Just A Little Bit More

Joop van de Pol[1], Nigel P. Smart[1], and Yuval Yarom[2]

[1] Dept. Computer Science, University of Bristol, United Kingdom.
joop.vandepol@bristol.ac.uk,nigel@cs.bris.ac.uk
[2] School of Computer Science, The University of Adelaide, Australia.
yval@cs.adelaide.edu.au

Abstract. We extend the FLUSH+RELOAD side-channel attack of Benger et al. to extract a significantly larger number of bits of information per observed signature when using OpenSSL. This means that by observing only 25 signatures, we can recover secret keys of the **secp256k1** curve, used in the Bitcoin protocol, with a probability greater than 50 percent. This is an order of magnitude improvement over the previously best known result.
The new method of attack exploits two points: Unlike previous partial disclosure attacks we utilize all information obtained and not just that in the least significant or most significant bits, this is enabled by a property of the "standard" curves choice of group order which enables extra bits of information to be extracted. Furthermore, whereas previous works require direct information on ephemeral key bits, our attack utilizes the indirect information from the wNAF double and add chain.

1 Introduction

The Elliptic Curve Digital Signature Algorithm (ECDSA) is the elliptic curve analogue of the Digital Signature Algorithm (DSA). It has been well known for over a decade that the randomization used within the DSA/ECDSA algorithm makes it susceptible to side-channel attacks. In particular a small leakage of information on the emphermal secret key utlized in each signature can be amortized over a number of signatures to obtain the entire key.

Howgrave-Graham and Smart [12] showed that DSA is vulnerable to such partial ephemeral key exposure and their work was made rigourous by Nguyen and Shparlinski [19], who also extended these results to ECDSA [20]. More specifically, if, for a polynomially bounded number of random messages and ephemeral keys about $\log^{1/2} q$ least significant bits (LSBs) are known, the secret key α can be recovered in polynomial time. A similar result holds for a consecutive sequence of the most significant bits (MSBs), with a potential need for an additional leaked bit due to the paucity of information encoded in the most significant bit of the ephemeral key. When an arbitrary sequence of consecutive bits in the ephemeral key is known, about twice as many bits are required. The attack works by constructing a lattice problem from the obtained digital signatures and side-channel information, and then applying lattice reduction techniques such as LLL [14] or BKZ [21] to solve said lattice problem.

Brumley and co-workers employ this lattice attack to recover ECDSA keys using leaked LSBs (in [4]) and leaked MSBs (in [5]). The former uses a cache side-channel to extract the leaked information and the latter exploits a timing side-channel. In both attacks, a fixed number of bits from each signature is used and signatures are used only if the values of these bits are all zero. Signatures in which the value of any of these bits are one are ignored. Consequently, both attacks require more than 2,500 signatures to break a 160-bit private key.

More recently, again using a cache based side-channel, Benger et al. [2] use the LSBs of the ephemeral key for a wNAF (a.k.a. sliding window algorithm) multiplication technique. By combining a new side-channel called the FLUSH+RELOAD side-channel [24, 25], and a more precise lattice attack strategy, which utilizes all of the leaked LSBs from every signature, Benger et al. are able to significantly reduces the number

of signatures required. In particular they report that the full secret key of a 256-bit system can be recovered with about 200 signatures in a reasonable length of time, and with a reasonable probability of success.

In this work we extend the FLUSH+RELOAD technique of Benger et al. to reduce the number of required signatures by an order of magnitude. Our methodology abandons the concentration on extraction of bits in just the MSB and LSB positions, and instead focuses on all the information leaked by all the bits of the ephemeral key. In particular we exploit a property of the standardized elliptic curves as used in OpenSSL. Our method, just as in [2], applies the FLUSH+RELOAD side-channel technique to the wNAF elliptic curve point multiplication algorithm in OpenSSL.

ECDSA Using Standard Elliptic Curves: The domain parameters for ECDSA are an elliptic curve E over a field $\mathbb{F}$, and a point G on E, of order q. Given a hash function h, the ECDSA signature of a message m, with a private key $0 < \alpha < q$ and public key $Q = \alpha G$, is computed by:

- Selecting a random ephemeral key $0 < k < q$
- Computing $r = x(kG) \pmod q$, the X coordinate of kG.
- Computing $s = k^{-1}(h(m) + \alpha \cdot r) \pmod q$.

The process is repeated if either $r = 0$ or $s = 0$. The pair (r,s) is the signature.

To increase interoperability, standard bodies have published several sets of domain parameters for ECDSA [1, 7, 18]. The choice of moduli for the fields used in these standard curves is partly motivated by efficiency arguments. For example, all of the moduli in the curves recommended by FIPS [18] are generalised Mersenne primes [22] and many of them are pseudo-Mersenne primes [9]. This choice of moduli facilitates efficient modular arithmetic by avoiding a division operation which may otherwise be required.

A consequence of using pseudo-Mersenne primes as moduli is that, due to Hasse's Theorem, not only is the finite field field order close to a power of two, but so is the elliptic curve group order. That is, q can be expressed as $2^n + \varepsilon$, where $|\varepsilon| < 2^p$ for some $p \approx n/2$. We demonstrate that such curves are more susceptible to partial disclosure of ephemeral keys than was hitherto known. This property increases the amount of information that can be used from partial disclosure and allows for a more effective attack on ECDSA.

Our Contribution: We demonstrate that the above property of the standardized curves allows the utilization of far more leaked information, in particular some arbitrary sequences of consecutive leaked bits. In a nutshell, adding or subtracting q to or from an unknown number is unlikely to change any bits in positions between $p+1$ and n. Based on this observation we are able to use (for wNAF multiplication algorithms) all the information in consecutive bit sequences in positions above $p+1$. Since the standard curves utilize a small value of p, this provides a large amount of leaked information per signature. (Assuming one can extract the sequence of additions and doubles in an algorithm.) The same property also implies that techniques for mitigating side-channel attack, such as the scalar blinding suggested in [4, 16], do not protect bits in positions above $p+1$.

Prior works deal with the case of partial disclosure of consecutive sequences of bits of the ephemeral key. Our work offers two improvements: It demonstrates how to use partial information leaked from the double and add chains of the wNAF scalar multiplication algorithm [11, 17]. In most cases, the double and add chain does not provide direct information on the value of bits. It only identifies sequences of repeating bits without identifying the value of these bits. We show how to use this information to construct a lattice attack on the private key. Secondly, our attack does not depend on the leaked bits being consecutive. We use information leaked through the double and add chain even though it is spread out along the ephemeral key.

By using more leaked information and exploiting the above property of the elliptic curves, our attack only requires a handful of leaked signatures to fully break the private key. Our experiments show that the information leaked by perfect double and add chains of only 13 signatures is sufficient for recovering the 256 bit private key of the **secp256k1** curve used in the Bitcoin protocol, with probability greater than 50 percent. For the 521 bit curve **secp521r1**, 40 signatures are required. We further demonstrate that for the **secp256k1** case observing 25 signatures is highly likely to recover 13 perfect double and add chains. Hence, by observing 25 Bitcoin transactions using the same key, an attacker can expect to recover the private key. For most of the paper we discuss the case of perfect side channels which result in perfect double and add chains, then in Section 7 we show how this assumption can be removed in the context of a real FLUSH+RELOAD attack.

