Recovering Secrets From Prefix-Dependent Leakage

Houda Ferradi[1], Rémi Géraud[2], Sylvain Guillet[3],
David Naccache[2], and Mehdi Tibouchi[1]

[1] NTT Secure Platform Laboratories
{ferradi.houda,tibouchi.mehdi}@lab.ntt.co.jp
[2] École normale supérieure
{remi.geraud,david.naccache}@ens.fr
[3] Telecom-ParisTech
sylvain.guilley@telecom-paristech.fr

Abstract. We discuss how to recover a secret bitstring given partial information obtained during a computation over that string, assuming the computation is a deterministic algorithm processing the secret bits sequentially. That abstract situation models certain types of side-channel attacks against discrete logarithm and RSA-based cryptosystems, where the adversary obtains information not on the secret exponent directly, but instead on the group or ring element that varies at each step of the exponentiation algorithm.

Our main result shows that for a leakage of a single bit per iteration, under suitable statistical independence assumptions, one can recover the whole secret bitstring in polynomial time. We also discuss how to cope with imperfect leakage, extend the model to k-bit leaks, and show how our algorithm yields attacks on popular cryptosystems such as (EC)DSA.

Keywords: Galton–Watson process, discrete logarithm problem, cryptanalysis.

1 Introduction

Many cryptographic algorithms iterate over a secret sequence of bits. This is the case for example in typical implementations of the exponentiation algorithms used in discrete logarithm and RSA-based cryptosystems. In such cases, one may hope to learn about the secret bits by observing side-channel leakage during computation. This is the basis of many side-channel attacks based on simple power analysis (SPA), including Kocher's seminal work [Koc96, KJJR11] and many followups.

However, there are cases in which the relationship between computation and leakage is nontrivial. For instance, if we consider a double-and-add scalar multiplication on an elliptic curve, the leakage of one bit of the x-coordinate of each intermediate point in the computation should be enough in an information-theoretic sense to recover the secret scalar, but it seems hard to write down an expression of the scalar in terms of that leakage. We also note that such a leakage can in fact occur in concrete attack settings: we discuss one such setting in Appendix A, where an attacker does obtain this type of leakage using a so-called "hardware trojan horse".

Although the relationship between the leakage and the secret can be quite involved, the structure of the algorithm, which iterates over secret bits sequentially, ensures that

the leakage at the i-th iteration depends only on the first i bits of the secret. If the algorithm is deterministic and the leakage is perfect, the leakage at the i-th iteration can therefore be viewed as a deterministic function of the i-bit prefix of the secret bitstring. This leads us to define the following general problem: recover a bitstring $s \in \{0,1\}^n$ given the leakage vector:

$$f(s) = \Big(f_1(s_{[1:1]}), f_2(s_{[1:2]}), \ldots, f_n(s_{[1:n]})\Big) \in \{0,1\}^n$$

where $s_{[1:i]} \in \{0,1\}^i$ is the i-bit prefix of s, and the functions f_i are regarded as known, independent random functions with values in $\{0,1\}$.

We show that this problem can be solved in expected polynomial time, in the sense that there is an algorithm with expected polynomial running time which outputs the list of all possible solutions to the problem (which is, in particular, of expected polynomial length). We also initiate the study of what happens when the leak is imperfect (e.g., due to noise considerations), and when more than a single bit of leakage is known to the attacker.

From a side-channel perspective, the algorithm we consider is a "single-trace" attack, in the sense that it recovers the secret from the leakage of a single execution of the target algorithm. In particular there is no notion of adaptive queries from the adversary in that context, which sets us apart from such questions as hardcore bits, and allows our attack to work in the presence of a large class of classical side-channel countermeasures, often designed to protect against multi-trace attacks like differential power analysis.

2 The secret prefix random leakage problem

Let s be an n-bit secret. Let $f_1, \ldots, f_n$ be functions $f_j : \{0,1\}^j \to \{0,1\}^k$, which are modeled as independent random oracles. Let $s_{[1:i]}$ denote the i-bit prefix of s for every $i = 1, \ldots, n$. This paper considers the following problem.

Definition 1 (k-leak recovery problem). *Given the functions f_j and the vector*

$$f(s) = \Big(f_1(s_{[1:1]}), f_2(s_{[1:2]}), \ldots, f_n(s_{[1:n]})\Big) \in \{0,1\}^{k \cdot n},$$

recover the value of s.

We will also discuss harder variants of this problem where the adversary is not given the vector $f(s)$ exactly, but only gets partial information about it.

These are abstractions of a situation arising in certain real-world side-channel attack settings, such as the one discussed in Appendix A.

The main concrete example of this problem that we consider throughout the paper is the case when the secret s is the secret exponent used in a discrete logarithm-based cryptosystem, which computes g^s using a (possibly side-channel protected) square-and-multiply algorithm, for some known group generator g. The function f_j is then some k-bit leakage function on the variable that contains the intermediate group element at iteration j of the square-and-multiply; for example, if the group is an elliptic curve, f_j could be the k lowest order bit of the x-coordinate of the intermediate curve point at iteration j.

