithms are secure using formal methods, MI-based evaluation and TVLA on an implementation of Boolean exponent splitting. Furthermore, we showed how an ECDSA signature can be generated by only manipulating shares of the random nonce used to secure the scheme. We note that our method does not prevent an attacker from using the intermediate states generated by the algorithms as a means of attack. However, inexpensive solutions such as randomizing projective points [WMPW98] or Ebeid and Lambert's blinding method for RSA [EL10] can be combined with out method to provide a high level of side-channel resistance.

The algorithms presented above will be more efficient than adding a multiple of the group order to the exponent, since the bit length of the exponent is not increased. Moreover, the resistance to collision attacks is superior, since one would need to conduct several attacks to derive each share and reconstruct the exponent.

Where one is adding a random multiple of the exponent any bits recovered directly relate to bits of the exponent used. It has not yet been shown that one can derive an exponent from gaining partial information on a series of blinded exponents, but significant advances have been made [SI11,Sch14,SW14].

Acknowledgments

The authors would like to thank Lauren De Meyer and Michael Hamburg for their helpful comments.

References

[BCLN16] Joppe W. Bos, Craig Costello, Patrick Longa, and Michael Naehrig. Selecting elliptic curves for cryptography: an efficiency and security analysis. *Journal of Cryptographic Engineering*, 6(4):259–286, 2016.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In M. Joye and J.-J. Quisquater, editors, *CHES 2004*, volume 3156 of *LNCS*, pages 16–29. Springer, 2004.

[BCP+14] Lejla Batina, Lukasz Chmielewski, Louiza Papachristodoulou, Peter Schwabe, and Michael Tunstall. Online template attacks. In W. Meier and D. Mukhopadhyay, editors, *INDOCRYPT 2014*, volume 8885 of *LNCS*, pages 21–36. Springer, 2014.

[BCPZ16] Alberto Battistello, Jean-Sébastien Coron, Emmanuel Prouff, and Rina Zeitoun. Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, pages 23–39, 2016.

[Ber06] Daniel J. Bernstein. Curve25519: New Diffie-Hellman speed records. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 207–228. Springer, 2006.

[BGG+14] Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and François-Xavier Standaert. On the cost of lazy engineering for masked software implementations. In Marc Joye and Amir Moradi, editors, *CARDIS 2014*, volume 8968 of *LNCS*, pages 64–81. Springer, 2014.

[BJ02] Eric Brier and Marc Joye. Weierstraß elliptic curves and side-channel attacks. In David Naccache and Pascal Paillier, editors, *PKC 2002*, volume 2274 of *LNCS*, pages 335–345. Springer, 2002.

[BL09] Daniel J. Bernstein and Tanja Lange. A complete set of addition laws for incomplete edwards curves. *IACR Cryptology ePrint Archive*, 2009:580, 2009.

[CJ01] Christophe Clavier and Marc Joye. Universal exponentiation algorithm. In C. K. Koç, D. Naccache, and C. Paar, editors, *CHES 2001*, volume 2162 of *LNCS*, pages 300–308. Springer, 2001.

[CJ03] Matthieu Ciet and Marc Joye. (virtually) free randomization techniques for elliptic curve cryptography. In S. Qing, D. Gollmann, and J. Zhou, editors, *ICICS 2003*, volume 2836 of *LNCS*, pages 348–359. Springer, 2003.

[Cor99] Jean-Sébastien Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In C. K. Koç and C. Paar, editors, *CHES 1999*, volume 1717 of *LNCS*, pages 292–302. Springer, 1999.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In *CHES 2002*, pages 13–28, 2002.

[Edw07] Harold M. Edwards. A normal form for elliptic curves. In Çetin .K. Koç and Christof Paar, editors, *Bulletin of the American Mathematical Society*, volume 44, pages 393–422, 2007. `http://www.ams.org/journals/bull/2007-44-03/S0273-0979-07-01153-6/home.html`.