2 Background

In this section we discuss three basic procedures we will be referring to throughout. Namely the FLUSH+RELOAD side-channel attack technique, wNAF scalar multiplication method and the use of lattices to extract secret keys from triples. The side-channel information we obtain from executing the wNAF algorithm produces instances of the Hidden Number Problem (HNP) [3]. Since the HNP is traditionally studied via lattice reduction it is therefore not surprising that we are led to lattice reduction in our analysis.

2.1 The FLUSH+RELOAD Side-Channel Attack Technique

FLUSH+RELOAD is a recently discovered cache side-channel attack [24,25]. The attack exploits a weakness in the Intel implementation of the popular X86 architecture, which allows a spy program to monitor other programs' read or execute access to shared regions of memory. The spy program only requires read access to the monitored memory.

Unlike most cache side-channel attacks, FLUSH+RELOAD uses the Last-Level Cache (LLC), which is the cache level closest to the memory. The LLC is shared by the execution cores in the processor, allowing the attack to operate when the spy and victim processes execute on different cores. Furthermore, as most virtual machine hypervisors (VMMs) actively share memory between co-resident virtual machines, the attack is applicable to virtualized environment and works cross-VM.

```
Input: adrs—the probed address
Output: true if the address was accessed by the victim
begin
    evict(adrs)
    wait_a_bit()
    time ← current_time()
    tmp ← read(adrs)
    readTime ← current_time()-time
    return readTime < threshold
end
```

Algorithm 1: FLUSH+RELOAD Algorithm

To monitor access to memory, the spy repeatedly evicts the contents of the monitored memory from the LLC, waits for some time and then measures the time to read the contents of the monitored memory. See Algorithm 1 for a pseudo-code of the attack. FLUSH+RELOAD uses the X86 `clflush` instruction to evict

contents from the cache. To measure time the spy uses the `rdtsc` instruction which returns the time since processor reset measured in processor cycles.

As reading from the LLC is much faster than reading from memory, the spy can differentiate between these two cases. If, following the wait, the contents of memory is retrieved from the cache, it indicates that another process has accessed the memory. Thus, by measuring the time to read the contents of memory, the spy can decide whether the victim has accessed the monitored memory since the last time it was evicted.

To implement the attack, the spy needs to share the monitored memory with the victim. For attacks occurring within the same machine, the spy can map files used by the victim into its own address space. Examples of these files include the victim program file, shared libraries or data files that the victim accesses. As all mapped copies of files are shared, this gives the spy access to memory pages accessed by the victim. In virtualized environments, the spy does not have access to the victim's files. The spy can, however, map copies of the victim files to its own address space, and rely on the VMM to merge the two copies using page de-duplication [13, 23]. It should be pointed that, as the LLC is physically tagged, the virtual address in which the spy maps the files is irrelevant for the attack. Hence, FLUSH+RELOAD is oblivious to address space layout randomization [15].

This sharing only works when the victim does not make private modifications to the contents of the shared pages. Consequently, all published FLUSH+RELOAD attacks target executable code pages, monitoring the times the victim executes specific code. The spy typically divides time into fixed width time slots. In each time slot the spy monitors a few memory locations and records the times that these locations were accessed by the victim. By reconstructing a trace of victim access, the spy is able to infer the data the victim is operating on. Prior works used this attack to recover the private key of GnuPG RSA [25] as well as for recovering the ephemeral key used in OpenSSL ECDSA signatures either completely, for curves over binary fields [24], or partially, for curves over prime fields [2].

2.2 The wNAF Scalar Multiplication Method

Several algorithms for computing the scalar multiplication kG have been proposed. One of the suggested methods is to use the *windowed nonadjacent form* (wNAF) representation of the scalar k, see [11]. In wNAF a number is represented by a sequence of digits k_i. The value of a digit k_i is either 0 or an odd number $-2^w < k_i < 2^w$, with each pair of non-zero digits separated by at least w zero digits. The value of k can be calculated from its wNAF representation using $k = \sum 2^i \cdot k_i$. See Algorithm 2 for a method to convert a scalar k into its wNAF representation. We use $|\cdot|_x$ to denote the reduction modulo x into the range $[-x/2, \ldots, x/2)$.

Input: Scalar k and window width w
Output: k in wNAF: $k_0, \ldots, k_{\ell-1}$
begin
  $\ell \leftarrow 0$
  while $k > 0$ **do**
    if $k \mod 2 = 1$ **then**
      $k_\ell \leftarrow |k|_{2^{w+1}}$
      $k \leftarrow k - k_\ell$
    else
      $k_\ell \leftarrow 0$
    end
    $k \leftarrow k/2$
    $\ell \leftarrow \ell + 1$
  end
end

Algorithm 2: Conversion to Non-Adjacent Form

Let $\overline{k_\ell}$ be the value of the variable k at the start of the ℓ^{th} iteration in Algorithm 2. From the algorithm, it is clear that

$$k_\ell = \begin{cases} 0 & \overline{k_\ell} \text{ is even} \\ |\overline{k_\ell}|_{2^{w+1}} & \overline{k_\ell} \text{ is odd} \end{cases} \tag{1}$$

Furthermore:

$$k = 2^\ell \cdot \overline{k_\ell} + \sum_{i<\ell} 2^i \cdot k_i \tag{2}$$

Let m and $m+l$ be the position of two consecutive non-zero digits of the wNAF representation of the ephemeral key k. That is, $k_m, k_{m+l} \neq 0$ and $k_{m+i} = 0$ for all $0 < i < l$. From (2) we obtain $k_m = \overline{k_m} - 2^l \cdot \overline{k_{m+l}}$.

One consequence of subtracting negative wNAF components is that the wNAF representation may be one digit longer than the binary representation of the number. For n-digits binary numbers Möller [17] suggests using $k_\ell \leftarrow \lfloor k \rfloor_{2^w}$ when $\ell = n - w - 1$ and k is odd, where $\lfloor \cdot \rfloor_x$ denotes the reduction modulo x into the interval $[0, \ldots, x)$. This avoids extending the wNAF representation in half the cases at the cost of weakening the non-adjacency property of the representation.

2.3 Lattice background

Before we describe how to get the necessary information from the side-channel attack, we recall from previous works what kind of information we are looking for. As in previous works [2, 4, 5, 12, 19, 20], the side-channel information is used to construct a lattice basis and the secret key is then retrieved by solving a lattice problem on this lattice. Generally, in previous works the authors somehow derive triples (t_i, u_i, z_i) from the side-channel information such that

$$v_i = |\alpha \cdot t_i - u_i|_q < q/2^{z_i+1}. \tag{3}$$

The use of a different z_i per equation was introduced in [2]. If we take d such triples we can construct the following lattice basis

$$B = \begin{pmatrix} 2^{z_1+1} \cdot q & & & \\ & \ddots & & \\ & & 2^{z_d+1} \cdot q & \\ 2^{z_1+1} \cdot t_1 & \ldots & 2^{z_d+1} \cdot t_d & 1 \end{pmatrix},$$

whose rows generate the lattice that we use to retrieve the secret key. Now consider the vector $\mathbf{u} = (2^{z_1+1} \cdot u_1, \ldots, 2^{z_d+1} \cdot u_d, 0)$, which consists of known quantities. Equation (3) implies the existence of integers $(\lambda_1, \ldots, \lambda_d)$ such that for the vectors $\mathbf{x} = (\lambda_1, \ldots, \lambda_d, \alpha)$ and $\mathbf{y} = (2^{z_1+1} \cdot v_1, \ldots, 2^{z_d+1} \cdot v_d, \alpha)$ we have

$$\mathbf{x} \cdot B - \mathbf{u} = \mathbf{y}.$$

Again using Equation (3), we see that the 2-norm of the vector $\mathbf{y}$ is at most $\sqrt{d \cdot q^2 + \alpha^2} \approx \sqrt{d+1} \cdot q$. Because the lattice determinant of $L(B)$ is $2^{d+\sum z_i} \cdot q^d$, the lattice vector $\mathbf{x} \cdot \mathbf{B}$ is heuristically the closest lattice vector to $\mathbf{u}$. By solving the Closest Vector Problem (CVP) on input of the basis B and the target vector $\mathbf{u}$, we obtain $\mathbf{x}$ and hence the secret key α.

