Journal of Cryptographic Engineering (2014) 4:33–45
DOI 10.1007/s13389-014-0072-z

Using Bleichenbacher's Solution to the Hidden Number Problem to Attack Nonce Leaks in 384-Bit ECDSA: Extended Version

Elke De Mulder · Michael Hutter · Mark E. Marson · Peter Pearson

Received: 31 October 2013 / Accepted: 1 January 2014 / Published online: 2 February 2014
The final publication is available at Springer via http://dx.doi.org/10.1007/s13389-014-0072-z

Abstract In this paper we describe an attack against nonce leaks in 384-bit ECDSA using an FFT-based attack due to Bleichenbacher. The signatures were computed by a modern smart card. We extracted the low-order bits of each nonce using a template-based power analysis attack against the modular inversion of the nonce. We also developed a BKZ-based method for the range reduction phase of the attack, as it was impractical to collect enough signatures for the collision searches originally used by Bleichenbacher. We confirmed our attack by extracting the entire signing key using a 5-bit nonce leak from 4 000 signatures.

Keywords Side Channel Analysis · ECDSA · Modular Inversion · Hidden Number Problem · Bleichenbacher · FFT · LLL · BKZ.

1 Introduction

In this paper we describe an attack against nonce leaks in 384-bit ECDSA [2] running on a modern smart card. The attack has several interesting and novel features. We first identified a leak during the modular inversion of the nonce, and used differential power analysis (DPA) [17] to identify the likely inversion algorithm. Although the algorithm contains conditional branches, they were not exploitable by simple power analysis (SPA). Instead, we extracted a few low-order bits of each nonce using a template-based power analysis attack [8].

Such nonce leaks are commonly attacked by mapping them to a hidden number problem (HNP), and using lattice methods such as LLL [18], BKZ [26], and Babai's nearest plane algorithm [3] to solve the resulting closest vector problem (CVP) or shortest vector problem (SVP). While it might have been possible to use lattice attacks successfully, our initial template attacks only recovered very few bits reliably. We therefore chose to explore Bleichenbacher's approach [4], which given enough signatures can work with small, even fractional, bit leaks. In contrast, current lattice methods require a minimum number of bits to leak, regardless of the number of signatures used.

Bleichenbacher introduced his FFT-based attack in 2000 during an IEEE P1363 Working Group meeting [1]. He used it to attack the pseudorandom number generator (PRNG) specified by the then-existing DSA standard. While the attack required a prohibitive amount of resources and was not considered practical, there was enough concern about it that the PRNG in the standard was modified [22]. Although this method is known to exist by the cryptanalytic community [25,28], it remains largely undocumented and has been referred to as an "underground" attack [28]. To remedy this we describe the technique in enough detail so that interested parties can continue studying it.

Bleichenbacher's original analysis required millions of signatures in order to reduce the range of certain values so they could be used in a practical inverse FFT. Since we only had about 4 000 signatures available, we

Elke De Mulder · Mark E. Marson · Peter Pearson
Cryptography Research, Inc., 425 Market Street, 11th Floor, San Francisco, CA 94105, USA
E-mail: {elke,mark}@cryptography.com
ppearson@spamcop.net

Michael Hutter
Institute for Applied Information Processing and Communications (IAIK), Graz University of Technology, Inffeldgasse 16a, 8010 Graz, Austria
E-mail: Michael.Hutter@iaik.tugraz.at

looked for a different method of range reduction. We developed a BKZ-based method for this phase of the attack, thereby avoiding an impractical data collection step.

We experimentally confirmed our attack methodology by extracting the entire secret key from a 5-bit nonce leak using about 4 000 power traces and corresponding signatures. The attack is an iterative process. Each iteration involves the derivation of about 3 000 usable points using BKZ, followed by a pass through an inverse FFT. During each iteration a block of the most significant bits (MSBs) of the unknown part of the secret key is recovered. Finally, our simulations show that a 4-bit leak is also exploitable, with a significant increase in required resources and available signatures. Future research should improve these results.

1.1 Related work

Many attacks against nonce leaks in DSA and ECDSA have been published. Boneh and Venkatesan [6] started looking at the HNP in 1996. They mapped the HNP to a CVP and used LLL lattice reduction together with Babai's nearest plane algorithm to study the security of the MSBs of the Diffie-Hellman key exchange and related schemes.

