Efficient Side-Channel Attacks on Scalar Blinding on Elliptic Curves with Special Structure

Werner Schindler* Andreas Wiemers†

Abstract

In this paper we introduce two new generic side-channel attacks on scalar blinding of elliptic curves where the order of the base point is close to a power of 2. These attacks are in particular relevant for elliptic curves over special prime fields where the prime is 'almost' a power of 2. As in the papers [9, 10] we assume that some side-channel attack has allowed the determination of the bits of the blinded scalars with some uncertainty, which is quantified by the error rate ϵ_b. Our new attacks are tailored to the special structure of these elliptic curves. They are far more efficient than the attacks for general elliptic curves [9, 10]. As a consequence such special elliptic curves need significantly longer blinding factors than general elliptic curves. Both attacks apply to ECC applications, which use a long-term key for the scalar multiplication.

1 Introduction

Papers [9, 10] address generic power attacks on RSA implementations and on ECC implementations where exponent blinding, relatedly scalar blinding, have been applied as algorithmic countermeasures against side-channel attacks. There it is assumed that an adversary has guessed the blinded exponent bits / blinded scalar bits on the basis of an SPA attack or a single-trace template attack on the particular device. Each bit guess is assumed to be false with probability $\epsilon_b > 0$. It may be the case, alternatively, that the bits can be guessed through the use of other side channel attacks such as by exploiting electromagnetic radiation or via a microarchitectural attack on a PC that is processing cryptographic software.

In the case of ECC it is assumed that the scalar multiplication uses a long-term key. This is the case, for example, with static ECDH and with the decryption process of the elliptic curve integrated encryption scheme (ECIES) [5]. A proposal has been made by H. Krawczyk that defines an authentication process for TLS 1.3 that does not rely on a signature. Within this context, static ECDH may take on additional importance [6]. Another important application are deterministic signatures [8] with static ephemeral keys.

In contrast to papers [9, 10] we do not consider general curves. Instead, we focus on elliptic curves where the order of the base point is close to 2^k for some k. This situation is relevant for elliptic curves over a prime fields when the prime is 'almost' a power of 2. Well-known representatives of this class include the following curves: Curve25519, M-511, ED448-Goldilocks,

* Bundesamt für Sicherheit in der Informationstechnik, Germany; Werner.Schindler@bsi.bund.de

† Bundesamt für Sicherheit in der Informationstechnik, Germany; Andreas.Wiemers@bsi.bund.de

Curve41417 and NIST P-384.

Section 2 introduces the definitions and notations from papers [9, 10] that are relevant for this investigation. In section 3, the elliptic curves with the special structure are introduced. In addition to this, the basic idea of our attacks are outlined. In sections 4 and 5, the details of our new attacks which exploit the order of the special base point property are presented. Both attacks are far more efficient than the attacks on general elliptic curves in [9, 10]. As a consequence these special curves require significantly longer blinding factors than general elliptic curves.

2 Definitions and Notation

For the purposes of this investigation, we agree with the starting assumptions in papers [9, 10] and assume that the target device (for example, a smart card, a microcontroller, an FPGA, a PC etc.) executes scalar multiplications on the elliptic curve where scalar blinding shall thwart power attacks.

The papers [9, 10] consider RSA implementations in addition to elliptic curve implementations. To cover both cases, the term 'blinded exponents' is used to avoid clumsy formulations. In this paper, we consequently speak of blinded scalars. The blinded scalars are of the form

$$v_j := d + r_j y \qquad \text{for } j = 1, 2, \ldots, N. \tag{1}$$

The letter y denotes the order of the base point of the elliptic curve, and $d < y$ equals the (randomly selected) long term key. The blinding factor r_j for the scalar multiplication j is drawn uniformly from the set $\{0, 1, \ldots, 2^R - 1\}$.

The binary representation of v_j is $(v_{j;k+R-1}, \ldots, v_{j;0})_2$ where leading zero digits are allowed. For example, on the basis of an SPA attack or a single-trace template attack on the power traces j (or by any other side-channel attack), the attacker guesses the blinded scalars and obtains the guess $\widetilde{v}_j = (\widetilde{v}_{j;k+R-1}, \ldots, \widetilde{v}_{j;0})_2$.

The attacker may commit two types of guessing errors: Although $v_{j;i} = 0$ he might guess $\widetilde{v}_{j;i} = 1$, or despite of $v_{j;i} = 1$ he might guess $\widetilde{v}_{j;i} = 0$. We assume that both errors occur with identical probability ϵ_b. We further assume that the individual bit guesses are independent, which should be justified if the double-and-always algorithm or the Montgomery ladder are applied (see [10], subsection 3.8). Our attacks certainly tolerate some deviation from both assumptions.

3 Special Curves

In this paper we assume that the order y of the base point is 'slightly' smaller or larger than 2^k for some k. Due to the Hasse-Weil Theorem, this assumption is fulfilled for an elliptic curve over a prime field if the characteristic of the field is 'almost' a power of 2 and if in addition the cofactor is 1 or a small power of 2. Moreover, we may assume that y is odd.

If y is larger than 2^k the binary representation of y is of the form:

$$y = 2^k + y_0 = 2^k + \sum_{j=0}^{t} a_j 2^j \quad \text{with } a_t = 1 \tag{2}$$

and $t < k$. Then clearly:

$$2^k + 2^t < y < 2^k + 2^{t+1} \text{ and } r_j y_0 < 2^{R+t+1}. \tag{3}$$

In the following we assume that the number of zeroes between the two most significant non-zero coefficients in the binary representation of y, or the 'gap' g between these coefficients,

$$g := k - t - 1 \quad \text{is large.} \tag{4}$$

It is well-known that if $k - 1 > t + R$ roughly $k - (R + t + 1)$ bits of the long-term key d remain unblinded, which simplifies the discrete log problem to some extent [4]. In this paper we will show that the situation is even more dramatic. Analogously, for $y < 2^k$ there is a unique $t < k$ with:

$$y = 2^k - y_0 = 2^k - \sum_{j=0}^{t} a_j 2^j \quad \text{with } a_t = 1. \tag{5}$$

Consequently,

$$2^k - 2^t > y > 2^k - 2^{t+1} \text{ and } r_j y_0 < 2^{R+t+1}. \tag{6}$$

Table 1 provides parameter sets (k, t, g) for some well-known curves.

curve	k	t	g
Curve25519	252	124	127
M-511	508	252	255
ED448-Goldilocks	446	223	222
Curve41417	411	204	206
NIST P-384	384	189	194

Table 1: Exemplary curves: For Curve25519 and curve M-511 we have $y > 2^k$ while $y < 2^k$ for the other curves.

3.1 A basic Observation

Assume for the moment that $y > 2^k$ and that $k-t-R-1 = g-R$ is significantly larger than 0 (let's say ≥ 7). Then it is unlikely that for the blinded scalar $v_j = d + r_j y$ a carry occurs at position $k-1$ so that $v_{j;k+i} = r_{j;i}$ for $i = 0, \ldots, R-1$. The side-channel attack then directly provides bit guesses $\widetilde{v}_{j;k+i}$ for the binary representation of the blinding factor r_j, each of which is false with probability ϵ_b. In our attacks, we apply these values as first stage estimates for $r_{j;0}, \ldots, r_{j;R-1}$. There is no equivalent for general curves.

While for $y > 2^k$ a carry at bit position $k-1$ occurs if $d + r_j y_0 > 2^k$ for $y < 2^k$ a carry occurs if $d < r_j y_0$. Since $|y - 2^k|/2^k \approx 0$ in both cases, for simplicity we may assume that $\text{Prob}(d = x) = 2^{-k}$ for all $x \in Z_{2^k}$. Hence, the probability for a carry at bit position $k-1$ is nearly the same in both cases.

4 The Wide Window Attack

In this section, we introduce and discuss the wide window attack. We begin with a definition.

Definition 1 *Let $Z_m := \{0, 1, \ldots, m-1\}$. For an integer x we denote by $x (\mathrm{mod}\, m)$ the unique element x' in Z_m with $x' \equiv x \bmod m$.*

We restrict the description of the "wide window attack" to the case $y = 2^k + y_0$. In this case we have

$$v_j = r_j 2^k + (d + r_j y_0) \qquad \text{for } j = 1, 2, \ldots . \quad (7)$$

We assume in the following that

$$d + r_j y_0 < 2^k \qquad \text{for } j = 1, 2, \ldots . \quad (8)$$

with very high probability. This is certainly the case, for instance, for

$$R \leq g - 7 .$$

Under this assumption, the pair $(\lfloor \widetilde{v}_j / 2^k \rfloor, \widetilde{v}_j (\mathrm{mod}\, 2^k))$ is just the pair $(r_j, d + r_j y_0)$ but with guessing errors. The latter pair allows:

- to solve for d,
- to determine the value r_j if d is known.