There are two important methods of solving the closest vector problem: using an exact CVP-solver or using the heuristic embedding technique to convert it to a Shortest Vector Problem (SVP). Exact CVP-solvers require exponential time in the lattice rank ($d+1$ in our case), whereas the SVP instance that follows from the embedding technique can sometimes be solved using approximation methods that run in polynomial

time. Because the ranks of the lattices in this work become quite high when attacking a 521 bit key, we mostly focus on using the embedding technique and solving the associated SVP instance in this case.

The embedding technique transforms the previously described basis B and target vector $\mathbf{u}$ to a new basis B', resulting in a new lattice of dimension one higher than that generated by B:

$$B' = \begin{pmatrix} B & 0 \\ \mathbf{u}' & q \end{pmatrix},$$

where $\mathbf{u}' = (\mathbf{u}, 0)$. Following the same reasoning as above, we can set $\mathbf{x}' = (\mathbf{x}, \alpha, -1)$ and obtain the lattice vector $\mathbf{y}' = \mathbf{x}' \cdot B' = (\mathbf{y}, -q)$. The 2-norm of $\mathbf{y}'$ is upper bounded by approximately $\sqrt{d+2} \cdot q$, whereas this lattice has determinant $2^{d+\sum z_i} \cdot q^{(d+1)}$. Note, however, that this lattice also contains the vector$(-t_1, \ldots, -t_d, q, 0) \cdot B' = (0, \ldots, 0, q, 0)$, which will most likely be the shortest vector of the lattice. Still, our approximation algorithms for SVP work on bases and it is obvious to see that any basis of the same lattice must contain a vector ending in $\pm q$. Thus, it is heuristically likely that the resulting basis contains the short vector $\mathbf{y}'$, which reveals α.

To summarize, we turn the side-channel information into a lattice and claim that, heuristically, finding the secret key is equivalent to solving a CVP instance. Then, we claim that, again heuristically, solving this CVP instance is equivalent to solving an SVP instance using the embedding technique. In Section 6 we will apply the attack to simulated data to see whether these heuristics hold up.

3 Improving the Past Results

We first, as a warm up, improve the results on arbitrary sequences of consecutive bits presented in [20]. Our improvement is based on two assumptions:

1. The value q is close to a power of two. More precisely, $q = 2^n + \varepsilon$ where $|\varepsilon| < 2^p$ for some $p \approx n/2$. This assumption holds for many of the standard curves.
2. If a sequence of bits leaks, then the least significant bit in the leaked sequence is in position $m > p$.

The second assumption implies that we obtain some leakage information of the form

$$k = a \cdot 2^{m+l} + b \cdot 2^m + c$$

where $0 \leq a < 2^{n-m-l}, 0 \leq c < 2^m$ are unknown and $0 \leq b < 2^l$ is the leaked information for some known values of l and m. Rearranging the signing equation in the ECDSA algorithm for calculating s gives us

$$k = \alpha \cdot r \cdot s^{-1} + h \cdot s^{-1} \pmod{q} \tag{4}$$

Or, equivalently,

$$\left(2^{n-l-m} \cdot (k - b \cdot 2^m)\right) - 2^{n-l-1} = \left(\alpha \cdot r \cdot s^{-1} \cdot 2^{n-l-m}\right) + \left((h \cdot s^{-1} - b \cdot 2^m) \cdot 2^{n-l-m}\right) - 2^{n-l-1} \pmod{q} \tag{5}$$

We now define the values

$$\begin{aligned} t &= \lfloor r \cdot s^{-1} \cdot 2^{n-l-m} \rceil_q \\ u &= \lfloor 2^{n-l-1} - (h \cdot s^{-1} - b \cdot 2^m) \cdot 2^{n-l-m} \rceil_q \\ v &= |\alpha \cdot t - u|_q \end{aligned}$$

By equation (5) we then obtain,

$$\begin{aligned}|v| &= \big|\ |2^{n-l-m}\cdot(k-b2^m)-2^{n-l-1}|_q\big| = \big|\ |2^{n-l-m}\cdot(a\cdot 2^{m+l}+c)-2^{n-l-1}|_q\big| \\ &= \big|\ |a\cdot 2^n + c\cdot 2^{n-l-m} - 2^{n-l-1}|_q\big| = \big|\ |a\cdot q - a\cdot\varepsilon + c\cdot 2^{n-l-m} - 2^{n-l-1}|_q\big| \\ &= \big|\ |(c-2^{m-1})\cdot 2^{n-l-m} - a\cdot\varepsilon|_q\big| \\ &\leq \big|\ |(c-2^{m-1})\cdot 2^{n-l-m}|_q\big| + \big||\ a\cdot\varepsilon|_q\big| \leq 2^{n-l+1} + 2^{n-l-m+p} \leq 2^{n-l+2} \approx q/2^{l-2}.\end{aligned}$$

Thus $(t,u,l-3)$ is one of our triples described in Subsection 2.3. Furthermore, if $2^p < c < 2^m - 2^p$, as is the case when leaking steps in the wNAF algorithm, we find $v \leq 2^{n-l-1} \approx q/2^{l-1}$, thus in this case we obtain the slightly better triple $(t,u,l-2)$. The rest follows using the same method of Nguyen and Shparlinski [19]. These results demonstrate a potential problem with curves whose group order is close to a power of two, making them more vulnerable than other curves to partially exposed ephemeral keys.

Recall that when given an arbitrary sequence of l consecutive leaked bits Nguyen and Shparlinski [19, Section 5.1] obtain a similar quantity v for DSA where

$$|v| \leq q/2^{l/2-1},$$

which they extend to ECDSA [20]. Note that this is for the general case, i.e., q can be of any form and the leaked bits can be in any position. Thus, under our two assumptions we get a factor two improvement on the number of leaked bits.

Having an order close to a power of two also negates the protection provided by scalar blinding [4, 16]. The method suggested by these works is to compute $(k+h\cdot q+\bar{h})G - \bar{h}G$ where h and $\bar{h}$ are small (e.g. 32-bit) random numbers. As $|\varepsilon|$ is significantly smaller than q and as $(k+h\cdot q+\bar{h}) = h2^n + (k+h\cdot\varepsilon+\bar{h})$, the $\lfloor\log(h)\rfloor$ MSB bits of $(k+hq+\bar{h})$ are exactly h for almost any value of k, allowing an attacker to remove a significant part of the blinding. Furthermore, even if the leak does not expose h, all the bits in positions $p+\lfloor\log(h)\rfloor$ to $n-1$ are the same in both k and in $(k+h\cdot q+\bar{h})$. Consequently, a leak of these bits in the blinded form also implies a leak of bits in the unblinded form.