In 1999 (and officially published in 2001), Howgrave-Graham and Smart [13] applied similar techniques to attack 160-bit DSA given multiple signatures with a fixed signing key and knowledge of some bits from each nonce. Experiments using NTL [27] showed they could recover the secret key given 8 bits of each nonce from 30 signatures, but experiments with 4 bits did not succeed.

In [23] Nguyen and Shparlinski gave a provable polynomial-time attack against DSA in which the nonces are partially known, under some assumptions on the modulus and on the hash function. They were able to recover a 160-bit key with only 3 bits of each nonce from 100 signatures, using the NTL as well. They also showed that given improved lattice reduction techniques it should be possible to recover the key with only 2 nonce bits known. In [24] the same authors extended their result to the ECDSA.

At PKC 2005, Naccache et al. [21] employed glitch attacks to ensure that the least significant bytes of the nonces were flipped to zero, allowing the authors to apply the same lattice techniques to recover keys from real smart cards. Recently, Liu and Nguyen [19] developed a new algorithm which allowed them to recover 160-bit keys with only 2 leaked nonce bits.

1.2 Roadmap

The paper is organized as follows. Sect. 2 describes how we used templates to extract the low-order bits of each nonce during the inversion step. In Sect. 3 we describe Bleichenbacher's solution to the HNP, followed by a description of the BKZ-based range reduction technique in Sect. 4. We discuss the parameter values used in the attack and some implementation issues encountered in Sect. 5. Finally, we summarize our results in Sect. 6.

2 Analysis of the Smart Card

We analyzed a commercially available smart card that implements ECDSA. The card implements the algorithm for both binary and prime field curves, and we focused on the signature generation process with the 384-bit prime field curve.

Algorithm 1 ECDSA signature generation

Require: Elliptic curve E defined over prime field curve $\mathbb{F}_p$, base point G with order q, private key x, and message hash $H = hash(m)$.
Ensure: Signature (r, s).
1: Generate a random nonce $K \in [1, q-1]$.
2: Compute $K * G = (u, v)$
3: Compute $r = u \bmod q$. If $r = 0$ then go to Step 1.
4: Compute $s = K^{-1}(H + rx) \bmod q$. If $s = 0$ then go to Step 1.
5: Return (r, s).

In this section, we describe how the algorithm is implemented on the card. We also describe power analysis results, and identify the several different leakages on the card. Finally, we describe attacks in which we recover either the secret key x or some bits of the nonce K. This paper is primarily concerned with the attack in which 7 bits from each nonce are recovered using power analysis against the modular inversion of the nonce in Line 4 of Alg. 1.

2.1 Description of the Implementation

Using both reference documentation from the manufacturer and power analysis we determined the card uses the following parameters and techniques.

1. Built-in domain parameters from ECC-Brainpool [20]. We analyzed the implementation for brainpoolP384r1.
2. Values are represented in Montgomery form for efficient arithmetic.
3. Curve points are represented in Jacobian projective coordinates.

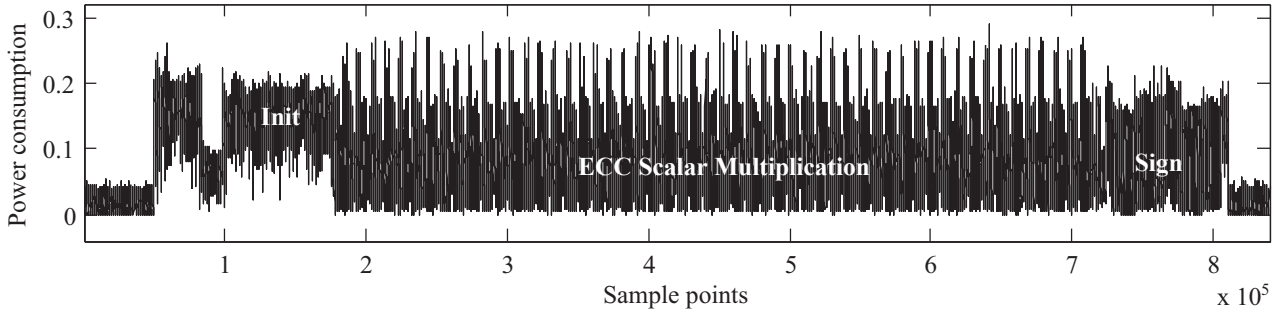

Fig. 1: Power trace of the ECDSA signature generation after post-processing.