The main idea of the 'wide window attack' is to find iteratively $d (\mathrm{mod}\, 2^w)$ where w runs from 0 to k in fixed steps of length w'. In each iteration μ we start with a $d (\mathrm{mod}\, 2^{w-w'})$ and candidates $r_j (\mathrm{mod}\, 2^{w-w'})$. In each iteration,

1. we find $d (\mathrm{mod}\, 2^w)$ by using the candidates $r_j (\mathrm{mod}\, 2^{w-w'})$.
2. we find new candidates $r_j (\mathrm{mod} 2^w)$ with the help of $d (\mathrm{mod}\, 2^w)$ when $w \leq R$.

Note that the case $y = 2^k - y_0$ is very similar. In this case we analogously require $d - r_j y_0 \geq 0$, and the pair $(\lfloor \widetilde{v}_j / 2^k \rfloor, \widetilde{v}_j (\mathrm{mod}\, 2^k))$ is just the pair

$$(r_j, d - r_j y_0)$$

with guessing errors.

4.1 Solve for d if $w \leq R$

In this subsection we formulate the algorithm, that solves for d if $w \leq R$. For the moment we fix integers w', m_0.

Algorithm 1 *The Wide Window Attack; here: ALGO 'd-solve' in iteration μ*

1. *From the previous iteration step we have A_μ pairs of the form $(\widetilde{v}_j, \widetilde{r}_j \pmod{2^{w-w'}})$ where $\widetilde{r}_j \pmod{2^{w-w'}}$ is assumed to equal $r_j \pmod{2^{w-w'}}$. Further, we have determined exactly one candidate $\widetilde{d} \pmod{2^{w-w'}}$ for $d \pmod{2^{w-w'}}$. The algorithm starts with $w = w'$ and $A_0 = N$.*
   a) *Generate all pairs (α_j, b_j) that differ jointly in at most m_0 bits within the w' most significant bits of $\lfloor \widetilde{v}_j/2^k \rfloor \pmod{2^w}$ and $\widetilde{v}_j \pmod{2^w}$.*
   b) *Since we know the value $d + y_0 r_j \pmod{2^{w-w'}}$ (provided that the respective guesses from the previous iteration steps are correct), we can compute for each b_j a candidate for $d + y_0 r_j \pmod{2^w}$. We use α_j and $r_j \pmod{2^{w-w'}}$ to compute a candidate for $r_j \pmod{2^w}$. In the end, each pair results in a candidate for $d \pmod{2^w}$.*
   c) *Collect all candidates for $d \pmod{2^w}$ in a list.*
2. *Select the candidate $\widetilde{d} \pmod{2^w}$, which occurs most frequently.*

The probability that at most m_0 bit guessing errors occur jointly in the w' most significant bits of both components $\lfloor \widetilde{v}_j/2^k \rfloor \pmod{2^w}$ and $\widetilde{v}_j \pmod{2^w}$ equals:

$$p_{m_0} = \sum_{m \le m_0} \binom{2w'}{m} \epsilon_b^m (1-\epsilon_b)^{2w'-m} .$$

We set

$$M_{m_0} = \sum_{m \le m_0} \binom{2w'}{m} .$$

Let G_μ denote the number of correct candidates for $r_j \pmod{2^{w-w'}}$ in iteration μ. We can assume that the number of correct candidates for $d \pmod{2^w}$ generated with this algorithm, is roughly:

$$p_{m_0} G_\mu \ge t . \tag{9}$$

We want to choose t such that the correct value is expected to be among the top-ranked candidates of the algorithm. For a moderately large t, we require (as in paper [10])

$$A_\mu M_{m_0} \le (t!)^{1/t} 2^{w'(t-1)/t} . \tag{10}$$

We give concrete values for both terms in subsection 4.3. Note, that we may choose different parameters m_0 in each iteration. Neglecting the exact workload of sorting, the running time of the algorithm is roughly $O(A_\mu M_{m_0})$.

4.2 Solve for d if $w > R$

We adapt the algorithm from subsection 4.1 to $w > R$. Since r_j is assumed to be known, we only generate values b_j in step 1a) that differ in at most s_0 bits from $(\widetilde{v}_j \pmod{2^w})$ in the w' most significant bits.

4.3 Finding candidates for r_j

In this section we present the algorithm that finds candidates for r_j. For the moment, we fix integers n_0, t_0.

Algorithm 2 *The Wide Window Attack; here: ALGO 'r_j-find' in iteration μ*

1. *From the previous iteration step and by Algorithm 1 we have A_μ pairs $(\widetilde{v}_j, \widetilde{r}_j \pmod{2^{w-w'}})$, where $\widetilde{r}_j \pmod{2^{w-w'}}$ is assumed to equal $r_j \pmod{2^{w-w'}}$. Further, we have exactly one candidate $\widetilde{d} \pmod{2^w}$ for $d \pmod{2^w}$. The algorithm starts with $w = w'$ and $A_0 = N$.*
   a) *Generate all values α_j that differ in at most n_0 bits from $\lfloor \widetilde{v}_j/2^k \rfloor \pmod{2^w}$ in the w' most significant bits.*
   b) *Since we know $r_j \pmod{2^{w-w'}}$ for each α_j we can compute a candidate for $r'_j \pmod{2^w}$.*
   c) *For each $r'_j \pmod{2^w}$ we compute the Hamming weight of the w' most significant bits of:*

$$((\widetilde{d} + y_0 r'_j) \pmod{2^w}) \oplus \widetilde{v}_j \pmod{2^w} \tag{11}$$

2. *Collect the candidates for $r'_j (\mathrm{mod}\, 2^w)$, for which the value computed in (11) is below some threshold t_0.*

We set for fixed w'

$$p'_{n_0} = \sum_{m \le n_0} \binom{w'}{m} \epsilon_b^m (1-\epsilon_b)^{w'-m}$$

and

$$N'_{n_0} = \sum_{m \le n_0} \binom{w'}{m} .$$

We can assume that the number of correct candidates is roughly:

$$G_{\mu+1} \approx p'_{n_0} p'_{t_0} G_\mu \,. \tag{12}$$

On the other hand, just by chance, we can expect to have

$$A_\mu N'_{n_0} N'_{t_0} 2^{-w'}$$

candidates that are collected by this algorithm. We can expect

$$A_{\mu+1} \approx G_{\mu+1} + A_\mu N'_{n_0} N'_{t_0} 2^{-w'} \,. \tag{13}$$

In a realistic attack scenario where N is bounded like $N \le 2^{20}$ the algorithm can only be successful if $p'_{n_0} p'_{t_0}$ is not too small (e.g $p'_{n_0} p'_{t_0} \approx 0.25$). This may be achieved, for example, by choosing the parameters:

$$n_0 \approx t_0 \approx w' \epsilon_b \,. \tag{14}$$

Due to (13) the term $N'_{n_0} N'_{t_0} 2^{-w'}$ determines the expected number of candidates $A_0, A_1, \ldots$ in the course of the attack. These numbers of candidates should not shrink too much, nor should they explode. We cannot control this second condition concurrently with the first as it is a property of the error rate ϵ_b. The running time of the algorithm is $O(A_\mu N'_{n_0})$.

4.4 Experimental Results

Simulation experiments were performed using Curve25519 and $R = 120$. Firstly, we searched for parameter sets that fulfill all the conditions (9), (10), (12), (13). A valid parameter set is

$$\begin{aligned} &\epsilon_b = 0.1, \;\; N \in \{500, 1000\}, \;\; w' = 24, \\ &n_0 = t_0 = s_0 = 3, \;\; m_0 = 2 \,. \end{aligned} \tag{15}$$

For this parameter set we have

$$M_{m_0} = 1177, p_{m_0} \approx 0.12, p'_{n_0} \approx 0.79, N'_{t_0} = 2325.$$

Since $n_0 = t_0 = 3 \ge w'\epsilon_b$, G_μ should not decrease too much. On the other hand, since

$$N'_{n_0} N'_{t_0} 2^{-w'} \approx 0.3$$

we may expect that A_μ does not increase. For $N = 1000$, Table 2 shows a typical experimental result. In this experiment, for each $\mu = 0, \ldots, 4$ we performed Algorithm 1 followed by Algorithm 2. For each $\mu = 5, \ldots, 9$, we only have to perform Algorithm 1. The column 'rank' in Table 2 gives the rank of the correct value $d (\mathrm{mod}\, 2^w)$ within the candidates, which were returned by Algorithm 1. 'rank 1' means that Algorithm 1 has found the correct value $d (\mathrm{mod}\, 2^w)$.