3 A polynomial time algorithm for 1-bit leaks

We first consider the special case of the problem described in Definition 1 when $k = 1$. In this case already, we expect the problem to be tractable in an information-theoretic sense, since we get n independent bits of information on the n-bit secret s. However, recovery is not a priori trivial.

A simple approach to solve the problem is to simply reconstruct the entire list of all possible i-bit prefixes of s compatible with the provided leakage $f(s)$, successively for $i = 1, 2, \ldots, n$. This amounts to building a binary tree of possible prefixes as in Figure 1, starting from the empty string ε at the root, and with i-bit prefixes at level i. Going down one level, we double the set of candidate prefixes of the secret exponent by extending them either by 0 or 1, and then remove all the candidates that are incompatible with the new bit of information. Our key observation is that, under the right conditions, this pruning compensates the tree's growth, so that the algorithm terminates in expected polynomial time.

Concretely, upon extending a candidate with a bit, the probability that the new candidate is correct is expected to be $1/2$, independently for all candidates. In particular, a node in the tree of possible candidates should have 0, 1 or 2 children according as whether none, either, or both of its extensions by 0 and 1 are compatible with observations; this happens with probability $1/4$, $1/2$ and $1/4$ respectively for all nodes, independently. Thus, our recovery algorithm is a search in a Galton–Watson tree with $p_1 = 1/2$, $p_0 = p_2 = 1/4$ and $p_k = 0$ for all $k > 2$ in the sense of the following definition.

Definition 2 (Galton–Watson process). *A* Galton–Watson process $\{Z_n\}_{n\geq 0}$ *with offspring distribution* $F = \{p_k\}_{k\geq 0}$ *is a discrete-time Markov chain taking values in the set* $\mathbb{Z}^+$ *of non negative integers, and whose transition probabilities are as follows:*

$$\Pr\left[Z_{n+1} = k \mid Z_n = m\right] = p_k^{*m},$$

where $\{p_k^{*m}\}$ *denotes the* m*-th convolution power of the distribution* $\{p_k\}$*, i.e. the conditional distribution of* Z_{n+1} *given that* $Z_n = m$ *is the distribution of the sum of* m *i.i.d. random variables each with distribution* $\{p_k\}$*. A* Galton–Watson tree *is a random tree with offspring distribution* $\{p_k\}$ *is a random tree in which each node independently has* k *offspring with probability* p_k *for all* $k \geq 0$*; it satisfies that the number* Z_n *of nodes at level* n *is given by a Galton–Watson process.*

In our case, each node in the tree has on average $\mu = \sum_{k=0}^{\infty} kp_k = 1$ child: in other words, we have a *critical* Galton–Watson process, so that $\mathbb{E}[Z_n] = \mu^n \mathbb{E}[Z_0] = 1$. In

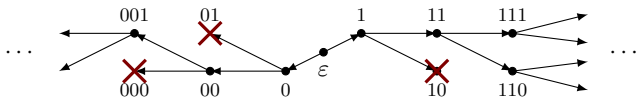

Fig. 1. Illustration of the algorithm as it runs: starting from the empty prefix ε (in the center), candidate prefixes are generated and appended as children. Prefixes that are incompatible with observations $f(x)$ are pruned.

Table 1. Simulation of the discrete logarithm computation in the 1-leak model, in the discrete log groups specified in RFC 5114. The leakage bit is recovered with probability p (see Section 4 for the case $p < 100\%$). Simulation in Sage run on a single core of Xeon E5-2697v3 workstation. Average over 100 trials for each parameter set.

	$p = 100\%$		$p = 99\%$		$p = 97\%$		$p = 95\%$	
	Space	CPU time	Space	CPU time	Space	CPU time	Space	CPU time
160-bit group	$6.6 \cdot 10^3$	50 ms	$1.4 \cdot 10^4$	113 ms	$4.8 \cdot 10^4$	430 ms	$5.2 \cdot 10^5$	5.8 s
224-bit group	$1.3 \cdot 10^4$	140 ms	$2.6 \cdot 10^4$	300 ms	$2.9 \cdot 10^5$	4.3 s	$4.3 \cdot 10^6$	73 s
256-bit group	$1.8 \cdot 10^4$	220 ms	$4.7 \cdot 10^4$	600 ms	$1.4 \cdot 10^6$	25 s	—	—

particular, the size of the full search space, which is given by $Z_1 + \cdots + Z_n$, does not undergo a combinatorial explosion. There are several ways to implement the Galton–Watson simulation; one possibility is to keep a pool of candidates from which new candidates are generated and pruned. This gives the algorithm of Figure 2.

We implemented this algorithm in the setting of square-and-multiply leaks in group exponentiations discussed in Section 2; implementation results are provided in (the first column of) Table 1. We can see in the table that the size of the search space increases *quadratically* rather than linearly with the bit length n of the exponent (despite the fact that $\mathbb{E}[Z_1 + \cdots + Z_n] = n$). This is because we are looking at a Galton–Watson tree *conditioned* on having at least one node at depth n, and we can show that $\mathbb{E}[Z_1 + \cdots + Z_n \mid Z_n \neq 0] = \Omega(n^2)$: see the discussion below. Nevertheless, our attack is polynomial and very practical for cryptographic sized problems, as exemplified by the simulations in Table 1.