4. Scalar multiplications take place on the curve twist brainpoolP384t1 and the final result is mapped back to brainpoolP384r1.
5. Scalar multiplications use the signed comb method [11,12] with 7 teeth. The nonces K are represented in a signed basis [15] with 385 bits.
6. The signed basis representation requires that K is odd. If K is even, q is added to it, which does not change the final result of the scalar multiplication.
7. The card stores 64 pre-computed points in memory for point additions, and computes points for subtraction on the fly.
8. $K^{-1} \bmod q$ is computed using a variant of the binary inversion algorithm.

2.2 Power Measurement Setup

The power consumption of the smart card was measured using an oscilloscope with a sampling frequency of 250 MS/s. We used two active 25 dB amplifiers (+50 dB) and a passive low-pass filter at 96 MHz. We also applied several filtering techniques to isolate the data-dependent frequency bands and downconvert them into baseband. These frequencies were identified in a prior device characterization step. Figure 1 shows a single power trace of the entire ECDSA signature generation process after signal processing.

Three main phases can be clearly identified: 1. The initial phase where the nonce K is generated, 2. the scalar multiplication $K * G$ and 3. the final phase where the signature (r, s) is calculated.

2.3 Power Analysis Attacks Against ECDSA

In this paper we are primarily concerned with attacking the modular inversion of the nonce K, in which only a few low-order bits leak. However, we found two other exploitable weaknesses in the card and will discuss them briefly.

2.3.1 Targeting the Scalar Multiplication with SPA.

Implemented correctly, the signed comb technique is naturally SPA resistant. If all the required points are pre-computed and stored in a lookup table, then the main loop of the scalar multiplication routine is very regular and avoids conditional branches. However, our previous analysis showed that the card only stores the points required for addition. When a point subtraction is needed, the index into the table is computed by complementing the tapped bits of the scalar, and then subtracting the accessed point from the current result. The power traces show a prominent spike which is only present when the point subtraction is needed.

This SPA leak revealed all 54 higher-order bits of the nonce. Figure 2 shows the leakage, after signal processing. We wrote a script to automatically extract the nonce bits. First, we aligned all traces on the elliptic curve point addition operation which could easily be identified and distinguished from the point doubling operation. After that, we trimmed all power trace segments that involve an addition operation into one single power trace to facilitate the signal processing effort. We also decimated (smoothed) the traces by a small factor to increase the success rate of the bit extraction. Finally, we applied a simple threshold detection method to reveal the 54 nonce bits. The lattice-based attack of [13] then enabled us to recover the entire secret key x using only 9 power traces.

2.3.2 Targeting the Private-Key Multiplication with DPA.

When computing the second half of an ECDSA signature, the fixed secret key x is multiplied by the known, varying first half of the signature r. This situation is typically vulnerable to standard DPA attacks [14].

Figure 3 shows the final ECDSA signature phase of the analyzed smart card. It shows the end of the scalar multiplication at the beginning of the power trace, two significant power-consumption blocks in the middle, and

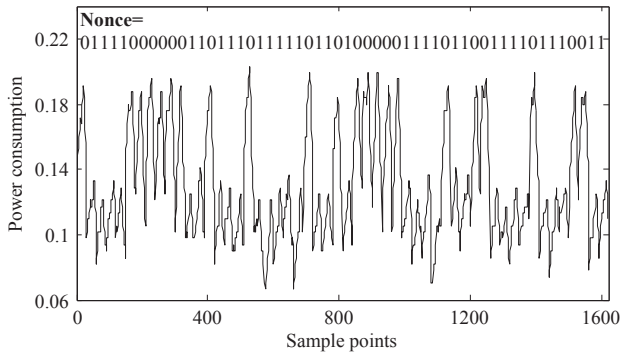

Fig. 2: SPA leak of all 54 higher-order bits of the nonce during scalar multiplication.