[EL10] Nevine Maurice Ebeid and Rob Lambert. A new CRT-RSA algorithm resistant to powerful fault attacks. In *WESS 2010*, page 8. ACM, 2010.

[FRV14] Benoit Feix, Mylène Roussellet, and Alexandre Venelli. Side-channel analysis on blinded regular scalar multiplications. In W. Meier and D. Mukhopadhyay:, editors, *INDOCRYPT 2014*, volume 8885 of *LNCS*, pages 3–20. Springer, 2014.

[GJJR11] Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatgi. P.: A testing methodology for side channel resistance validation, 2011. NIST non-invasive attack testing workshop.

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In C. K. Koç, D. Naccache, and C. Paar, editors, *CHES 2001*, volume 2162 of *LNCS*, pages 251–261. Springer, 2001.

[Gou01] Louis Goubin. A sound method for switching between Boolean and arithmetic masking. In C. K. Koç, D. Naccache, and C. Paar, editors, *CHES 2001*, volume 2162 of *LNCS*, pages 3–15. Springer, 2001.

[Gou03] Louis Goubin. A refined power-analysis attack on elliptic curve cryptosystems. In Yvo Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 199–210. Springer, 2003.

[Ham15] Michael Hamburg. Ed448-Goldilocks, a new elliptic curve. Cryptology ePrint Archive, Report 2015/625, 2015. `http://eprint.iacr.org/2015/625`.

[HKT15] Neil Hanley, HeeSeok Kim, and Michael Tunstall. Exploiting collisions in addition chain-based exponentiation algorithms using a single trace. In K. Nyberg, editor, *CT-RSA 2015*, volume 9048 of *LNCS*, pages 431–448. Springer, 2015.

[HS01] Nick Howgrave-Graham and Nigel P. Smart. Lattice attacks on digital signature schemes. *Des. Codes Cryptography*, 23(3):283–290, 2001.

[IISO10] Masami Izumi, Jun Ikegami, Kazou Sakiyama, and Kazou Ohta. Improved countermeasures against address-bit DPA for ECC scalar multiplication. In G. De Michell, B. M. Al-Hashimi, W. Müller, and E. Macii, editors, *DATE 2010*, pages 981–984. IEEE, 2010.

[IIT02] Kouichi Itoh, Tetsuya Izu, and Masahiko Takenada. Address-bit differential power analysis of cryptographic schemes OK-ECDH and OK-ECDSA. In B. S. Kaliski Jr., Ç. K. Koç, and C. Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 129–143. Springer, 2002.

[IIT03] Kouichi Itoh, Tetsuya Izu, and Masahiko Takenada. A practical countermeasure against address-bit differential power analysis. In C. D. Walter, Ç. K. Koç, and C. Paar, editors, *CHES 2003*, volume 2779 of *LNCS*, pages 382–396. Springer, 2003.

[JY02] Marc Joye and Sung-Ming Yen. The Montgomery powering ladder. In B. S. Kaliski Jr., editor, *CHES 2002*, volume 2523 of *LNCS*, pages 291–302. Springer, 2002.

[KJJ99] Paul Kocher, Josh Jaffe, and Ben Jun. Differential power analysis. In M. J. Wiener, editor, *CRYPTO '99*, volume 1666 of *LNCS*, pages 388–397. Springer, 1999.

[KKYH10] HeeSeok Kim, Tae Hyun Kim, Joong Chui Yoon, and Seokhie Hong. Practical second-order correlation power analysis on the message blinding method and its novel countermeasure for RSA. *ETRI Journal*, 32(1):102–111, 2010.

[Koc96] Paul Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In N. Koblitz, editor, *CRYPTO '96*, volume 1109 of *LNCS*, pages 104–113. Springer, 1996.

[LTT15] Duc-Phong Le, Chik-How Tan, and Michael Tunstall. Randomizing the Montgomery powering ladder. In R. N. Akram and S. Jajodia, editors, *WISTP 2015*, volume 9311 of *LNCS*, pages 155–170. Springer, 2015.