Brumley and Hakala [4] also suggest that the requirement of $\bar{h}$ being a small number can be relaxed and that a similar performance can be achieved if $\bar{h}$ has a low Hamming weight. This form of blinding masks some of the high bits of k; However, for a low Hamming weight, the number of masked bits will be low, and a leak of bits in the blinded form still translates to a leak in the unblinded form.

We next show how to combine this analysis with the information obtained from a FLUSH+RELOAD attack against the wNAF algorithm.

4 Using the wNAF Information

In this section we extend the the technique described above to the double and add chains extracted from the wNAF multiplication as implemented in OpenSSL. The main issue addressed in this section is that while the double and add chain for the wNAF algorithm identifies groups of repeating bits, it does not provide information on the value of these bits. Thus, the technique in Section 3 does not apply directly to the wNAF double and add chains.

In this section, we assume a perfect side-channel, i.e. a side-channel which can produce the exact double and add chain used by a point multiplication algorithm without any errors. We discuss the practical issues of handling real-life side-channels in Section 7.

As described in [2], the OpenSSL implementation departs slightly from the descriptions of ECDSA in Section 1. As a countermeasure to the Brumley and Tuveri remote timing attack [5], OpenSSL adds q or $2\cdot q$

to the randomly chosen ephemeral key, ensuring that k is $n+1$ bits long. While the attack is only applicable to curves defined over binary fields, the countermeasure is applied to all curves. Consequently, our analysis assumes that $2^n \leq k < 2^{n+1}$.

Given a double and add chain, the positions of the add operations in the chain correspond to the non-zero digits in the wNAF representation of the ephemeral key k. Let m and $m+l$ be the position of two consecutive add operations in the chain such that $p-w+2 < m < n-l$. Set the following values:

$$\begin{aligned}
a &= \frac{\overline{k_{m+l}} - 1}{2} \\
c &= \sum_{i<m} 2^i \cdot k_i + 2^m \cdot \lfloor \overline{k_m} \rfloor_{2^w} = k - 2^m \cdot \overline{k_m} + 2^m \cdot \lfloor \overline{k_m} \rfloor_{2^w} \\
b &= \frac{k - a \cdot 2^{m+l+1} - c}{2^{m+w}} = \frac{2^m \cdot \overline{k_m} - 2^{m+l} \cdot \overline{k_{m+l}} + 2^{m+l} - 2^m \cdot \lfloor \overline{k_m} \rfloor_{2^w}}{2^{m+w}} \\
&= \frac{2^{m+l} + 2^m \cdot k_m - 2^m \cdot \lfloor \overline{k_m} \rfloor_{2^w}}{2^{m+w}} = 2^{l-w} + (k_m - \lfloor \overline{k_m} \rfloor_{2^w}) \cdot 2^{-w}
\end{aligned}$$

By equation (1), $k_m = |\overline{k_m}|_{2^{w+1}}$, hence, $k_m - \lfloor \overline{k_m} \rfloor_{2^w}$ is either 0 or -2^w. Thus, the value of b is either 2^{l-w} or $2^{l-w} - 1$ and, consequently, k can be written as $k = a \cdot 2^{m+l+1} + b \cdot 2^{m+w} + c$, where $0 \leq c < 2^{m+w}$, $0 \leq a < 2^{n-m-l}$ and b is either 2^{l-w} or $2^{l-w} - 1$. Alternatively,

$$k + 2^{m+w} = a \cdot 2^{m+l+1} + 2^{m+l} + c' \tag{6}$$

where $0 \leq c' < 2^{m+w+1}$. On the other hand, by equation (4),

$$\begin{aligned}
2^{n-m-l-1} \cdot (k + 2^{m+w} - 2^{m+l}) - 2^{n+w-l-1} &= \alpha \cdot r \cdot s^{-1} \cdot 2^{n-m-l-1} \\
&\quad + (h \cdot s^{-1} + 2^{m+w} - 2^{m+l}) \cdot 2^{n-m-l-1} - 2^{n+w-l-1}
\end{aligned} \tag{7}$$

Following the method in Section 3, we now define the values:

$$\begin{aligned}
t &= \lfloor r \cdot s^{-1} \cdot 2^{n-m-l-1} \rfloor_q, \\
u &= \lfloor 2^{n+w-l-1} - (h \cdot s^{-1} + 2^{m+w} - 2^{m+l}) \cdot 2^{n-m-l-1} \rfloor_q, \\
v &= |\alpha \cdot t - u|_q.
\end{aligned}$$

Using the same method as above, for $m > p-w+2$ by equation (4) we obtain,

$$\begin{aligned}
|v| &= \big|\, |2^{n-m-l-1}(k + 2^{m+w} - 2^{m+l}) - 2^{n+w-l-1}|_q \big| = \big|\, |2^{n-l-m-1}(a2^{m+l+1} + c') - 2^{n+w-l-1}|_q \big| \\
&= \big|\, |a2^n + c'2^{n-l-m-1} - 2^{n+w-l-1}|_q \big| = \big|\, |aq - a\varepsilon + c'2^{n-l-m-1} - 2^{n+w-l-1}|_q \big| \\
&= \big|\, |(c' - 2^{m+w})2^{n-m-l-1} - a\varepsilon|_q \big| \\
&\leq \big|\, |(c' - 2^{m+w})2^{n-m-l-1}|_q \big| + \big|\, |a\varepsilon|_q \big| \leq 2^{n-l+w-1} + 2^{n-m-l+p} \leq 2^{n-l+w} \approx q/2^{l-w}
\end{aligned}$$

Note that if $m > p+1$, then the ephemeral key bits at position m and $m-1$ are different, and consequently, $2^{p+1} < c' < 2^{m+w+1} - 2^{p+1}$ and, therefore, $0 < c' + 2 \cdot \varepsilon < 2^{m+w+1}$. Hence, if $m > p+1$ we obtain the slightly stronger inequality $|v| \leq q/2^{l-w+1}$. In terms of the lattice attack outlined in Subsection 2.3, we have that $|v| \leq q/2^{z+1}$ for z equal to $l-w-1$ or $l-w$. This means we have a triple (t,u,z) that we can use to construct the lattice.

5 Heuristic Analysis

Now we know how to derive our triples t_i, u_i and z_i that are used to construct the lattice. The next obvious question is: How many do we need before we can retrieve the private key α? Because the lattice attack relies on several heuristics, it is hard to give a definitive analysis. However, we will give heuristic reasons here, similar to those for past results.