Alg	μ	rank	G_μ	A_μ	$G_{\mu+1}$	$A_{\mu+1}$
1, 2	0	1	1000	1000	622	8640
1, 2	1	1	622	8640	386	7966
1, 2	2	1	386	7966	247	5711
1, 2	3	1	247	5711	152	3773
1, 2	4	1	152	3773	99	2277
1	5	1	99	2277		
1	6	1	99	2277		
1	7	1	99	2277		
1	8	1	99	2277		
1	9	1	99	2277		

Table 2: Wide window attack: Example simulation result for $\epsilon = 0.10$, $N = 1,000$. In Step 0 to Step 4 the lower parts of d and the blinding factors were guessed. In Step 5 to Step 9 the upper part of d was guessed.

Notes 1 *(i) In the steps 0 to 4 the number of correct G_μ decreases roughly by a factor of* 0.6. *This fits very well to the factor of $p'_{n_0}p'_{t_0} \approx 0.6$, which we expect by our choice of parameters.*
(ii) As explained above we expected that $A_{\mu+1} \approx G_{\mu+1} + 0.3 \cdot A_\mu$, especially $A_1 \approx 600 + 300 = 900$. In our simulation we observed that the number of A_μ is much larger than expected! We looked at a few examples and found that the errors typically occur in the most significant bits of r_j. The reason for this effect seems to be the special structure of $r_j y_0 + d$: Since the most significant bits of $r_j y_0 + d$ only depend on the most significant bits of r_j, it is difficult to correct errors in the most significant bits of r_j. However, it is very likely that these errors will be corrected in the next iteration step.

We performed 10 simulations for each $N \in \{250, 500, 1000\}$ where we used the parameter set (15). We counted an attack to be successful if, in each step, the rank of the correct d was 1. The results are given in Table 3. We repeated

curve	R	ϵ_b	N	success rate
Curve25519	120	0.10	250	2/10
Curve25519	120	0.10	500	7/10
Curve25519	120	0.10	1,000	9/10

Table 3: Wide window attack

the simulation for larger ϵ_b. For instance, a valid parameter set is:

$$\epsilon_b = 0.14, \;\; N = 30,000, \;\; w' = 24,$$
$$n_0 = t_0 = s_0 = 3, m_0 = 2\,. \qquad (16)$$

For this parameter set we obtain

$$M_{m_0} = 1177, p_{m_0} \approx 0.028, p'_{n_0} \approx 0.56, N'_{t_0} = 2325.$$

As above, we have $n_0 = t_0 = 3 \approx w'\epsilon_b$ and $N'_{n_0}N'_{t_0}2^{-w'} \approx 0.3$. For $N = 60,000$ Table 4 shows a typical experimental result.

Alg	μ	rank	G_μ	A_μ	$G_{\mu+1}$	$A_{\mu+1}$
1, 2	0	1	60,000	60,000	18,972	243,546
1, 2	1	1	18,972	243,546	6,022	143,486
1, 2	2	1	6,022	143,486	1,877	66,236
1, 2	3	1	1,877	66,236	629	27,942
1, 2	4	2	629	27,949	185	11,413
1	5	1	185	11,413		
1	6	1	185	11,413		
1	7	1	185	11,413		
1	8	1	185	11,413		
1	9	1	185	11,413		

Table 4: Wide window attack: Example simulation result for $\epsilon = 0.14$, $N = 60,000$.

Notes 2 *(i) In the steps 0 to 4 the G_μ decrease roughly by factor* 0.3 *per iteration, which again fits very well to $p'_{n_0}p'_{t_0} \approx 0.3$.*
(ii) As for $\epsilon_b = 0.10$ the value A_μ is much larger than predicted by (13).
(iii) In Step 4 $d \pmod{2^{120}}$ was only ranked 2.

A natural question is: What is the largest ϵ_b, for which the attack might work? To answer this question, we searched for valid parameter sets that fulfill the conditions (9), (10), (12), (13). under certain restrictions. We chose $N \leq 2^{24}$ and limited the overall running time by $\leq 2^{50}$ operations. Under these restrictions for $R = 120$ we did not find a admissible parameter set for $\epsilon_b \geq 0.19$.

5 The Narrow Window Attack

In this section we develop the so-called narrow window attack. Like the wide window attack it guesses the long-term key d and the blinding factors $r_1, \ldots, r_N$ in portions. Similarities and differences between both attacks will become evident in the following. Its name is motivated by the fact that the window size w' is much smaller than for the wide window attack ($w' = 8$ – 10 vs. $w' \approx 30$)

As in section 4 we assume that for $j = 1, 2, \ldots$

we have $d + r_j y_0 < 2^k$ if $y > 2^k$ or $d - r_j y_0 > 0$ if $y < 2^k$ with high probability. This is certainly the case for $R \leq g - 7$, for example. Subsubsection 5.1.4 considers the case $R > g - 7$. The narrow window attack falls into three phases.

Algorithm 3 *Narrow Window Attack (Generic description)*

- **Phase 1** *Guess the R least significant bits of the long-term key d and the blinding factors $r_1, \ldots, r_N$.*
- **Phase 2** *Identify the guesses of the blinding factors, which are correct. Remove the other guesses.*
- **Phase 3** *Guess the remaining bits of d from the guesses $\widetilde{r}_{j_1}, \widetilde{r}_{j_2}, \ldots, \widetilde{r}_{j_u}$, which have survived Phase 2.*

5.1 Phase 1

At the beginning of Phase 1 we set

$$\widetilde{r}_{j;i} := \widetilde{v}_{j;k+i} \quad \text{for } j = 1, \ldots, N,\ i = 0, \ldots, R-1\,. \tag{17}$$

By assumption a carry from bit $k-1$ occurs with non-negligible probability and thus

$$\text{Prob}(r_{j;i} = \widetilde{r}_{j;i}) = 1 - \epsilon_b \text{ for all pairs } (j, i)\,. \tag{18}$$

The goal of Phase 1 is to guess $d (\text{mod}\, 2^R)$ and to correct the false bit guesses $\widetilde{r}_{j;i}$ for a (sufficiently large) subset of the blinding factors, which will allow us to finish the overall attack successfully in Phase 3. Before formulating Algorithm 4 we will examine its theoretical background.

Definition 2 *The term* $\text{HD}(a, b)$ *denotes the Hamming distance between the binary representations of the integers a and b. The term $(b \gg i)$ means that the binary representation of the integer b is shifted i positions to the right.*

We assume that d and the blinding factors r_j are realizations of independent random variables X and Z_j (i.e., values taken on by these random variables), which are uniformly distributed on (for simplicity) Z_{2^k} or on Z_{2^R} respectively. For $i \geq 1$ then $X (\text{mod}\, 2^i)$, $Z_j (\text{mod}\, 2^i)$ and (since y is odd) thus also $yZ (\text{mod}\, 2^i)$ are uniformly distributed on Z_{2^i}. In particular, Z_j and

$$V_j := (X (\text{mod}\, 2^i) + y_0 Z_j (\text{mod}\, 2^i)) (\text{mod}\, 2^i) \tag{19}$$

are independent and uniformly distributed on Z_{2^i}. More precisely,

$$\begin{aligned} &\text{For fixed } i, \text{ the random variables} \\ &X (\text{mod}\, 2^i),\ Z_j (\text{mod}\, 2^i) \text{ and } V_j (\text{mod}\, 2^i) \\ &\text{are uniformly distributed on } Z_{2^i}\,. \\ &\text{Any two of them are independent.} \end{aligned} \tag{20}$$