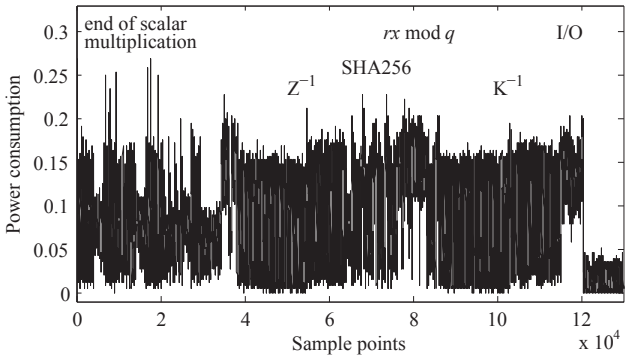

Fig. 3: Final ECDSA phase where the signature (r,s) is calculated. The DPA attack targets $rx \bmod q$.

an input/output communication at the end of the trace. Using known-key DPA we identified that the two blocks in the middle are modular inversions. The first is the modular inversion of the 'Z' coordinate to map the result of the scalar multiplication from Jacobian projective coordinates to affine coordinates. The second is the modular inversion of the secret nonce K (see Line 4 in Algorithm 1). Hashing of the message and the modular multiplication of the signature output r and the private key x is done right between the two modular inversions.

We performed the attack by targeting intermediate values during the modular multiplication of $rx \bmod q$. We measured 10 000 traces and aligned them using the least mean squares (LMS) algorithm. This analysis revealed that the card implements an MSB-first digit-serial modular multiplication method with a full multiplication of rx followed by modular reduction by division. The high-order 384 bits of the 768-bit intermediate result leak at different positions during the reduction step. Hence our attack proceeded as follows.

First, we defined a search range of 12 bits and targeted the 8th bit. After calculating all possible 4 096 hypotheses, we performed a difference of means test that showed peaks for the correct hypotheses. The remaining bits of the secret x can be recovered iteratively. In total, we recovered the entire key x in a few hours, where most of the time was spent generating all possible intermediate values.

2.3.3 Side Channel Analysis Reverse Engineering (SCARE) and Template Attack on the Inversion Algorithm

Several authors noticed that weak implementations of finite field operations such as modular additions, subtractions or multiplications can lead to successful side channel attacks [10,16,29,30]. They proposed eliminating all conditional statements and branches from both software and hardware implementations. This includes final conditional subtractions or reduction steps, which are often found in modular arithmetic operations such as Montgomery multiplication. However, we did not find any publications describing successful template attacks against modular inversions.

The analyzed smart card implements a variant of the binary inversion algorithm. This was identified after a detailed reverse engineering phase in which several intermediate variables of different inversion algorithms were targeted in known-key DPA attacks.

Analysis of the (likely) binary inversion implementation revealed that it does not run in constant time. The execution time depends on the values of both the nonce and the modulus. This is because the algorithm has several conditional branches that depend on the operands. Each branch executes different operations such as modular addition, subtraction, or simple shifts. We were able to construct a set of power consumption templates which represent the power profile for each nonce value. In the next section, we describe the template building and template matching phase in detail and show how we extracted the 7 low-order bits of the nonce with 100 % accuracy.

2.4 Recovering the Low-Order Bits of the Nonce

We targeted the low-order bits of the nonce which are processed at the beginning of the modular inversion. To limit the computational complexity we targeted the first 8 bits and generated 256 templates. 1 000 000 traces were collected: 950 000 traces for building templates and 50 000 for testing. Some sample traces are shown in Figure 4.