Each triple (t_i, u_i, z_i) gives us z_i bits of information. If this triple comes from a pair (m, l) such that $p+1 < m < n-l$, then $z_i = l - w$. Roughly speaking, we have that in half of the cases $l = w+1$, in a quarter of the cases $l = w+2$ and so on. Thus, on average we have that $l = w + \sum_i i/2^i = w+2$. On average we lose $(w+2)/2$ bits before the first usable triple and $(w+2)/2$ after the last usable triple, which leaves us with $n-1-(p+2)-(w+2)$ bits where our triples can be. The average number of triples is now given by $(n-p-3-(w+2))/(w+2)$ and each of these triples gives us $l-w=2$ bits on average. Combining this yields $2 \cdot (n-p-3-(w+2))/(w+2) = 2 \cdot (n-p-3)/(w+2) - 2$ bits per signature. For the **secp256k1** curve we have that $n = 256$, $p = 129$ and $w = 3$, leading to 47.6 bits per signature on average. Our data obtained from perfect side-channels associated to 1001 signatures gives us an average of 47.6 with a 95% confidence interval of ± 0.2664. For the **secp521r1** curve, we have that $n = 521$, $p = 259$ and $w = 4$, which suggests 84.33 bits per signature on average. The data average here is 84.1658 with a 95% confidence interval of ± 0.3825. See also the $Z = 1$ cases of Figures 1 and 2, which show the distribution of the bits leaked per signature in the 256-bit and 521-bit cases, respectively.

This formula suggests that on average, six signatures would be enough to break a 256-bit key (assuming a perfect side channel), since $47.6 \cdot 6 = 285.6 > 256$. However, in our preliminary experiments the attack did not succeed once when using six or even seven signatures. Even eight or nine signatures gave a minimal success probability. This indicates that something is wrong with the heuristic. In general there are two possible reasons for failure. Either the lattice problem has the correct solution but it was too hard to solve, or the solution to the lattice problem does not correspond to the private key α. We will now examine these two possibilities and how to deal with them.

5.1 Hardness of the lattice problem

Generally, the lattice problem becomes easier when adding more information to the lattice, but it also becomes harder as the rank increases. Since each triple adds information but also increases the rank of the lattice, it is not always clear whether adding more triples will solve the problem or make it worse. Each triple contributes z_i bits of information, so we would always prefer triples with a higher z_i value. Therefore, we set a bound $Z \geq 1$ and only keep those triples that have $z_i \geq Z$. However, this decreases the total number of bits of information we obtain per signature. If Z is small enough, then roughly speaking we only keep a fraction 2^{1-Z} of the triples, but now each triple contributes $Z+1$ bits on average. Hence, the new formula of bits per signature becomes

$$2^{1-Z} \cdot (Z+1) \cdot ((n-p-3)/(w+2) - 1).$$

Our data reflects this formula as well as can be seen in Figures 1 and 2 for the 256-bit and the 521-bit cases, respectively. In our experiments we will set an additional bound d on the number of triples we use in total, which limits the lattice rank to $d+1$. To this end, we sort the triples by z_i and then pick the first d triples to construct the lattice. We adopt this approach for our experiments and the results can be found in Section 6.

5.2 Incorrect solutions

The analysis of Nguyen and Shparlinski [20] requires that the t_i values in the triples are taken uniformly and independently from a distribution that satisfies some conditions. However, it is easy to see that when two triples are taken from the same signature, the values for the $t_i = \lfloor r \cdot s^{-1} \cdot 2^{n-m_i-l_i-1} \rfloor_q$ and $t_j = \lfloor r \cdot s^{-1} \cdot 2^{n-m_j-l_j-1} \rfloor_q$ are not even independent, as they differ $\bmod q$ by a factor that is a power of 2 less than 2^n.

Recall from Sections 2.3 and 4 how the triples are used and created, respectively. Consider a triple (t_{ij}, u_{ij}, z_{ij}) corresponding to a signature (r_i, s_i, h_i). The corresponding $v_{ij} = |\alpha \cdot t_{ij} - u_{ij}|_q$ satisfies

$$|v_{ij}| = \left||\alpha \cdot (r_i \cdot s_i^{-1} \cdot 2^{n-m_j-l_j-1}) - 2^{n+w-l_j-1} + (h_i \cdot s_i^{-1} + 2^{m_j+w} - 2^{m_j+l}) \cdot 2^{n-m_j-l_j-1}|_q\right| \leq q/2^{z_{ij}+1},$$

which is equivalent to

$$|v_{ij}| = \left||(\alpha \cdot r_i + h_i) \cdot s_i^{-1} \cdot 2^{n-m_j-l_j-1} - 2^{n-1}|_q\right| \leq q/2^{z_{ij}+1},$$

where $p+1 < m_j < n - l_j$ and $z_{ij} = l - w$. Now $(\alpha \cdot r_i + h_i) \cdot s_i^{-1} = k_i \mod q$ and we know that the previous statement holds due to the structure of k_i, specifically due to its bits $m_j + w, \ldots, m_j + l_j - 1$ repeating, with bit $m_j + l_j$ being different than the preceding bit. But the map $x \mapsto (x \cdot r_i + h_i) \cdot s_i^{-1}$ is a bijection $\bmod q$, and hence for each i there will be many numbers X such that for all j

$$|v_{ij}(X)| = \left||(X \cdot r_i + h_i) \cdot s_i^{-1} \cdot 2^{n-m_j-l_j-1} - 2^{n-1}|_q\right| \leq q/2^{z_{ij}+1}.$$

Let $S_i = \{X : |v_{ij}(X)| \leq q/2^{z_{ij}+1} \text{ for all } j\}$. If we now have that there exists an $X \in \bigcap_i S_i$ such that

$$X^2 + \sum_{i,j} (2^{z_{ij}} \cdot v_{ij}(X))^2 < \alpha^2 + \sum_{i,j} (2^{z_{ij}} \cdot v_{ij}(\alpha))^2,$$

then it is very unlikely that the lattice algorithm will find α, because X corresponds to a better solution to the lattice problem. Note that this problem arises when fewer signatures are used, because this leads to fewer distinct values for (r_i, s_i, h_i) and hence fewer sets S_i that need to intersect. This suggests that increasing the number of signatures could increase the success probability.

Assuming that the S_i are random, we want to determine what is the probability that their intersection is non-empty. First we consider the size of the S_i. Recall that S_i consists of all $X \mod q$ such that $v_{ij}(X)$ has 'the same structure as k_i'. This means that for each triple specified by m_j and l_j, the bits $m_j + w, \ldots, m_j + l_j - 1$ repeat, and bit $m_j + l_j$ is the opposite of the preceding bits. There are approximately $2^{n-(l_j-w+1)+1}$ numbers $\bmod q$ that have this structure. Let f_i be the number of triples of signature i and $g_{ij} = (l_j - w + 1)$ be the number of bits fixed by triple j of signature i. Then, because the triples do not overlap and because $v_{ij}(.)$ is a bijection, we have that

$$\log_2(|S_i|) = n - \sum_{j=1}^{f_i} (1 - g_{ij}) = n - f_i + \sum_{j=1}^{f_i} g_{ij}.$$

Let $s_i = |S_i|$ and assume that the S_i are chosen randomly and independently from all the subsets of integers in the range $[0, \ldots, N-1]$ (of size s_i), where $N = 2^n$. Consider the following probability

$$p_i = \mathbb{P}(0 \in S_i) = s_i/N,$$

since S_i is randomly chosen. Now, because the S_i are also chosen independently, we have

$$\mathbb{P}\left(0 \in \bigcap_i S_i\right) = \prod_i p_i.$$

Finally, since this argument holds for any $j \in [0,\ldots,N-1]$, we can apply the union bound to obtain

$$p_{\text{fail}} = \mathbb{P}\left(\bigcup_j \left(j \in \bigcap_i S_i\right)\right) \leq \sum_j \mathbb{P}\left(0 \in \bigcap_i S_i\right) = N \cdot \prod_i p_i. \tag{8}$$

Recall that each signature has $f_i = 2^{1-Z} \cdot ((n-p-3)/(w+2)-1)$ triples on average and each triple contributes $Z+1$ bits on average, which means $g_{ij} = Z+2$ on average. If we plug in the numbers $n = 256$, $p = 129$, $w = 3$ and $Z = 3$, we get that $f_i \approx 6$, $g_{ij} = 5$ and hence $p_i \approx 2^{-6\cdot(5-1)} \approx 2^{-24}$ if we assume an average number of triples and bits in each signature. This in turn gives us an upper bound of $p_{\text{fail}} \leq N/2^{24\cdot k}$. If $k \geq 11$, this upper bound is less than one, so this clearly suggests that from about eleven signatures and up, we should succeed with some probability, which is indeed the case from our experiments.