Our algorithm is reminiscent of the cold boot attack of Heninger and Shacham against factorization [HS09] and its numerous follow-ups, such as [HMM10, PPS12,

Secret recovery from 1-bit leakage information:
Input: $f(s) = (y_1, \ldots, y_n)$.
Output: set X_n of all $s^* \in \{0,1\}^n$ such that $f(s^*) = f(s)$.

1. $X_0 \leftarrow \{\varepsilon\}$
2. **for** $i = 1, \ldots, n$
3. $\quad X_i \leftarrow \varnothing$
4. $\quad$ **for each** $\pi \in X_{i-1}$
5. $\quad\quad$ **if** $f_i(\pi \,\|\, 0) = y_i$ **then** $X_i \leftarrow X_i \cup \{\pi \,\|\, 0\}$
6. $\quad\quad$ **if** $f_i(\pi \,\|\, 1) = y_i$ **then** $X_i \leftarrow X_i \cup \{\pi \,\|\, 1\}$
7. **return** X_n

Fig. 2. Recovery of a secret in the 1-bit leak model

KSI13, KH14]. This is interesting, as these cold boot attacks do not really have a natural polynomial-time counterpart in the discrete logarithm setting (even the attack of Poettering and Sibborn [PS15] is basically exponential). Applied to a leaky exponentiation algorithm, our algorithm of side-channel provides such a counterpart. Moreover, like in the extensions of the original attack of Heninger and Shacham, one can consider variants of our attack model in which the leak is altered (bit flips, or analog noise, rather than erasures). We discuss some generalizations of that nature in the next section.

Galton–Watson conditioning and search space growth. We now explain why the search space for the critical Galton–Watson search described above is found to increase quadratically (see Table 1). This is due to the fact that our Galton–Watson tree is guaranteed to have a node at depth n. In other words, the average search space size that we want to estimate is $\mathbb{E}[Z_1 + \cdots + Z_n \mid Z_n \neq 0]$, where Z_i is the number of nodes at depth i. Our analysis relies on the following result.

Proposition 1. *Consider a Galton–Watson process $\{Z_n\}_{n\geq 0}$ with $Z_0 = 1$, with offspring distribution $\{p_k\}_{k\geq 0}$, which is* critical*, i.e. $\mu = \sum_{k\geq 0} kp_k = 1$, and non trivial, in the sense that $p_1 \neq 1$. Denote by f the generating function of the offspring distribution:*

$$f(x) = \sum_{k=0}^{+\infty} p_k x^k.$$

and assume that $f''(1) < +\infty$. Then the following asymptotic estimate holds:

$$\Pr[Z_n \neq 0] \underset{n\to+\infty}{\sim} \frac{2}{f''(1)} \cdot \frac{1}{n}.$$

Proof. Note first that under the assumptions of the theorem, we have $f(1) = 1$ (because $\{p_k\}$ is a probability distribution) and $f'(1) = 1$ (because of criticality). As a result, we claim that $f''(x) > 0$ for $x > 0$. Indeed, $f''(x) = \sum_{k\geq 2} k(k-1)p_k x^{k-2}$ is certainly non negative, and can only vanish if $p_k = 0$ for all $k \geq 2$. But if that were the case, we would get $p_1 = f'(1) = 1$, contradicting non triviality.

As a consequence, we must have $f(x) > x$ for all $x \in [0, 1)$. Indeed, the function $g(x) = f(x) - x$ satisfies $g'' = f''$, and is thus strictly convex over $[0, 1]$. Hence $g' = f' - 1$ is monotonically increasing, and it vanishes as 1, hence $g' < 0$ over $[0, 1)$. By the same argument, this implies $f(x) > x$ for all $x \in [0, 1)$ as required.

Now, let $u_n = \Pr[Z_n = 0]$ and $v_n = 1 - u_n = \Pr[Z_n \neq 0]$. By definition of the Galton–Watson process, the sequence (u_n) satisfies the recurrence relation $u_0 = 1$ and $u_{n+1} = f(u_n)$. By the argument above, the sequence (u_n) is strictly increasing, and since it is bounded by 1, it must converge to the only fixed point of f in $[0, 1]$, which is 1. In particular, v_n tends to 0, and using the Taylor expansion of f at 1, we get:

$$\begin{aligned} v_{n+1} = 1 - u_{n+1} = 1 - f(1 - v_n) &= 1 - \Big(f(1) - f'(1)v_n + \frac{f''(1)}{2}v_n^2 + o(v_n^2)\Big) \\ &= v_n \cdot \Big(1 - \frac{f''(1)}{2}v_n + o(v_n)\Big). \end{aligned}$$

Raising this relation to the power -1 yields:

$$\begin{aligned} v_{n+1}^{-1} &= v_n^{-1} \cdot \left(1 - \frac{f''(1)}{2} v_n + o(v_n)\right)^{-1} = v_n^{-1} \cdot \left(1 + \frac{f''(1)}{2} v_n + o(v_n)\right) \\ &= v_n^{-1} + \frac{f''(1)}{2} + o(1). \end{aligned}$$

This shows that $v_{n+1}^{-1} - v_n^{-1}$ converges to $\frac{f''(1)}{2}$ as $n \to +\infty$, and hence, by Cesàro's lemma:

$$\lim_{n \to +\infty} \frac{v_n^{-1}}{n} = \frac{f''(1)}{2}$$

which is exactly what we needed to prove.