Assume that during Phase 1 (intermediate) i-bit guesses $\widetilde{d}_{(sf)}$, $\widetilde{r}_{j;(sf)}$, $\widetilde{v}_{j;(sf)} \in Z_{2^i}$ have been derived ('sf' stands for 'so far'). For the moment we assume that $\widetilde{r}_{j;(sf)} = r_j (\text{mod}\, 2^i)$ and $\widetilde{v}_{j(sf)} = v_j (\text{mod}\, 2^i)$, i.e., that both intermediate guesses are correct. Since $d \equiv (\widetilde{v}_{j(sf)} - \widetilde{r}_{j(sf)} y_0) (\text{mod}\, 2^i)$ the intermediate guess $\widetilde{d}_{(sf)}$ is correct, too. On basis of the next initial bit guesses $\widetilde{r}_{j;i+w'-1}, \ldots, \widetilde{r}_{j;i}$ and $\widetilde{v}_{j;i+w'-1}, \ldots, \widetilde{v}_{j;i}$ ((17)), we want to compute the probability for the candidates for next w' bits $d_{i+w'-1}, \ldots, d_i$ of the long-term key. In terms of random variables, we are interested in the conditional probability

$$\begin{aligned} &\text{Prob}((X \gg i) (\text{mod}\, 2^{w'}) = x' \mid \ \times \\ &\times \widetilde{r}_{j;i+w'-1}, \ldots, \widetilde{r}_{j;i}, \widetilde{v}_{j;i+w'-1}, \ldots, \widetilde{v}_{j;i}, \widetilde{r}_{j;(sf)}, \widetilde{v}_{j;(sf)}) \end{aligned} \tag{21}$$

for all $x' \in Z_{2^{w'}}$. Due to (20) the conditional probability (21) equals

$$\sum_{(r', v') \in M(x')} p(r', v' \mid \widetilde{r}_j^*, \widetilde{v}_j^*)\,. \tag{22}$$

Here $M(x')$ denotes all pairs $(r'', v'') \in Z_{2^{w'}} \times Z_{2^{w'}}$, for which the binary representation of $x' \in Z_{2^{w'}}$ equals the bits $i + w' - 1, \ldots, i$ of the term $((v'' 2^i + \widetilde{v}_{j(sf)}) - (r'' 2^i + \widetilde{r}_{j(sf)}) y_0) (\text{mod}\, 2^{w'})$. Further, $\widetilde{r}_j^* := (\widetilde{r}_{j;i+w'-1}, \ldots, \widetilde{r}_{j;i})_2$ and $\widetilde{v}_j^* := (\widetilde{v}_{j;i+w'-1}, \ldots, \widetilde{v}_{j;i})_2$. The term $p(r', v' \mid \widetilde{r}_j^*, \widetilde{v}_j^*)$ denotes the conditional probability that $r' := (r_{i+w'-1}, \ldots, r_i)_2$ and $v' :=$

$(v_{i+w'-1}, \ldots, v_i)_2$ are correct if the binary representation of $\widetilde{r}_j^*$ and $\widetilde{v}_j^*$ are the initial guesses (17). By (18)

$$p(r', v' \mid \widetilde{r}_j^*, \widetilde{v}_j^*) = \epsilon_b^h (1-\epsilon_b)^{2w'-h} \quad (23)$$
$$\text{with } h = \mathrm{HD}(r', \widetilde{r}_j^*) + \mathrm{HD}(v', \widetilde{v}_j^*)\,.$$

The value x^*, for which (21) is maximal, provides the most probable candidate for the bits $d_{i+w'-1}, \ldots, d_i$ (maximum-likelihood estimate). Step 1 of Algorithm 4 shall find the position where this maximum occurs. The array $P[2^{w'}]$ stores the conditional probabilities (21). Step 1 of Algorithm 4 determines a candidate $\widetilde{d}_{(i+w')} := x^* 2^i + \widetilde{d}_{(sf)}$ for $d (\mathrm{mod}\, 2^{i+w'})$.

Step 2 of Algorithm 4 guesses the bits $r_{j;i}$ and $v_{j;i}$ for all $j \le N$. It may turn out that the initial guesses $\widetilde{r}_{j;i}$ and / or $\widetilde{v}_{j;i}$ have to be flipped. For $m \in \{0, 1\}$ we define

$$p_{j;m} := \sum_{\substack{(r',v') \in M(x^*): \\ r' (\mathrm{mod}\ 2) = m}} p(r', v' \mid \widetilde{r}_j^*, \widetilde{v}_j^*) \quad (24)$$

In Algorithm 4 we set $\widetilde{r}_{j;i} = 0$ if $p_{j;0} \ge p_{j;1}$ and $\widetilde{r}_{j;i} = 1$ otherwise. (This may reverse the initial guesses.) From $\widetilde{d}_{(i+w')}$ and the (possibly modified) value $\widetilde{r}_{j;i}$, we obtain $\widetilde{v}_{j;i}$. Then

$$q_{j;\mathrm{c}} := \frac{p_{j;m}}{p_{j;0} + p_{j;1}} \text{ if } p_{j;m} \ge p_{j;1-m} \quad (25)$$

quantifies the probability that $\widetilde{r}_{j;i}$ and $\widetilde{v}_{j;i}$ are (now) correct if $\widetilde{d}_{(i+w')}$ is correct. The double array $Q[N][R]$ contains the conditional probabilities (25). For each i of the outer for-loop the intermediate guesses $\widetilde{d}_{(sf)}$, $\widetilde{r}_{j;(sf)}$ and $\widetilde{v}_{j;(sf)}$ are updated and extended by one bit. We point out that for the last bits, namely if $i > R - w'$, in Step 1 we have ($r' \in Z_{2^{R-i}}$, $v' \in Z_{2^{w'}}$) instead of ($r' \in Z_{2^{w'}}$, $v' \in Z_{2^{w'}}$). The sum (22) is calculated analogously as above.

Algorithm 4 *Narrow Window Attack, Phase 1*
$\widetilde{d}_{(sf)} := 0$; $\widetilde{r}_{j;(sf)} := 0$; $\widetilde{v}_{j;(sf)} := 0$;
for $i = 0$ *to* $R - 1$ *do* {

1. *for* $m = 0$ *to* $2^{w'} - 1$ *do* $P[m] := 0$;
   for $j = 1$ *to* N *do* {
   calculate the cond. probabilities (22) for all $x' \in Z_{2^{w'}}$
   add these values to the array P }
   select x^* *for which* $P[x^*]$ *is maximal.*
   $\widetilde{d}_{(i+w'-1)} := x^* \cdot 2^i + \widetilde{d}_{(sf)}$ /*End of Step 1*/

2. *for* $j = 1$ *to* N *do* {
   compute $p_{j;0}$ *and* $p_{j;1}$
   if $(p_{j;0} \ge p_{j;1})$ *then* $\widetilde{r}_{j;i} := 0$ *else* $\widetilde{r}_{j;i} := 1$
   $Q[j][i] := q_{j;c}$
   $z := (\widetilde{d}_{(i+w')} + (\widetilde{r}_{j;i} 2^i + \widetilde{r}_{j;(sf)}) y_0) \gg i$
   $\widetilde{v}_{j;i} = z (\mathrm{mod}\, 2)$
   $\widetilde{r}_{j;(sf)} := \widetilde{r}_{j;i} 2^i + \widetilde{r}_{j;(sf)}$ /* *new guess* */
   $\widetilde{v}_{j;(sf)} := \widetilde{v}_{j;i} 2^i + \widetilde{v}_{j;(sf)}$ /* *new guess* */
   } /* *End of Step 2* */

$\widetilde{d}_{(sf)} := x^* (\mathrm{mod}\, 2) \cdot 2^i + \widetilde{d}_{(sf)}$/* *new guess* */
} /* *End of the i-loop* */

5.1.1 Rationale

In Step 1 we guess w' bits of the long-term key d, and we use these bit guesses to determine $\widetilde{r}_{j;i}$ and $\widetilde{v}_{j;i}$. At the end of the outer for-loop, we then discard the upper $w' - 1$ bit guesses $\widetilde{d}_{i+w'-1}, \ldots, \widetilde{d}_{i+1}$. This may be surprising at first sight. The reason is the following: a guessing error in $\widetilde{r}_{j;s}$ does not only affect bit position s, but also many positions $s' > s$. Since the 'horizon' of our window ends at position $i + w' - 1$, the guess $\widetilde{r}_{j;i}$ should usually be the most reliable one within $\{\widetilde{r}_{j;i}, \ldots, \widetilde{r}_{j;i+w'-1}\}$. The next windows will give more precise information on the higher bits.