Repeating this for $n = 521$, $p = 259$, $w = 4$ and $Z = 4$, we obtain $f_i \approx 5$, $g_{ij} = 6$ and hence $p_i \approx 2^{-5\cdot(6-1)} \approx 2^{-25}$. Consequently, $p_{\text{fail}} \leq N/2^{25\cdot k}$, which is less than one when $k \geq 21$. However, in our experiments we require at least 30 signatures to obtain the secret key with some probability. Thus the above analysis is only approximate as the secret key length increases.

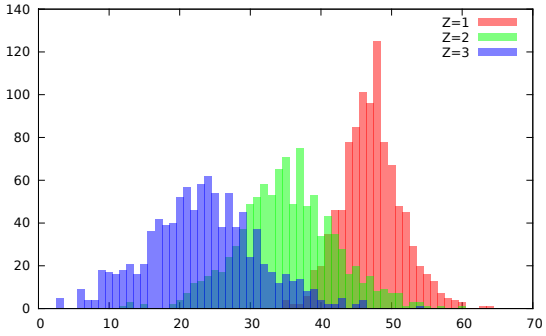

Fig. 1: Number of signatures against bits per signature in the 256 bit case.

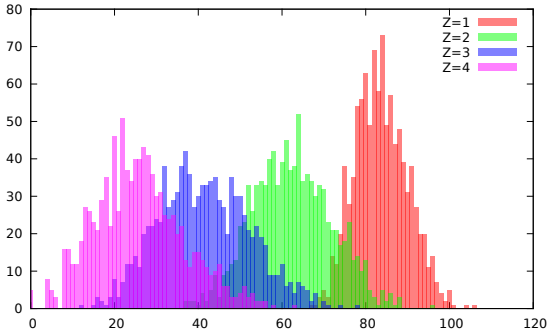

Fig. 2: Number of signatures against bits per signature in the 521 bit case.

6 Results With a Perfect Side-Channel

Subsection 2.3 outlined our (heuristic) approach to obtain the secret key from a number of triples (t_i, u_i, z_i) using lattices and Section 4 outlined how to generate these triples from the side-channel information. In this section we will look at some experimental results to see if our heuristic assumptions are justified.

As per Section 5, we used the following approach for our experiments. First, we fix a number of signatures s, a lattice rank d and a bound Z. We then take s signatures at random from our data set and derive all triples such that $z_i \geq Z$, sorting them such that the z_i are in descending order. If we have more than d triples, we only take the first d to construct the lattice. Finally we attempt to solve the lattice problem and note the result. All executions were performed in single thread on an Intel Core i7-3770S CPU running at 3.10 GHz.

When solving the CVP instances there are three possible outcomes. We obtain either no solution, the private key or a wrong solution. No solution means that the lattice problem was too hard for the algorithm and constraints we used, but spending more time and using stronger algorithms might still solve it. When a

'wrong' solution is obtained, this means that our heuristics failed: the solution vector was not unique, in the sense that there were other lattice vectors within the expected distance from our target vector.

When solving the SVP instance there are only two outcomes. Either we obtain the private key or not. However, in this case it is not as clear whether a wrong solution means that there were other solutions due to the additional heuristics involved. Full details of our experimental data is given in the Appendix.

6.1 256 bit key

For the 256 bit case, we used BKZ with block size 20 from fplll [6] to solve the SVP instances, as well as to pre-process the CVP instances. To solve the CVP, we applied Schnorr-Euchner enumeration [21] using linear pruning [10] and limiting the number of enumerated nodes to 2^{29}.

The CVP approach seems the best, as the lattice rank $(d+1)$ remains quite small. We restrict our triples to $Z = 3$ to keep the rank small, but a smaller Z would not improve our results much. See the appendix for details. We observed that failures are mostly caused by 'wrong' solutions in this case, rather than the lattice problem being too hard. In all cases we found that using 75 triples gave the best results. Table 1 lists the runtimes and success probabilities of the lattice part of the attack for varying s. The results are graphically presented in Figures 3 and 4.

s	Time (s)	p_{succ} (%)
10	2.25	7.0
11	4.66	25.0
12	7.68	38.5
13	11.30	54.0

Table 1: CVP results for 75 triples taken from s signatures with a 256-bit key ($Z = 3$)

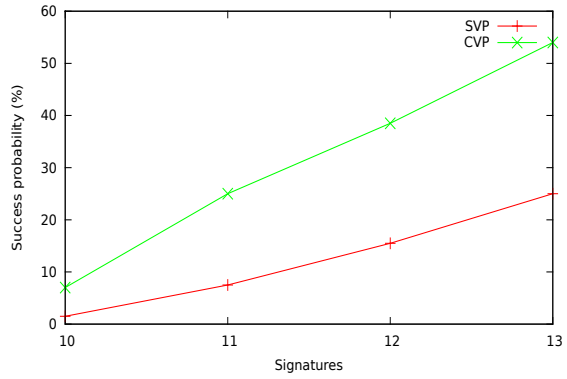

Fig. 3: Success probability per number of signatures against a 256 bit key

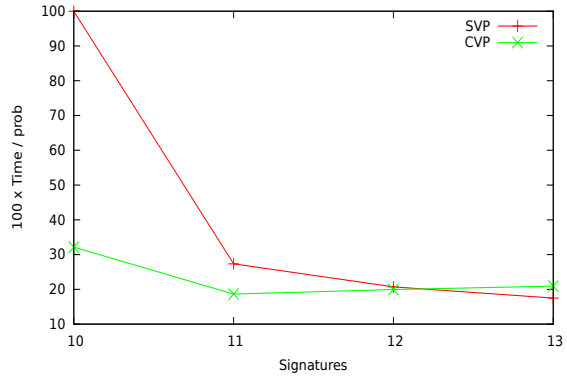

Fig. 4: Expected running time per number of signatures against a 256 bit key

6.2 521 bit key

For the 521 bit case, we used BKZ with block size 20 from fplll [6] to solve the SVP instances. Due to the higher lattice ranks in this case, solving the CVP instances proved much less efficient, even when restricting the triples to $Z = 4$.