From Proposition 1, we can easily obtain the asymptotic behavior of the conditional expectation $\mathbb{E}[Z_n \mid Z_n \neq 0]$. Indeed, we have:

$$\begin{aligned} 1 = \mathbb{E}[Z_n] &= \mathbb{E}[Z_n \mid Z_n = 0] \cdot \Pr[Z_n = 0] + \mathbb{E}[Z_n \mid Z_n \neq 0] \cdot \Pr[Z_n \neq 0] \\ &= 0 + \mathbb{E}[Z_n \mid Z_n \neq 0] \cdot \Pr[Z_n \neq 0]. \end{aligned}$$

As a result:

$$\mathbb{E}[Z_n \mid Z_n \neq 0] = \frac{1}{\Pr[Z_n \neq 0]} \underset{n \to +\infty}{\sim} \frac{f''(1)}{2} \cdot n.$$

In our case of interest, $p_0 = p_2 = 1/4$, $p_1 = 1/2$ and $p_k = 0$ for $k \geq 2$, so that $f''(1) = 1/2$. Thus $\mathbb{E}[Z_n \mid Z_n \neq 0] \sim n/4$.

What we want to estimate is $\mathbb{E}[Z_1 + \cdots + Z_n \mid Z_n \neq 0] = \sum_{j=1}^{n} \mathbb{E}[Z_j \mid Z_n \neq 0]$. To do so, we can observe that

$$\mathbb{E}[Z_j \mid Z_j \neq 0] \leq \mathbb{E}[Z_j \mid Z_n \neq 0] \leq \mathbb{E}[Z_n \mid Z_n \neq 0].$$

Hence the series $\sum_{j=1}^{n} \mathbb{E}[Z_j \mid Z_n \neq 0]$ is asymptotically bounded below by $\sum_{j=1}^{n} j/4 \sim n^2/8$ and above by $n \cdot n/4 \sim n^2/4$. As a result, we obtain

$$\mathbb{E}[Z_1 + \cdots + Z_n \mid Z_n \neq 0] = \Omega(n^2)$$

as required, with the implied constant between $1/8$ and $1/4$. This reflects the results of Table 1.

Dealing with imperfect information. A central argument in the polynomial-time execution of the algorithm in the noiseless case is that every additional bit of information allowed to (safely) prune on average half of the candidates, compensating exactly the tree's growth. This argument no longer holds under imperfect information, as pruning may eliminate correct candidates (or conversely keep in the pool incorrect ones), resulting in a blowup and/or incorrect results.

If we have less than one bit of information at each generation, the population grows exponentially (i.e. $\mu > 1$). In Table 1 we indicate the probability p of the leaked bit being correct, and simulate the corresponding blowup when p is slightly less that 100%.

Table 2. Computation in the k-leak model, in the 256-bit discrete log group of RFC 5114, in the all-or-nothing model (at each iteration, all k bits are recovered with probability p, and nothing is recovered otherwise). The probability $p_{\text{crit}} = 2^{k-1}/(2^k - 1)$ is the theoretical bound for a polynomial-size search tree. Average over 100 trials for each parameter set.

		$p = 80\%$		$p = 70\%$		$p = 60\%$		$p = 50\%$	
	p_{crit}	Space	CPU time	Space	CPU time	Space	CPU time	Space	CPU time
$k = 2$	66%	990	14 ms	1500	16 ms	$2.3 \cdot 10^4$	250 ms	—	—
$k = 3$	57%	280	5 ms	400	7 ms	1120	13 ms	$1.1 \cdot 10^4$	120 ms
$k = 4$	53%	240	5 ms	330	6 ms	630	9 ms	2010	25 ms

Table 3. Computation in the k-leak model, in the 256-bit discrete log group of RFC 5114, in the bitwise model (at each iteration, each of the k bits is recovered with probability p). The probability $p_{\text{crit}} = 2 \cdot (1 - 2^{-1/k})$ is the theoretical bound for a polynomial-size search tree. Average over 100 trials for each parameter set.

		$p = 60\%$		$p = 50\%$		$p = 40\%$		$p = 30\%$	
	p_{crit}	Space	CPU time	Space	CPU time	Space	CPU time	Space	CPU time
$k = 2$	59%	3500	50 ms	—	—	—	—	—	—
$k = 3$	41%	550	10 ms	890	15 ms	$5.1 \cdot 10^4$	870 ms	—	—
$k = 4$	32%	370	9 ms	480	11 ms	930	18 ms	$1.2 \cdot 10^4$	190 ms

4 Recovery with imperfect k-bit leakage

We now turn our attention to a more general setting, in which the leakage at each iteration consists of k-bit values, but is not always recovered exactly. More precisely, we consider two possible models: in the simpler "all-or-nothing" model, for each step j, the k-bit leakage at step j is recovered in its entirety with some probability p, and not at all otherwise; in the more involved "bitwise" model, each bit of leakage independently has probability p of being recovered.