5.1.2 Removing False Guesses

During Phase 1 false bit guesses $\widetilde{r}_{j;i}$ will definitely occur. Although the maximum-likelihood estimator $\widetilde{d}_{(i+w'-1)}$ is robust in the sense that it tolerates a large fraction of false 'so far' guesses $\widetilde{r}_{j';(sf)}$ of the blinding factors, the fraction of correct guesses clearly should not become too small. The term $q_{j;\mathrm{c}}$ quantifies the probability that the decision for $\widetilde{r}_{j;i}$ (and thus also for $\widetilde{v}_{j;i}$) is correct assuming that $\widetilde{d}_{(i+w'-1)}$ itself is correct. Intermediate guesses $\widetilde{r}_{j;(sf)}$, which are likely to be

false, should be removed. Two general strategies exist: continuous withdrawal for each $i \leq R-1$ and withdrawal at distinguished bit positions i.

Applied as a pure strategy, the first option of discarding (de-activating) all the power traces for which $q_{j;\mathrm{C}}$ is below some predefined critical threshold $cb(\epsilon_b, \cdot)$ keeps the fraction of false, intermediate guesses small. This, however, requires a large sample size N since many correct power traces are discarded as well.

On the other hand a bit guessing error $\widetilde{r}_{j;s}$ also affects many later bit positions $s' > s$. Hence a small product of conditional probabilities $\prod_{s=0}^{i} Q[j][s]$ is an indicator for a wrong bit guess in the past. Another (though weaker) criterion is the number of bit flips in $\widetilde{r}_{j;0}, \ldots, \widetilde{r}_{j;i}, \widetilde{v}_{j;0}, \ldots, \widetilde{v}_{j;i}$, again due to the propagation of guessing errors. These criteria may be applied every st bits to remove power traces.

In our experiments we followed a mixed strategy. First of all, for each $i \leq R-1$ we removed all power traces j, for which $q_{j;\mathrm{C}} < cb(\epsilon_b, i)$. This threshold $cb(\epsilon_b, i)$ increased in ϵ_b and i and ranges in the interval $[0.505, 0.53]$. Every $st = 16$ bits we ordered the power traces, which were still active at that time (i.e., which had not already been removed earlier) with regard to their products $\prod_{s=0}^{i} Q[j][s]$ in descending order. To each power trace we assigned its rank $rk_Q(j)$. Then we ordered the same power traces with regard to the number of corrections in $\widetilde{r}_{j;0}, \ldots, \widetilde{r}_{j;i}, \widetilde{v}_{j;0}, \ldots, \widetilde{v}_{j;i}$ compared to the respective initial guesses (17) in ascending order. This yielded $rk_C(j)$. Since the second criterion is weaker than the first, we computed the overall rank of power trace j to:

$$rk(j) := rk_Q(j) + 0.2 rk_C(j)\,. \qquad (26)$$

Finally, we ordered these power traces in ascending order with regard to their overall rank (26). From $N(i)$ active power traces, the $\lfloor \alpha(\epsilon_b, i) N(i) \rfloor$ top-ranked survived, the remaining power traces were discarded. The survival rate $\alpha(\epsilon_b, i)$ increases in both ϵ_b and i. For $\epsilon_b = 0.10$, for instance, we used the values $\alpha(0.10, 15) = 0.94$, $\alpha(0.10, 31) = 0.90$, $\alpha(0.10, 47) = 0.85$, $\alpha(0.10, 63) = \alpha(0.10, 79) = 0.75$, $\alpha(0.10, 95) = \alpha(0.10, 111) = 0.72$. For $\epsilon_b = 0.13$ we used 0.86, 0.80, 0.70, 0.53, 0.53, 0.43, and 0.43 at the corresponding bit positions.

5.1.3 Increasing the Efficieny of Algorithm 4

Step 1 requires the computation of $2^{2w'}$ probabilities (23) per power trace while Step 2 only needs $2^{w'}$ such probabilities. This means that Step 1 determines the workload of Algorithm 4. The guessing procedure for the maximum in Step 1 is very robust and tolerates a large fraction of false intermediate guesses $\widetilde{r}_{j;(sf)}$. To save computation time in Step 1 we never used more than $n(\epsilon_b, i)$ power traces. The threshold $n(\epsilon_b, i)$ increased in ϵ_b and in i since the fraction of false intermediate guesses usually increases in the course of the attack. For instance, we used $n(0.10, i) = 250$ for $i < 64$ and $n(0.10, i) = 450$ for all remaining cases. For $\epsilon_b > 0.10$ we added 50 traces per 0.01 step. If more than $n(\epsilon_b, i)$ were still active for bit i then $n(\epsilon_b, i)$ candidates were drawn randomly from this set. In Step 2 all still active power traces were used.

5.1.4 Extending R beyond $g - 7$

Up to that point we had assumed $k - t - R \geq 8$, which makes a carry bit from position $k-1$ very unlikely. In the following we consider the problem when R is larger than $g - 7 = k - t - 8$. Assume $y > 2^k$ for the moment. Let c_i denote the carry bit (binary representation) in $d + r_j y$, which occurs in bit position i and is added to $d_{i+1} + (r_j y)_{i+1}$. As in Section 5 we assume the long-term key d and the blinding factor r_j are realizations of random variables X and Z_j. If $R + t < k - 1$, i.e. if $k - R - t > 1$ then the (average) probability for a randomly selected long-

term key d that $\widetilde{r}_{j;0}$ is affected by a carry equals

$$\nu_k := \mathrm{Prob}(c_{k-1} = 1) = \mathrm{Prob}(X + y_0 Z_j \geq 2^k) = 2^{-R} \sum_{z \in Z_{2^R}} \mathrm{Prob}(X \geq 2^k - z y_0) =$$

$$2^{-R} \sum_{z \in Z_{2^R}} \frac{z y_0}{2^k} \approx 2^{-R} \int_0^{2^R - 1} \frac{z y_0}{2^k}\, dz =$$

$$\int_0^1 \frac{u y_0 2^R}{2^k}\, du = \frac{y_0}{2^{k-R+1}}\,. \qquad (27)$$

Since $c_{k-1+i} = 1$ iff $c_{k-1} = 1$, $r_k = \ldots = r_{k+i-1} = 1$ we conclude

$$\nu_{k+i} := \mathrm{Prob}(c_{k-1+i} = 1) = 2^{-i} \nu_k \text{ for } i \geq 0\,. \qquad (28)$$

Since the probability for a carry bit decreases by 50% per bit position its impact on $\widetilde{r}_{j;i}$ needs to be considered only for few bits. More precisely, (18) changes to:

$$\mathrm{Prob}(r_{j;i} = \widetilde{r}_{j;i}) = 1 - \epsilon'_{b;i} \text{ with}$$
$$\epsilon'_{b;i} := (1 - \epsilon_{b;i})\nu_{k+i} + \epsilon_{b;i}(1 - \nu_{k+i})\,. \qquad (29)$$

If $\nu_{k+i} \approx 0$ both formulae match. For the lowest bits the computation of the conditional probability $p(r', v' \mid \widetilde{r}^*_j, \widetilde{v}^*_j)$ becomes a little bit more costly since individual probabilities have to be multiplied.

Example 1 *(Numerical example, Curve25519)* $R = 125$, $\epsilon_b = 0.12$*: For* $i = 0, \ldots, 5$ *we obtain* $\nu_{252+i} = 0.082$, 0.041, 0.020, 0.010, 0.005, *and* 0.025. *Hence* $\mathrm{Prob}(r_{j;i} = \widetilde{r}_{j;i}) = 1 - \epsilon_{b;i}$ *with* $1 - \epsilon_{b;i} = 0.818$, 0.849, 0.865, 0.872, 0.876, *and* 0.878 *while* $1 - \epsilon_b = 0.88$.

5.2 Phase 2

At the end of Phase 1 the situation is the following: some power traces $j_1, \ldots, j_M$ have survived. The lowest R bits of the long-term key d, the complete blinding factors $r_{j_1}, \ldots, r_{j_M}$ and the R lowest bits of the blinded scalars $v_{j_1}, \ldots, v_{j_M}$ have been guessed. During Phase 1 heuristic criteria were applied to remove (presumably) false guesses. However, for large error rates ϵ_b, it may happen that although the intermediate guess $\widetilde{d}_{(sf)}$ is still correct, less than 10% of the remaining M blinding factor guesses $\widetilde{r}_{j_1}, \ldots, \widetilde{r}_{j_M}$ are (completely) correct. This phenomenon can drastically be reduced by choosing smaller thresholds cb and smaller survival rates $\alpha(\epsilon_b, i)$. On the negative side this approach requires considerably larger sample size N. Since the guessing procedure for the missing bits of d is robust, one might hope that Phase 3 will be successful even in the presence of many false guesses $\widetilde{r}_{j_i}$. However, a very efficient algorithm exists, which effectively filters out the correct blinding factors. W.l.o.g. we may assume that $j_i = i$ for all $i \leq M$ (relabelling).