With 30 signatures we get a small probability of success in the lattice attack whereas with 40 signatures we can obtain the secret key in more than half of the cases. It should be noted that as the number of signatures increases, the choice of d becomes less important, because the number of triples with more information increases. See Table 2 for details and Figures 5 and 6 for a graphical representation.

s	d	Time (s)	p_{succ} (%)
30	130	50.10	4.0
31	130	48.50	7.5
32	150	70.77	9.5
33	150	70.54	13.5
34	140	62.83	16.0
35	135	55.33	24.5
36	145	62.32	29.0
37	155	69.14	34.5
38	145	61.31	42.5
39	145	57.08	47.5
40	130	47.73	53.0

Table 2: SVP results for d triples taken from s signatures with a 521-bit key ($Z = 4$)

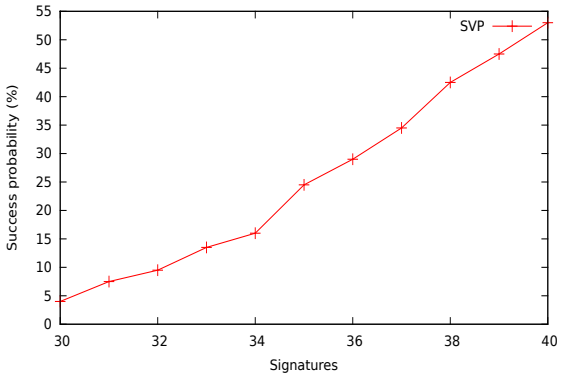

Fig. 5: Success probability per number of signatures against a 521 bit key

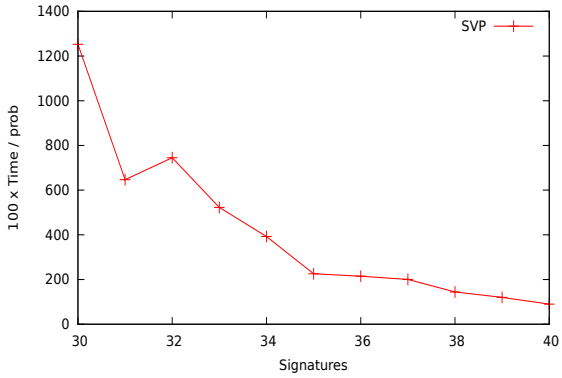

Fig. 6: Expected running time per number of signatures against a 521 bit key

7 Results in a Real-Life Attack

So far our discussion was based on the assumption of a perfect side-channel. That is, we assumed that the double-and-add chains are recovered without any errors. Perfect side-channels are, however, very rare. In this section we extend the results to the actual side-channel exposed by the FLUSH+RELOAD technique.

The attack was carried on an HP Elite 8300, running CentOS 6.5. The victim process runs OpenSSL 1.0.1f, compiled to include debugging symbols. These symbols are not used at run-time and do not affect the performance of OpenSSL. We use them because they assist us in finding the addresses to probe by avoiding reverse engineering [8].

The spy uses a time slot of 1,200 cycles (0.375μs). In each time slot it probes the memory lines containing the last field multiplication within the group add and double functions. (`ec_GFp_simple_add` and

ec_GFp_simple_dbl, respectively.) Memory lines that contain function calls are accessed both before and after the call, reducing the chance of a spy missing the access due to overlap with the probe. Monitoring code close to the end of the function eliminates false positives due to speculative execution. See Yarom and Falkner [25] for a discussion of overlaps and speculative execution.

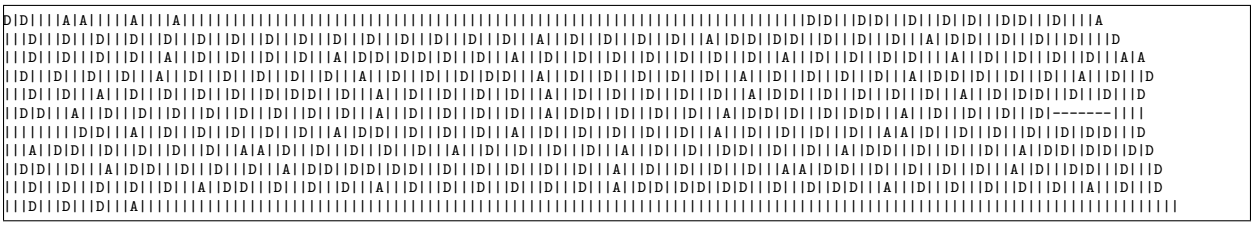

Fig. 7: FLUSH+RELOAD spy output. Vertical bars indicate time-slot boundaries; 'A' and 'D' are probes for OpenSSL access to add and double; dashes indicate missed time-slots.

Figure 7 shows an example of the output of the spy when OpenSSL signs using **secp256k1**. The double and three addition operations at the beginning of the captured sequence are the calculation of the precomputed wNAF digits. Note the repeated capture of the double and add operations due to monitoring a memory line that contains a function call. The actual wNAF multiplication starts closer to the end of the line, with 7 double operations followed by a group addition.

In this example, the attack captures most of the double and add chain. It does, however, miss a few time-slots and consequently a few group operations in the chain. The spy recognises missed time-slots by noting inexplicable gaps in the processor cycle counter. As we do not know which operations are missed, we lose the bit positions of the operations that precede the missed time-slots. We believe that the missed time-slots are due to system activity which suspends the spy.

Occasionally OpenSSL suspends the calculation of the scalar multiplication to perform memory management functions. These suspends confuse our spy program, which assumes that the scalar multiplication terminated. This, in turn, results in a short capture, which cannot be used for the lattice attack.

To test prevalence of capture errors we captured 1,000 scalar multiplications and compared the capture results to the ground truth. 342 of these captures contained missed time-slots. Another 77 captures contains less than 250 group operations and are, therefore, too short. Of the remaining 581 captures, 577 are perfect while only four contain errors that we could not easily filter out.

Recall, from Section 6, that 13 perfectly captured signatures are sufficient for breaking the key of a 256 bits curve with over 50% probability. An attacker using FLUSH+RELOAD to capture 25 signatures can thus expect to capture 14 that contain no obvious errors. With less than 1% probability that each of these 14 captures contains an error, the probability that more than one of these captures contains an error is also less than 1%. Hence, the attacker only needs to test all the combination of choosing 13 captures out of these 14 to achieve a 50% probability of breaking the signing key.

Several optimisations can be used to improve the figure of 25 signatures. Some missed slots can be recovered and the spy can be improved to correct short captures. Nevertheless, it should be noted that this figure is still an order of magnitude than the previously best known result of 200 signatures [2].

Acknowledgements

The authors would like to thank Ben Sach for helpful conversations during the course of this work. The first and second authors work has been supported in part by ERC Advanced Grant ERC-2010-AdG-267188-CRIPTO, by EPSRC via grant EP/I03126X, and by Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under agreement number FA8750-11-2-0079[1].

The third author wishes to thank Dr Katrina Falkner for her advice and support and the Defence Science and Technology Organisation (DSTO) Maritime Division, Australia, who partially funded his work.