[MD99] Thomas S. Messerges and Ezzy A. Dabbish. Investigations of power analysis attacks on smartcards. In Scott B. Guthery and Peter Honeyman, editors, *Smartcard 1999*. USENIX Association, 1999.

[MDS99] Thomas S. Messerges, Ezzy A. Dabbish, and Robert H. Sloan. Power analysis attacks of modular exponentiation in smartcards. In Çetin Kaya Koç and Christof Paar, editors, *CHES'99*, volume 1717 of *LNCS*, pages 144–157. Springer, 1999.

[ML14] Jörn-Marc Schmidt Torsten Schütze Manfred Lochter, Johannes Merkle. Requirements for standard elliptic curves. Cryptology ePrint Archive, Report 2014/832, 2014. `https://eprint.iacr.org/2014/832`.

[MMS01] David May, Henk L. Muller, and Nigel P. Smart. Random register renaming to foil DPA. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *CHES 2001*, volume 2162 of *LNCS*, pages 28–38. Springer, 2001.

[Mon87] Peter L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48(177):243–264, 1987.

[Nat09] National Institute of Standards and Technology (NIST). Recommended elliptic curves for federal government use. In the appendix of FIPS 186-3, available from `http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf`, June 2009.

[NNTW05] David Naccache, Phong Q. Nguyen, Michael Tunstall, and Claire Whelan. Experimenting with faults, lattices and the DSA. In S. Vaudenay, editor, *PKC 2005*, volume 3386 of *LNCS*, pages 16–28. Springer, 2005.

[NP16] Christophe Nègre and Thomas Plantard. Efficient regular modular exponentiation using multiplicative half-size splitting. *Journal of Cryptographic Engineering*, pages 1–9, 2016.

[OM07] Elisabeth Oswald and Stefan Mangard. Template attacks on masking - resistance is futile. In *CT-RSA*, volume 4377 of *LNCS*, pages 243–256. Springer, 2007.

[QS01] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (EMA): Measures and countermeasures for smart cards. In I. Attali and T. P. Jensen, editors, *E-smart 2001*, volume 2140 of *LNCS*, pages 200–210. Springer, 2001.

[RSA78] Ron Rivest, Adi Shamir, and Leonard M. Adleman. Method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[Sch14] Werner Schindler. Exclusive exponent blinding may not suffice to prevent timing attacks on RSA. Cryptology ePrint Archive, Report 2014/869, 2014. `http://eprint.iacr.org/`.

[SI11] Werner Schindler and Kouichi Itoh. Exponent blinding does not always lift (partial) SPA resistance to higher-level security. In J. Lopez and G. Tsudik, editors, *ACNS 2011*, volume 6715 of *LNCS*, pages 73–90. Springer, 2011.

[SM15] Tobias Schneider and Amir Moradi. Leakage assessment methodology - A clear roadmap for side-channel evaluations. In Tim Güneysu and Helena Handschuh, editors, *CHES 2015*, volume 9293 of *LNCS*, pages 495–513. Springer, 2015.

[SMY09] François-Xavier Standaert, Tal Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 443–461. Springer, 2009.

[SOP08] Nigel Smart, Elisabeth Oswald, and David Page. Randomised representations. *IET Proceedings on Information Security*, 2(2):19–27, 2008.

[SW14] Werner Schindler and Andreas Wiemers. Power attacks in the presence of exponent blinding. *J. Cryptographic Engineering*, 4(4):213–236, 2014.

[WMPW98] Erik De Win, Serge Mister, Bart Preneel, and Michael J. Wiener. On the performance of signature schemes based on elliptic curves. In Joe Buhler, editor, *ANTS 1998*, volume 1423 of *LNCS*, pages 252–266. Springer, 1998.

[WvWM11] Marc F. Witteman, Jasper G. J. van Woudenberg, and Federico Menarini. Defeating RSA multiply-always and message blinding countermeasures. In Aggelos Kiayias, editor, *CT-RSA 2011*, volume 6558 of *LNCS*, pages 77–88. Springer, 2011.