The algorithm of Figure 2 extends directly to both settings: pruning is simply done with all the available information at each step instead of just a single bit. The question is then to determine under which condition on k and p the number of candidates and the running time are expected to remain polynomial. We address this question below.

All-or-nothing model. In the *all-or-nothing model*, at each iteration, all k bits of leakage are recovered exactly with some probability p, and with probability $1-p$, no information is available at all.

Mathematically, this yields a *generation-dependent* Galton–Watson process: with probability p there will be an average of 2^{1-k} offsprings, and with probability $1-p$ there will be an average of 2 offsprings. It turns out that the expected number of nodes at depth n is then exactly the product of the expected numbers of offsprings at each generation [Fea72, Proposition 4]. As a result, after n iterations, of which ℓ were successful extractions (we learned all k bits) and $n-\ell$ failed (we learned nothing), the average number of offsprings is $\mu_\ell = (2^{-k+1})^\ell \cdot 2^{n-\ell}$. Naturally, we do not know ℓ, so that we have to compute the weighted sum over all possible values:

$$\begin{aligned}\mu &= \sum_{\ell=0}^{n} \binom{n}{\ell} p^\ell (1-p)^{n-l} \mu_\ell = \sum_{\ell=0}^{n} \binom{n}{\ell} (2^{-k+1}p)^\ell (2-2p)^{n-\ell} \\ &= (2^{-k+1}p + 2 - 2p)^n = \mu_0^n. \end{aligned} \tag{1}$$

Our algorithm runs in polynomial time exactly when $\mu_0 \leq 1$, i.e., if and only if $p \geq p_{\text{crit}}$ where $p_{\text{crit}} = 2^{k-1}/(2^k-1)$. For instance, in the 4-leak model, this requires a success probability of $8/15 \approx 54\%$. Simulation results for this scenario are given in Table 2.

Bitwise model. In the *bitwise model*, at each iteration, each bit of leakage (among k bits in total) is recovered independently with probability p.

Assume that we recover j_1 bits at iteration 1, j_2 bits at iteration 2 and so on. The expected number of leaves in that case is then $2^{1-j_1} \cdots 2^{1-j_n}$. Now since we learn each bit with probability p, the probability of recovering j bits of leakage at any fixed iteration is exactly $\binom{k}{j} p^j (1-p)^j$. As a result, the expected number of leaves in the tree when accounting for all possibilities for $j_1, \ldots, j_n$ is given by

$$\begin{aligned}\mu &= \sum_{j_1} \cdots \sum_{j_n} \binom{k}{j_1} p^{j_1} (1-p)^{j_1} \cdots \binom{k}{j_n} p^{j_n} (1-p)^{j_n} 2^{1-j_1} \cdots 2^{1-j_n} \\ &= 2^n \sum_{j_1=0}^{k} \binom{k}{j_1} (p/2)^{j_1} (1-p)^{j_1} \cdots \sum_{j_n=0}^{k} \binom{k}{j_n} (p/2)^{j_1} (1-p)^{j_n} \\ &= 2^n \left(1 - \frac{p}{2}\right)^{kn} = \mu_0^n \end{aligned} \tag{2}$$

this time with $\mu_0 = 2(1-p/2)^k$. The smallest value of p for which our attack runs in polynomial time is therefore $p_{\text{crit}} = 2(1-2^{-1/k})$. For example, when $k=4$, the condition is $p \gtrsim 32\%$. Simulation results for this attack are given in Table 3. Note that the critical probability is significantly lower in this case, despite the fact that we obtain the same number of bits of leakage per iteration on average. This is due to the fact that we obtain usable information more often, and as a result, we can prune the search tree earlier.

5 Application to (EC)DSA

What we have described so far is a generic attack against cryptographic schemes. Particular schemes among them, however, can be vulnerable to stronger attacks. For example,

it is possible to efficiently break (EC)DSA (or more generally Schnorr-like signatures) in the 1-leak model, even if the presence of noise causes the corresponding leakage bit to be recoverable with probability $p < 1$ (this is in stark contrast with the generic attack above, in which a less than perfect recovery makes the search space in the 1-leak case exponentially large).

A first possible approach is to get a one bit leak about the nonce. That bit of information can then used to recover the secret signing key given sufficiently many signatures, using the statistical attack of Bleichenbacher [Ble00, MHMP14]. The attack with a single bit of information requires many signatures, but Aranha et al. [AFG+14] have shown it to be practical at least against 160-bit groups. And if the leakage is recoverable only with probability p, the same attack can be mounted by simply increasing the number of signatures by a factor of $1/p$ and throwing away those for which the bit is unrecoverable.