References

1. American National Standards Institute. *ANSI X9.62, Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm*, 1999.
2. Naomi Benger, Joop van de Pol, Nigel P. Smart, and Yuval Yarom. "Ooh aah. . . , just a little bit": A small amount of side channel can go a long way. Cryptology ePrint Archive, Report 2014/161, March 2014. `http://eprint.iacr.org/`.
3. Dan Boneh and Ramarathnam Venkatesan. Hardness of computing the most significant bits of secret keys in diffie-hellman and related schemes. In *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 129–142, 1996.
4. Billy Bob Brumley and Risto M. Hakala. Cache-timing template attacks. In Mitsuru Matsui, editor, *Advances in Cryptology - ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 667–684. Springer-Verlag, 2009.
5. Billy Bob Brumley and Nicola Tuveri. Remote timing attacks are still practical. In Vijay Atluri and Claudia Diaz, editors, *Computer Security - ESORICS 2011*, pages 355–371, Leuven, Belgium, September 2011.
6. David Cadé, Xavier Pujol, and Damien Stehlé. FPLLL-4.0.4. `http://perso.ens-lyon.fr/damien.stehle/fplll/`, 2013.
7. Certicom Research. *SEC 2: Recommended Elliptic Curve Domain Parameters, Version 2.0*, January 2010.
8. Teodoro Cipresso and Mark Stamp. Software reverse engineering. In Peter Stavroulakis and Mark Stamp, editors, *Handbook of Information and Communication Security*, chapter 31, pages 659–696. Springer, 2010.
9. Richard E. Crandall. Method and apparatus for public key exchange in a cryptographic system. US Patent 5,159,632, October 1992.
10. Nicolas Gama, Phong Q. Nguyen, and Oded Regev. Lattice enumeration using extreme pruning. In *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 257–278, 2010.
11. Daniel M. Gordon. A survey of fast exponentiation methods. *Journal of Algorithms*, 27(1):129–146, April 1998.
12. Nick Howgrave-Graham and Nigel P. Smart. Lattice attacks on digital signature schemes. *Designs, Codes and Cryptography*, 23(3):283–290, 2001.
13. Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. kvm: the Linux virtual machine monitor. In *Proceedings of the Linux Symposium*, volume one, pages 225–230, Ottawa, Ontario, Canada, June 2007.
14. Arjen K. Lenstra, Hendrik W. Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515 –534, 1982.
15. Lixin Li, James E. Just, and R. Sekar. Address-space randomization for Windows systems. In *Proceedings of the 22nd Annual Computer Security Applications Conference*, pages 329–338, Miami Beach, Florida, United States, December 2006.
16. Bodo Möller. Parallelizable elliptic curve point multiplication method with resistance against side-channel attacks. In Agnes Hui Chan and Virgil D. Gligor, editors, *Proceedings of the fifth International Conference on Information Security*, number 2433 in Lecture Notes in Computer Science, pages 402–413, São Paulo, Brazil, September 2002.
17. Bodo Möller. Improved techniques for fast exponentiation. In P. J. Lee and C. H. Lim, editors, *Information Security and Cryptology - ICISC 2002*, number 2587 in Lecture Notes in Computer Science, pages 298–312. Springer-Verlag, 2003.
18. National Institute of Standards and Technology. *FIPS PUB 186-4 Digital Signature Standard (DSS)*, 2013.
19. Phong Q. Nguyen and Igor E. Shparlinski. The insecurity of the digital signature algorithm with partially known nonces. *Journal of Cryptology*, 15(3):151–176, June 2002.
20. Phong Q. Nguyen and Igor E. Shparlinski. The insecurity of the elliptic curve digital signature algorithm with partially known nonces. *Designs, Codes and Cryptography*, 30(2):201–217, September 2003.

[1] The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Defense Advanced Research Projects Agency (DARPA) or the U.S. Government.

21. Claus-Peter Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. In *Fundamentals of Computation Theory – FCT 1991*, volume 529 of *Lecture Notes in Computer Science*, pages 68–85. Springer, 1991.
22. Jerome A. Solinas. Generalized Mersenne numbers. Technical Report CORR-39, University of Waterloo, 1999.
23. Carl A. Waldspurger. Memory resource management in VMware ESX Server. In David E. Culler and Peter Druschel, editors, *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation*, pages 181–194, Boston, Massachusetts, United States, December 2002.
24. Yuval Yarom and Naomi Benger. Recovering OpenSSL ECDSA nonces using the FLUSH+RELOAD cache side-channel attack. Cryptology ePrint Archive, Report 2014/140, February 2014. `http://eprint.iacr.org/`.
25. Yuval Yarom and Katrina Falkner. FLUSH+RELOAD: a high resolution, low noise, L3 cache side-channel attack. In *To appear: USENIX 2014*, 2014.

A Experimental results

		SVP		CVP	
s	d	Time (s)	p_{succ} (%)	Time (s)	p_{succ} (%)
10	60	1.47	0.0	1.56	0.5
10	65	1.42	1.0	1.90	2.5
10	70	1.44	1.5	2.45	4.0
10	75	1.50	1.5	2.25	7.0
11	60	1.28	0.0	1.63	0.5
11	65	1.68	5.0	2.35	6.5
11	70	1.86	2.5	3.15	19.0
11	75	2.05	7.5	4.66	25.0
11	80	2.12	6.0		
12	60	1.27	2.0	1.69	7.0
12	65	1.71	2.5	2.45	10.5
12	70	2.20	7.5	3.99	29.5
12	75	2.57	10.5	7.68	38.5
12	80	2.90	13.0		
12	85	3.12	8.5		
12	90	3.21	15.5		
13	60	1.30	3.5	1.92	8.5
13	65	1.77	6.0	2.79	25.5
13	70	2.39	11.0	4.48	46.5
13	75	3.16	19.0	11.30	54.0
13	80	3.67	18.5		
13	85	3.81	21.5		
13	90	4.37	25.0		

Table 3: Results for d triples taken from s signatures with a 256-bit key ($Z = 3$)

s	d	Time (s)	p_{succ} (%)
30	130	50.10	4.0
30	135	58.80	3.0
30	140	66.65	3.5
30	145	69.68	2.5
32	130	50.15	6.5
32	135	58.07	6.5
32	140	62.55	4.0
32	145	67.46	5.0
32	150	70.77	9.5
34	130	50.00	15.5
34	135	55.93	10.5
34	140	62.83	16.0
34	145	64.41	14.0
34	150	70.50	16.0
34	155	71.07	11.5
36	130	48.71	24.5
36	135	54.74	21.0
36	140	59.25	22.5
36	145	62.32	29.0
36	150	65.60	29.0
36	155	68.57	24.5
38	130	49.04	38.5
38	135	53.86	36.0
38	140	57.14	38.5
38	145	61.31	42.5
38	150	66.75	36.5
38	155	66.52	36.5
40	130	47.73	53.0
40	135	50.80	49.0
40	140	54.88	52.0
40	145	60.47	47.0
40	150	64.77	53.0
40	155	64.95	52.5

s	d	Time (s)	p_{succ} (%)
31	130	48.50	7.5
31	135	59.91	3.5
31	140	67.35	6.0
31	145	69.96	5.5
33	130	49.70	8.0
33	135	56.52	11.5
33	140	60.31	11.5
33	145	66.39	8.5
33	150	70.54	13.5
33	155	75.49	8.5
35	130	49.76	12.0
35	135	55.33	24.5
35	140	59.50	15.5
35	145	65.59	19.5
35	150	66.93	24.0
35	155	69.67	20.0
37	130	48.20	24.0
37	135	54.79	23.5
37	140	58.60	28.0
37	145	60.05	29.0
37	150	63.40	27.5
37	155	69.14	34.5
39	135	50.99	45.5
39	140	58.81	46.0
39	145	57.08	47.5
39	150	62.35	41.5
39	155	64.99	42.5

Table 4: SVP results for d triples taken from s signatures with a 521-bit key ($Z = 4$)