At the beginning we set $\widetilde{r}_{j;i} := \widetilde{v}_{j;i+k}$. During Phase 1 some guesses $\widetilde{r}_{j;i}$ were flipped (hopefully thereby corrected). We now use these corrections in the opposite direction and set $\widetilde{v}_{j;i+k} := \widetilde{r}_{j;i}$ for $i = 0, \ldots, R-1$. The key observation of our filtering algorithm below is that $(\widetilde{v}_j - \widetilde{r}_j y) - (\widetilde{v}_m - \widetilde{r}_m y) = e_j - e_m$ if $\widetilde{r}_j = r_j$ and $\widetilde{r}_m = r_m$. Here $e_j := (\widetilde{v}_j - \widetilde{r}_j y) - d$ and $e_m := (\widetilde{v}_m - \widetilde{r}_m y) - d$ denote the error vectors. In signed representation, these error vectors have low Hamming weight. In analogy to the enhanced attack [9, 10] it seems to be reasonable to consider the NAF Hamming weight of this difference as an indicator whether $\widetilde{r}_j$ and $\widetilde{r}_m$ are correct. In fact, this is the key, but this approach requires deeper analysis.

In our argumentation below we restrict ourselves to the case $y > 2^k$. We first note that due to Algorithm 4 and the above mentioned correction procedure both the bits $i < R$ and $i \geq k$ of the difference $\widetilde{v}_j - \widetilde{r}_j y = \widetilde{v}_j - \widetilde{r}_j \cdot 2^k \pm \widetilde{r}_j y_0$ are zero. Since $v_j = d - r_j y$ we conclude:

$$\begin{aligned} \widetilde{v}_j - \widetilde{r}_j y &= d + (((r_j - \widetilde{r}_j) y_0) \gg R) 2^R + \\ &\quad + e'_j \cdot 2^R + e''_j \cdot 2^{t+R} + z_j \cdot 2^R \\ \text{with } e'_j \in Z_{2^t},\ & e''_j \in Z_{2^{k-t-R}},\ z_j \in \{0, -1\} \end{aligned} \qquad (30)$$

Elementary, but careful calculations yield:

$$\begin{aligned}&(\widetilde{v}_j - \widetilde{r}_j y) - (\widetilde{v}_m - \widetilde{r}_m y) = \\ &((\Delta_{j,m} y_0) \gg R) 2^R + (e'_j - e'_m) \cdot 2^R + \\ &\quad + (e''_j - e''_m) \cdot 2^{t+R} + z_{j,m} 2^k \\ &\qquad \text{with } \Delta_{j,m} := (r_j - \widetilde{r}_j) - (r_m - \widetilde{r}_m) \\ &\qquad \text{and } z_{j,m} \in \{1, 0, -1, -2\}\,. \end{aligned} \tag{31}$$

The cases $\Delta_{j,m} = 0$ and $\Delta_{j,m} \neq 0$ differ in the bits $R, \ldots, t+R-1$ of (31). We define

$$D_{j,m} := (((\widetilde{v}_j - \widetilde{r}_j y) - (\widetilde{v}_m - \widetilde{r}_m y)) \gg R) (\bmod\ 2^t) \tag{32}$$

If $\Delta_{j,m} = 0$ the term $D_{j,m}$ essentially is the difference of two error vectors of length t, and the expectation and variance of

$$\text{ham}(\text{NAF}(D_{j,m})) \tag{33}$$

follow from Table 5 by multiplying the respective values by the factor $t/1000$.

ϵ_b	0.10	0.12	0.13
$E(\text{ham}(\text{NAF}(\cdot)))$	160.82	184.43	195.40
$\text{Var}(\text{ham}(\text{NAF}(\cdot)))$	10.10	10.46	10.44
ϵ_b	0.14	0.15	0.20
$E(\text{ham}(\text{NAF}(\cdot)))$	205.52	215.08	225.19
$\text{Var}(\text{ham}(\text{NAF}(\cdot)))$	10.46	10.40	10.31

Table 5: Empirical values for the expectation and variance of $\text{ham}(\text{NAF}(\widetilde{x}_{j;1} - \widetilde{x}_{j;2}))$. The numbers $\widetilde{x}_{j;1}$ and $\widetilde{x}_{j;2}$ are noisy guesses (error rate ϵ_b) of a randomly selected 1000-bit integer x_j. Our figures were derived from 100,000 simulation experiments.

For $\Delta_{j,m} \neq 0$ the term $\text{ham}(\text{NAF}(D_{j,m}))$ behaves statistically similar as $\text{ham}(\text{NAF}(T))$ for a random variable T, which is uniformly distributed on Z_{2^t}. By [9], Lemma 1(iii)

$$E(\text{ham}(\text{NAF}(T))) \approx 0.333t \quad \text{and} \tag{34}$$

$$\text{Var}(\text{ham}(\text{NAF}(T))) \approx 0.075t\,. \tag{35}$$

We define the threshold th_{NAF} as the average of the expectations of the term (33) for the two cases $\Delta_{j,m} = 0$ and $\Delta_{j,m} \neq 0$.

Algorithm 5 *Narrow Window Attack, Phase 2*

for $j = 1$ *to* M *do*
  for $m = 1$ *to* M *do* $A[j][m] := 0$;
for $j = 1$ *to* M *do*
  for $m = 1$ *to* $j-1$ *do* {
    if $(\text{ham}(\text{NAF}(D_{j,m})) < th_{NAF}$ *then*
      $A[j][m] := A[m][j] := 1\}$
for $j = 1$ *to* M *do* $S[j]{:=}0$
for $j = 1$ *to* M *do*
  for $m = 1$ *to* M *do* $S[j] := S[j] + A[j][m]$;
$c := max\{S[j] \mid 1 \leq j \leq M\}$
$c2 := \lfloor c/2 \rfloor + 1$;
Remove all traces j *with* $S[j] < c2$

5.2.1 Rationale

If $\widetilde{r}_j$ and $\widetilde{r}_m$ are correct then $\Delta_{j,m} = 0$. If exactly one guess is wrong then $\Delta_{j,m} \neq 0$. If both $\widetilde{r}_j$ and $\widetilde{r}_m$ are wrong then usually $\Delta_{j,m} \neq 0$, but $\Delta_{j,m} = 0$ is also possible. Assume that $s \leq M$ guesses $\widetilde{r}_j$ are correct. For the correct guesses ideally $S[j] = s - 1$ and $S[j] = 0$ otherwise. Since for each pair of wrong guesses $(\text{ham}(\text{NAF}(D_{j,m})) < th_{\text{NAF}}$ is possible as well, and due to statistical noise, we introduced the lower bound $c2$ in Algorithm 5. Simulation experiments showed that Algorithm 5 is very effective. When it failed usually s was very small, typically $s \leq 5$, and $s \ll M$. However, in these cases, usually a bit guessing error in $\widetilde{d}$ had occurred before, so that the attack would not have been successful anyway. NAF representations can be computed very efficiently ([7], Theorem 10.24 with $r = 2$). Hence the execution time of Algorithm 5 does not provide a significant contribution to the overall attack.

5.3 Phase 3

After Phase 2, $u \geq 0$ power traces, or more precisely, their corresponding guesses $\widetilde{r}_j$ and $\widetilde{v}_j (\bmod 2^R)$ remain. If the r-bit guess $\widetilde{d}_{(sf)} = \widetilde{d} (\bmod 2^R)$ from Phase 1 is correct, we may expect that (at least essentially) all guesses $\widetilde{r}_j$ (and thus also the guesses $\widetilde{v}_j (\bmod 2^R)$) are correct. After relabelling these, traces are numbered by

$1, \ldots, u$. The goal of Phase 3 is to guess the bits $k-1, \ldots, R$ of the long-term key d. Since the blinding factors $\widetilde{r}_j$ remain constant in Phase 3, Algorithm 6 is very similar to Algorithm 4 but less time-consuming. Moreover, the number of power traces u is much smaller than in Phase 1, which allows to use a larger window size w''. Algorithm 6 begins with the guesses $\widetilde{d}_{(sf)}$ and $\widetilde{v}_{j;(sf)}$ from the end of Phase 1 while $\widetilde{r}_j := \widetilde{r}_{j;(sf)}$.