A more efficient approach is to combine this leak with nonce-based lattice attacks on (EC)DSA [HS01, NS03]. In the 1-leak model, where we recover the leakage bit with probability $p < 1$, although we do not recover the entire nonce we are still be able to learn a prefix of it (i.e. the MSBs of the nonce, for a left-to-right square-and-multiply) with good probability. And if we have sufficiently many signatures for which we know the MSBs of the nonce, standard lattice techniques will recover the signing key.

More precisely, consider the first bit of the nonce (the MSB), which may be 0 or 1. With probability p, we learn one leakage bit, which may itself be compatible (with equal probability) with that bit being 0, 1, or both (but not neither of them, because our search tree is conditioned on knowing that a solution actually exists). If it is both, we learn nothing, but otherwise we learn that MSB: this happens with probability $2p/3$. And in that case, we can make the same argument for the second bit, and then the third, and so on. With probability at least $(2p/3)^k$, we will learn the k most significant bits of the nonce.[4]

Now how many MSBs do we need to mount the lattice attack? This depends on the bit size n of the group, and the maximum lattice dimension $d_{\max}$ in which we can reliably find the shortest vector of a random lattice (nowadays, $d_{\max} = 100$ is a reasonable rule-of-thumb using reduction algorithms like BKZ 2.0 [CN11]; academic records on the SVP Hall of Fame go all the way to dimension 150 as of this writing). Indeed, it is standard that we can recover the signing key by computing the SVP in a lattice of dimension $d = n/(k - c)$, where $c = \log_2 \sqrt{\pi e/2}$, so the lowest usable k is given by $k = \lceil c + n/d_{\max} \rceil$. And we then need d signatures with k known nonce MSBs to carry out the attack. This can be obtained by collecting $m = \left(\frac{3}{2p}\right)^k \cdot d$ signatures with the leakage above, and keeping those for which the leakage is enough to learn the k most significant bits.

In a 256-bit group and with $d_{\max} = 100$, we have $k = 4$ and $d \approx 87$. If the probability of learning a bit of the leakage is $p = 1/2$, we thus get $m \approx 7000$: collecting 7000 signatures should yield enough signatures with 4 known MSBs to mount the attack and recover the signing key. This is much better than the Bleichenbacher approach, which would require billions of signatures at this group size, and a fortiori better than

[4] This is actually a lower bound, since the search tree can in principle grow and then collapse back to a single node at depth k, but that lower bound is sufficient for our purposes.

trying to apply the generic algorithm, since the expected size of the search space in that case is at least $\mu = (p + 2 - 2p)^n = (3/2)^{256} \approx 2^{150}$ by (1).

Remark 1. Bauer and Vergnaud [BV15] have recently cryptanalyzed (EC)DSA-type schemes in the presence of leakage on randomly-located bits of the nonces. Although seemingly relevant, this attack does not apply to our setting, because the one bit leakage will not reveal many randomly located bits of the nonce: to learn a bit with certainty, it should be the only bit compatible when extending all previous candidate prefixes, and this is exponentially unlikely to happen when the set of candidate prefixes is already large.

6 Countermeasures and perspectives

In this section, we discuss the effectiveness of various possible side-channel countermeasures used in implementations of discrete logarithm-based cryptosystems with respect to the attack considered in this paper. We also suggest possible perspectives for further work.

Protecting exponentiation algorithms. As we have seen, our algorithm gives rise to efficient side-channel attacks on exponentiation algorithms in cryptographic groups (assuming of course that the "prefix-dependent leakage" it relies on can be collected in practice, possibly using an approach like that of Appendix A). Interestingly, the corresponding side-channel attacks is unaffected by several large classes of common side-channel countermeasures deployed in implementations of discrete logarithm-based cryptosystems.

Indeed, a first important family of countermeasures used in that setting includes modification to the exponentiation algorithm that make it regular, in the sense that the same types of operations are carried out at each iteration, regardless of whether the bit of the secret at that iteration is 0 or 1. This includes the square-and-multiply-always algorithm [CFG+11], the Montgomery ladder [Mon87], and the use of elliptic curves with complete addition laws [BL07, BL09, RCB16]. None of these approach thwart our attack, since it does not rely on the particular control flow of the algorithm but only on essentially arbitrary leakage information on intermediate values.

Another family of side-channel countermeasures used in exponentiation algorithm relies on randomizing the secret exponent. This includes exponent blinding [Cor99, Section 5.1], where a random multiple of the group order is added to the exponent, and exponent splitting [CJRR99, CJ01], where the exponent s is written as a random sum $s_0 + \cdots + s_d$ modulo the group order, and the exponentiation is computed as $g^s = g^{s_0} \cdots g^{s_d}$. Again, neither of these approaches thwart our attack. Indeed, it is sufficient to recover the longer blinded exponent in full in the case of blinding, or all of the additive shares in the case of splitting; this increases the complexity of the attack slightly, but the recovery algorithm remains polynomial time (at least for a constant number of additive shares in the case of splitting).

A countermeasure that *does* work, however, consists in randomizing the base point of the exponentiation [JT01]. Indeed, to fix ideas, if for example g^x is computed as

$g_1^x \cdot g_2^x$ where $g = g_1 \cdot g_2$ is a random decomposition, then it is no longer possible for the attacker to recompute the leakage functions f_j locally and therefore to build to corresponding Galton–Watson tree. This applies similarly to other base point randomization techniques using, e.g., isomorphic elliptic curves or field isomorphisms, as well as related techniques like the use of randomized projective coordinates [Cor99, Section 5.3]. In all of these cases, our approach fails because, from the adversary's viewpoint, the leakage functions f_j are no longer deterministic: they depend probabilistically on the randomness used to blind the base point of the exponentiation (resp. rerandomize projective coordinates).

Open problems. There are several generalizations of the problems considered in this paper that would be natural to consider and explore in further work.

A first one is the extension to algorithm that iterate over secrets a few bits at a time instead of one by one. This is typically the case for k-ary or window-based exponentiation algorithms. This also includes cryptographic computations based on non-binary secrets (such as the use of *signed* binary expansions like non-adjacent forms). Our approach should natural generalize to those settings, but the leakage bounds to achieve polynomial time recovery are of course different.

It would also be interesting to consider more general noise models for the leakage. For example, one could ask how to solve the following problem, with an arbitrary noise distribution: with the same notation as Definition 1, recover the secret s from:

$$f(s) \oplus \boldsymbol{e} = \Big(f_1(s_{[1:1]}) \oplus e_1, \ldots, f_n(s_{[1:n]}) \oplus e_n\Big) \in \{0,1\}^{k \cdot n},$$

where $\boldsymbol{e} = (e_1, \ldots, e_n)$ is a vector of independent identically distributed noise values sampled from some fixed distribution χ over $\{0,1\}^k$.

Of course, for some distributions, the problem is clearly intractable (e.g. when χ is uniform), and in general one can only hope to recover the secret with some probability, but the tree-based approach should generalize naturally, provided that it is combined with a suitable algorithm for pruning branches that have a low probability of being consistent with the leakage, instead of branches that are literally incompatible. Quantifying that intuition and obtaining concrete bounds to ensure expected polynomial time recovery with high probability are left as open problems for future work.

References

AFG+14. Diego F. Aranha, Pierre-Alain Fouque, Benoît Gérard, Jean-Gabriel Kammerer, Mehdi Tibouchi, and Jean-Christophe Zapalowicz. GLV/GLS decomposition, power analysis, and attacks on ECDSA signatures with single-bit nonce bias. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 262–281. Springer, Heidelberg, December 2014.

BL07. Daniel J. Bernstein and Tanja Lange. Faster addition and doubling on elliptic curves. In Kaoru Kurosawa, editor, *ASIACRYPT 2007*, volume 4833 of *LNCS*, pages 29–50. Springer, Heidelberg, December 2007.

BL09. Daniel J. Bernstein and Tanja Lange. Complete addition laws for elliptic curves. Talk at Algebra and Number Theory Seminar (Universidad Autonomo de Madrid), 2009. http://cr.yp.to/talks/2009.04.17/slides.pdf.

Ble00. Daniel Bleichenbacher. On the generation of one-time keys in DL signature schemes. Presentation at IEEE P1363 Working Group meeting, 2000.

BV15. Aurélie Bauer and Damien Vergnaud. Practical key recovery for discrete-logarithm based authentication schemes from random nonce bits. In Tim Güneysu and Helena Handschuh, editors, *CHES 2015*, volume 9293 of *LNCS*, pages 287–306. Springer, Heidelberg, September 2015.

CFG+11. Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet, and Vincent Verneuil. Square always exponentiation. In Daniel J. Bernstein and Sanjit Chatterjee, editors, *INDOCRYPT 2011*, volume 7107 of *LNCS*, pages 40–57. Springer, Heidelberg, December 2011.

CJ01. Christophe Clavier and Marc Joye. Universal exponentiation algorithm. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *CHES 2001*, volume 2162 of *LNCS*, pages 300–308. Springer, Heidelberg, May 2001.

CJRR99. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 398–412. Springer, Heidelberg, August 1999.

CN11. Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 1–20. Springer, Heidelberg, December 2011.

Cor99. Jean-Sébastien Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In Çetin Kaya Koç and Christof Paar, editors, *CHES'99*, volume 1717 of *LNCS*, pages 292–302. Springer, Heidelberg, August 1999.

Fea72. Dean H. Fearn. Galton-watson processes with generation dependence. In *Proceedings of the Sixth Berkeley Symposium on Mathematical Statistics and Probability (Univ. California, Berkeley, Calif., 1970/1971)*, volume 4, pages 159–172, 1972.

HMM10. Wilko Henecka, Alexander May, and Alexander Meurer. Correcting errors in RSA private keys. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 351–369. Springer, Heidelberg, August 2010.

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

HS09. Nadia Heninger and Hovav Shacham. Reconstructing RSA private keys from random key bits. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 1–17. Springer, Heidelberg, August 2009.

JT01. Marc Joye and Christophe Tymen. Protections against differential analysis for elliptic curve cryptography. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *CHES 2001*, volume 2162 of *LNCS*, pages 377–390. Springer, Heidelberg, May 2001.