Analogously to (21) for $i \geq R$ we are interested in the conditional probability

$$\begin{aligned} &\mathrm{Prob}((X \gg i) (\mathrm{mod}\, 2^{w''}) = x' \mid \quad \times (36) \\ &\times \ \widetilde{r}_j, \widetilde{v}_{j;i+w''-1}, \ldots, \widetilde{v}_{j;i}, \widetilde{v}_{j;(sf)}) \end{aligned}$$

for all $x' \in Z_{2^{w''}}$. Formula

$$\sum_{v' \in M'(x')} p(v' \mid \widetilde{v}_j^*) \qquad (37)$$

is the equivalent to (22) but $M'(x')$ denotes all $v'' \in Z_{2^{w''}}$ for which the binary representation of $x' \in Z_{2^{w''}}$ equals the bits $i+w'-1, \ldots, i$ of the term $((v''2^i + \widetilde{v}_{j(sf)}) - \widetilde{r}_j y_0) (\mathrm{mod}\, 2^{w''})$.

Further, $\widetilde{v}_j^* := (\widetilde{v}_{j;i+w''-1}, \ldots, \widetilde{v}_{j;i})_2$. The term $p(v' \mid \widetilde{v}_j^*)$ denotes the conditional probability that $v' := (v_{i+w''-1}, \ldots, v_i)_2$ if the binary representation of $\widetilde{v}_j^*$ is given by the initial guesses (17). Analogously to (23)

$$\begin{aligned} &p(v' \mid \widetilde{v}_j^*) = (1-\epsilon_b)^{w''-h} \epsilon_b^h \qquad (38) \\ &\text{with } h = \mathrm{HD}(v', \widetilde{v}_j^*)\,. \end{aligned}$$

As in Phase 1 the value x^*, for which (36) is maximal, provides the most probable candidate for the bits $d_{i+w''-1}, \ldots, d_i$ (maximum-likelihood estimate). Step 1 of Algorithm 6 shall find the position of this maximum. The array $P[2^{w''}]$ stores the conditional probabilities (36). Step 1 of Algorithm 6 determines a candidate $\widetilde{d}_{(i+w'')} := x^* 2^i + \widetilde{d}_{(sf)}$ for $d (\mathrm{mod}\, 2^{i+w''})$

Step 2 of Algorithm 6 guesses the bits $v_{j;i}$ for all $j \leq u$. initial guesses $\widetilde{v}_{j;i}$ may be flipped. For $m \in \{0,1\}$ we define

$$p'_{j;m} := \sum_{\substack{v' \in M'(x^*): \\ v' (\mathrm{mod}\ 2) = m}} p(v' \mid \widetilde{v}_j^*)\,. \qquad (39)$$

In Algorithm 6 we set $\widetilde{v}_{j;i} = 0$ if $p'_{j;0} \geq p'_{j;1}$ and $\widetilde{v}_{j;i} = 0$ otherwise. (This may reverse the initial guesses.) We point out that for the last bits, namely when $i > k - w''$, in Step 1 we use $d_k = d_{k+1} = \cdots = 0$. The term (37) is calculated analogously.

Algorithm 6 *Narrow Window Attack, Phase 3*
for $i = R$ to $k-1$ do {

1. *for $m = 0$ to $2^{w''} - 1$ do $P[m] := 0$;*
   for $j = 1$ to m do {
   calculate the cond. probabilities (37) for all $x' \in Z_{2^{w''}}$
   add these values to the array P }
   select x^* for which $P[x^*]$ is maximal.
   *$\widetilde{d}_{(i+w''-1)} := x^* \cdot 2^i + \widetilde{d}_{(sf)}$ /*End of Step 1*/*

2. *for $j = 1$ to u do {*
   compute $p'_{j;0}$ and $p'_{j;1}$
   if ($p'_{j;0} \geq p'_{j;1}$) then $\widetilde{v}_{j;i} := 0$ else $\widetilde{v}_{j;i} := 1$
   $\widetilde{v}_{j;(sf)} := \widetilde{v}_{j;i} 2^i + \widetilde{v}_{j;(sf)}$ / new guess */*
   } / End of Step 2 */*

$\widetilde{d}_{(sf)} := x^* (\mathrm{mod}\, 2) \cdot 2^i + \widetilde{d}_{(sf)}$/ new guess */*
} / End of the i-loop */*

Remark 1 *In place of Algorithm 6 one may alternatively apply Algorithm 1 in [10].*

5.4 Error Correction

In Phase 1 a false bit guess $\widetilde{d}_i$ usually spoils many forthcoming guesses $\widetilde{d}_{i*}$ for $i^* > i$. This is because $\widetilde{d}_i = d_i$ implies many false bit guesses $\widetilde{r}_{i;j}$, and relatedly, many false 'so far' guesses $\widetilde{r}_{j;(sf)}$. In contrast, in Phase 3 a guessing error $\widetilde{d}_i = d_i$ only has local impact. This motivates the following error correction strategy if $\widetilde{d}$ turns out to be wrong: move a sliding window of length w''' (let's say $w''' = 4$) from $\widetilde{d}_R, \ldots, \widetilde{d}_{R+w'''-1}$ to $\widetilde{d}_{k-w'''}, \ldots, \widetilde{d}_{k-1}$ and exhaustively check all alternatives within the current window. This procedure corrects one local error within Phase 3 at negligible costs.

5.5 Experimental Results

We performed simulation experiments for all curves from Table 1. We selected d and the blinding factors $r_1, \ldots, r_N$ uniformly from $\{0, \ldots, 2^k - 1\}$ and from $\{0, \ldots, 2^R - 1\}$, respectively. We counted an attack as successful if it was possible to fully recover d (i.e., if all bits were correct) after the error correction process in Phase 3 (if necessary). As usual, N denotes the sample size. In Phase 1 we used the window size $w' = 8$ and in Phase 3 $w'' = 10$.

Table 6 to Table 9 show simulation results for different curves and blinding lengths. For identical error rates ϵ_b, longer blinding factors require larger sample sizes N. This is due to guessing errors and the thin-out process in Phase 1. Our experiments verify that short blinding factors such as $R = 32$ tolerate even 20% bit errors while $R = 253$, for instance, tolerates 12% bit errors anyway. Table 7 and Table 8 show that for identical sample size N, the success rates for $R = 125$ (Curve25519) and $R = 253$ (curve M-511) are to some degree smaller than for $R = 120$ and $R = 250$, respectively. Apart from the fact that the blinding factors are a little bit longer the main reason for this effect in both cases is that $g - R = 127 - 125 = 255 - 253 = 2$ is very small (see subsection 5.6). Our simulation experiments support the assumption that R and ϵ_b determine the sample size N while the specific properties of the particular curve is of subordinate meaning. In our simulations the variance of the number of traces u, which survived Phase 2 (and thus were available in Phase 3), was rather large. Hence, for precise estimates of the particular success rates, a large number of trials would be necessary.

curve	R	ϵ_b	N	success rate
Curve25519	32	0.10	40	7/10
Curve25519	32	0.10	70	10/10
Curve25519	32	0.12	130	10/10
Curve25519	32	0.15	400	27/30
Curve25519	32	0.17	1000	9/10
Curve25519	32	0.20	6,000	12/25
Curve25519	64	0.10	150	9/10
Curve25519	64	0.12	400	9/10
Curve25519	64	0.14	3,000	9/10
Curve25519	64	0.15	7,500	8/10

Table 6: Narrow window attack (small blinding factors): $g = 127$

curve	R	ϵ_b	N	success rate
Curve25519	120	0.10	700	19/20
Curve25519	120	0.12	5,000	19/20
Curve25519	120	0.13	15,000	23/30
Curve25519	120	0.14	60,000	18/30
Curve25519	120	0.15	400,000	5/10
Curve25519	125	0.10	1000	10/10
Curve25519	125	0.12	6,000	16/20
Curve25519	125	0.13	17,000	8/10
Curve25519	125	0.14	60,000	14/30