KH14. Noboru Kunihiro and Junya Honda. RSA meets DPA: Recovering RSA secret keys from noisy analog data. In Lejla Batina and Matthew Robshaw, editors, *CHES 2014*, volume 8731 of *LNCS*, pages 261–278. Springer, Heidelberg, September 2014.

KJJR11. Paul C. Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. Introduction to differential power analysis. *J. Cryptographic Engineering*, 1(1):5–27, 2011.

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

KSI13. Noboru Kunihiro, Naoyuki Shinohara, and Tetsuya Izu. Recovering RSA secret keys from noisy key bits with erasures and errors. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 180–197. Springer, Heidelberg, February / March 2013.

MH78. Ralph C. Merkle and Martin E. Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE Trans. Information Theory*, 24(5):525–530, 1978.

MHMP14. Elke De Mulder, Michael Hutter, Mark E. Marson, and Peter Pearson. Using Bleichenbacher's solution to the hidden number problem to attack nonce leaks in 384-bit ECDSA: extended version. *J. Cryptographic Engineering*, 4(1):33–45, 2014.

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

NS03. Phong Q. Nguyen and Igor E. Shparlinski. The insecurity of the elliptic curve digital signature algorithm with partially known nonces. *Des. Codes Cryptography*, 30(2):201–217, 2003.

PPS12. Kenneth G. Paterson, Antigoni Polychroniadou, and Dale L. Sibborn. A coding-theoretic approach to recovering noisy RSA keys. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 386–403. Springer, Heidelberg, December 2012.

PS15. Bertram Poettering and Dale L. Sibborn. Cold boot attacks in the discrete logarithm setting. In Kaisa Nyberg, editor, *CT-RSA 2015*, volume 9048 of *LNCS*, pages 449–465. Springer, Heidelberg, April 2015.

RCB16. Joost Renes, Craig Costello, and Lejla Batina. Complete addition formulas for prime order elliptic curves. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 403–428. Springer, Heidelberg, May 2016.

A Prefix-dependent leaks from resistor-based hardware trojans

As a conceptual example (not necessarily a viable implementation) of a system from which one bit of leakage can be measured, we may consider the setup given in Figure 3. In this case, the attacker added a resistor between two output bits, and is monitoring power consumption. Whenever $Y_1 = Y_2$, the resistor has no effect; however if $Y_1 \neq Y_2$, a small but measurable current flow will dissipate energy, and this loss will be compensated for by the IC's power supply, resulting in a small and temporary increase in power consumption.

Since installing such a resistor would be done during manufacturing, but would in all likelihood remain with the tampered device unbeknownst to its user, we refer to this modification as a hardware trojan. In that respect, read amplifiers for embedded memory (such as SRAM, EEPROM, FLASH, etc.) are attractive targets because:

– they have a strong drive—this is by design, for the memory read to be reliable;

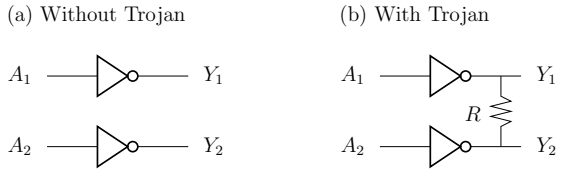

Fig. 3. A possible implementation of the one-bit leak oracle.

- they are easily spotted in a design (by an attacker) by searching for them on the boundary of memory blocks;
- they are placed in the layout side by side, which corresponds to our proof-of-concept example; and, most importantly,
- they carry information which is easily related to high-level variables and is hence cryptanalytically exploitable.

Building on that idea, a k-leaking system may be constructed by using the above construction several times: the signals would then add up. Denoting (x_i, y_i) the bitlines which the i-th resistor connects, the measurement is $\sum_i \|x_i \oplus y_i\|$, where $\|\cdot\|$ denotes the Hamming weight. While this may be sufficient in some settings, we may wish to distinguish between the different signals. One possibility is to give *sufficiently* different values to the resistances. As a result, we measure instead $\sum_{i=1}^{T} \rho_i z_i$ where T is the number of trojans, $z_i = \|x_i \oplus y_i\|$, and ρ_i is determined by the resistance R_i and is known. Finding the collection $(z_i)_{i=1}^{T} \in \{0,1\}^T$ is then simply solving a $(0,1)$-subset-sum problem. While this problem is known to be NP-complete in general, in practice T is very small—furthermore we are at liberty to choose ρ_i, so we may select a superincreasing sequence, for which the subset-sum problem is easy [MH78].

In practice, we may have to deal with noise resulting both from measurement and from the fact that we only know the resistance approximately in the first place. We also have to account for the limited range of resistances available. This reduces the practical number of leaks that can be accurately used simultaneously.

For instance, if we can only realize resistances in the range $10, \ldots, 100\,\mathrm{k}\Omega$, and noise levels impose a minimum separation of $\pm 5\,\mathrm{k}\Omega$, then we may use simultaneously 4 resistances $R_1 = 10$, $R_2 = 20$, $R_3 = 40$, $R_4 = 80\,\mathrm{k}\Omega$.