Table 7: Narrow window attack: $g = 127$

5.6 Possible Improvements

If $R \leq g - 7$, the bit $v_{j;k}$ is usually not affected by a carry from position $k - 1$. Occasional carries are not harmful and their effect might be interpreted as a slightly increased error rate ϵ_b for bit $r_{j;0}$ and its neighbours. However, if d is extremely close to 2^k, i.e. if, let's say, $g - R = 7$ and $d_{k-1} = \ldots = d_{k-12} = 1$, then nearly every blinding factor r_j results a carry to $r_{j;0}$ and with probabilities $\approx 0.5, 0.25, \ldots$ to its neighbours, see (28). Then $\widetilde{v}_{j;k-1}$ is 0 in about $(1 - \epsilon_b)N$ traces; the same effect as for $d_{k-1} = 0$ where essentially no carries occur. In the first case the attack might already fail at the very first bit. However, this event is very rare and thus does not significantly affect the success probability of the attack. For $R > g - 7$, this event becomes more likely. The considerations from Subsubsection 5.1.4 reduce this phenomenon to some degree. The experimental results emphasize that even for $R = g - 2$ the narrow window attack is

curve	R	ϵ_b	N	success rate
M-511	250	0.07	500	10/10
M-511	250	0.10	30,000	9/10
M-511	253	0.10	40,000	8/10
ED448	220	0.10	30,000	10/10
ED448	220	0.11	120,000	9/10
ED448	220	0.12	700,000	9/10

Table 8: Narrow window attack: $g = 255$ (M-511) or $g = 222$ (ED448)

curve	R	ϵ_b	N	success rate
Curve41417	200	0.07	400	10/10
Curve41417	200	0.10	7,000	8/10
NIST P-384	190	0.10	4,000	10/10
NIST P-384	190	0.12	70,000	9/10

Table 9: Narrow window attack: $g = 206$ (Curve41417) or $g = 194$ (NIST P-384)

successful with high probability.

In subsection 5.4 we introduced an error correction strategy which detects local guessing errors in Phase 3. One might similarly try to identify the position of the first guessing error if it already has occurred in Phase 1. An option would be to restart the attack at a bit position where the maximum likelihood decision in Step 1 of Algorithm 4 was 'close' and to decide for the second likely alternative. Finally, a more sophisticated choice of the parameters $(cb, st, \alpha(\epsilon_b, i))$ might also improve the efficiency of the narrow window attack.

6 Relation Between Both Attacks

The Wide Window Attack considers large windows, which allow to catch the impact of error propagation within a wide horizon, but it does not exploit all information. In contrast, the Narrow Window Attack exploits every piece of information; but within a smaller horizon. Our simulation experiments seem to indicate that the efficiency of both attacks is comparable.

Phase 1 of the Narrow Window Attack dominates the workload. Step 1 of Algorithm 4 evaluates $2^{2w'}$ probabilities $p(r', v' \mid \widetilde{r}_j^*, \widetilde{v}_j^*)$ for each i and each selected index j while Step 2 computes only $2^{w'}$ such probabilities, but for all j, which are still active. The probabilities $p(r', v' \mid \widetilde{r}_j^*, \widetilde{v}_j^*)$ only depend on the Hamming distance of two vectors of length $2w'$ (23) (possibly apart from the very first bits; see subsubsection 5.1.4), which allows to store these probabilities in an array of length $2w' + 1$. Moreover, within the loop it suffices to count the occurrences of the particular Hamming weights for each candidate $x' \in Z_{2^{w'}}$.

Step 1 of Algorithm 4 has to be carried out for at most $n(\epsilon_b, i)$ traces; regardless of the sample size N. Thus the Narrow Window Attack scales rather well. In contrast, for Step 2 all active traces are relevant. Averaged over all i, this number should be smaller than $N/2$. Altogether, Algorithm 4 essentially costs $O(Rn2^{2w'}) + O(RN2^{w'-1})$ operations (consisting of several inexpensive basic operations, at least when the probabilities $p(r', v' \mid \widetilde{r}_j^*, \widetilde{v}_j^*)$ only depend on the Hamming distance; the values $r'y_0 (\mathrm{mod}\, 2^{w'})$ may be stored). Unless the sample size N is extremely large, the first term should dominate. For small (R, ϵ_b), the average of $n(\epsilon_b, i)$ may roughly be estimated by $N/2$.

The parameter sets $(R, \epsilon_b, w', N) \in \{(64, 0.10, 8, 150), (120, 0.12, 8, 5000), (120, 0.14, 8, 60, 000), (220, 0.12, 8, 700, 000)\}$, for example, require very roughly 2^{28}, 2^{32}, 2^{32} or 2^{34} operations, respectively. This is far better than the attacks on general elliptic curves [9, 10].

7 Attacks on General Elliptic Curves

In papers [9, 10] attacks on general elliptic curves were considered. The basic attack is only prac-

tical for short blinding factors.

For ECC the enhanced attack requires about 2^{R+5} NAF calculations [10], Subsect. 3.5. At cost of 2^{69} NAF calculations $R = 64$ tolerates ≈ 0.11 errors [10], Subsect. 3.7. Depending on the error rate ϵ_b the alternate attack ([10], Sect. 4) may allow us even to attack blinding lengths $R > 64$ but at the cost of a large workload.

8 Countermeasures

Our results show that the blinding length must exceed the gap $g = k - t - 1$, which is $\approx k/2$ for curves over special primes. This feature at least reduces the efficiency gain of such special curves in comparison to general curves; the degree to which depends on the concrete platform. Efficiency considerations are beyond the scope of this paper.

Within the context of signatures for Curve25519 D. Bernstein proposes to use 512 bit nonces, i.e. blinding factors of length $R > 256$ [1]. To our knowledge such long blinding factors cannot successfully be attacked.

Remark 2 *Side channel attacks and fault attacks on ECDSA usually aim at the certain knowledge of a few bits of several ephemeral keys, which would allow to find the long-term key by a lattice-based attack. Scalar blinding shall devaluate such knowledge since the attacker has to reduce the blinded ephemeral keys $v_j^* = v_j + ry$ modulo y to obtain short lattice vectors. If $y = 2^k \pm y_0$ and $R < g$ the most significant bits of v_j coincide with the respective bits of v_j^* [3], sect. VIII 6.2, 113, which nullifies the impact of scalar blinding at these bit positions. Hence, for ECDSA the parameter R should exceed g as well.*

9 Conclusion

We have introduced two generic attacks on scalar blinding. When a prime p is 'almost' 2^{k+b} elliptic curves over GF(p) with cofactor 2^b ($b \geq 0$) require much larger blinding factors than general curves. This property at least reduces their efficiency gain in comparison to general elliptic curves.

References

[1] D. Bernstein: Re: Mishandling twist attacks. December 1, 2014; http://www.ietf.org/mail-archive/web/cfrg/current/msg05636.html

[2] Daniel J. Bernstein and Tanja Lange: SafeCurves: Choosing Safe Curves for Elliptic-Curve Cryptography. http://safecurves.cr.yp.to, accessed at April 21, 2015.

[3] M. Ciet: Aspects of Fast and Secure Arithmetics for Elliptic Curve Cryptography. PhD thesis, Catholic University of Louvain, Belgium, 2003.

[4] S.D. Galbraith, J.M. Pollard, R. Ruprai: Computing Discrete Logarithms in an Interval. Math. Comput. 82, 2013, 1191–1195.

[5] V.G. Martínez, F.H. Álvarez, L.K. Encinas, C.S. Ávila: A Comparison of the Standardized Version of ECIES. Sixth International Conference on Information Assurance and Security, 2010.

[6] H. Krawczyk: [TLS] OPTLS: Signature-less TLS 1.3. November 1, 2014; https://www.ietf.org/mail-archive/web/tls/current/msg14385.html

[7] J.H. van Lint: Introduction to Coding Theory. 2^{nd} edition, Springer, Graduate Texts in Mathematics, Berlin 1991.

[8] T. Pornin: RFC 6979: Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA). August 2013; https://tools.ietf.org/html/rfc6979#section-3

[9] W. Schindler, K. Itoh: Exponent Blinding Does not Always Lift (Partial) SPA Resistance to Higher-Level Security. In: J. Lopez, G. Tsudik (eds.): Applied Cryptography and Network Security — ACNS 2011, Springer, LNCS 6715, Berlin 2011, 73 – 90.

[10] W. Schindler, A. Wiemers: Power Attacks in the Presence of Exponent Blinding. J Cryptogr Eng 4 (2014), 213–236. online http://dx.doi.org/10.1007/s13389-014-0081-y