To build the templates we first aligned all the traces at the beginning of the modular inversion. We then sorted the traces by their similarity to the total mean trace using the LMS algorithm, and excluded all traces which had a low matching factor. A low matching factor occurred in situations when the alignment was not pos-

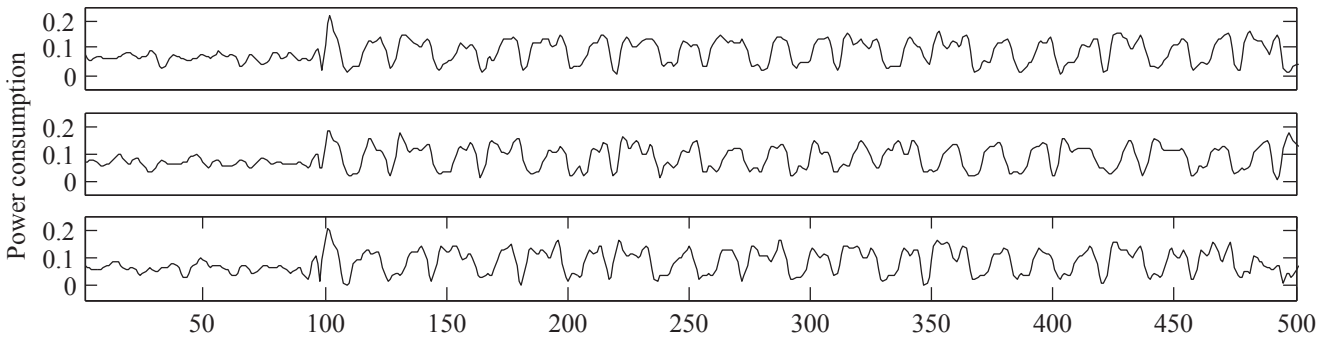

Fig. 4: Power traces during inversion of the first lower-order bits of the nonce.

sible and/or high noise was interfering with the measurement. The traces with high matching factor were then partitioned in 256 sets, based on the low-order 8 bits of the corresponding nonces. We then computed the 256 templates by averaging the traces in each partition.

To increase the success rate during the matching phase, we applied two further enhancements. First, we filtered out all the templates which had a high cross-correlation with other templates. Very similar templates which differed by only a few sample points were not used for the final byte classification. As a result, we only used 102 templates out of 256. Second, we added a length property for each template. Since the processing time of each nonce byte is not constant but variable during the inversion, the template had to be adapted such that the entire processing time is covered completely. Hence a different number of sample points was used to build the template and to match them with test traces.

2.5 Analysis Results

We used 50 000 test traces to evaluate the templates. First, we aligned all traces and filtered out 4 000 traces due to misalignment and high noise. Second, each trace was matched with all 102 templates using the LMS algorithm. For the classification, we followed a threshold detection approach by rejecting all traces that were below a certain matching threshold. Only those traces with a high quality factor (high correlation with a template) were considered as correctly classified. We obtained 4 451 candidates that met all the requirements. For these candidates, all 7 low-order bits were classified correctly, with a success rate of 100 %.

Although we were able to extract 7 bits of each nonce in our final template analysis, our earlier attempts recovered only a couple of bits reliably. Hence we decided to implement Bleichenbacher's attack and see if it could succeed with fewer bits. The remainder of this paper describes this attack for a 5-bit leak.

3 Bleichenbacher's Solution to the Hidden Number Problem

3.1 ECDSA Nonce Leaks and the Hidden Number Problem

We briefly review the basics of exploiting an ECDSA nonce leak by mapping the problem to an HNP. Our notation is mostly consistent with Bleichenbacher's presentation [4,5]. Let q be the order of the base point. For $0 \le j \le L-1$, where L is the number of signatures, let H_j denote the hashes of the messages to be signed, x the private key, K_j the ephemeral secret nonces, and r_j and s_j the two halves of the signatures. Then

$$\begin{aligned} s_j &= K_j^{-1}(H_j + r_j x) \bmod q, \\ K_j &= s_j^{-1}(H_j + r_j x) \bmod q. \end{aligned} \tag{1}$$

In our case the low-order b bits ($b = 5$) of K_j, denoted $K_{j,lo}$, were recovered using a template attack. Writing $K_j = 2^b K_{j,hi} + K_{j,lo}$ and rearranging Eq. (1) we get

$$\begin{aligned} 2^b K_{j,hi} &= (s_j^{-1} H_j - K_{j,lo}) + s_j^{-1} r_j x \bmod q, \\ K_{j,hi} &= 2^{-b}\left(s_j^{-1} H_j - K_{j,lo}\right) + 2^{-b} s_j^{-1} r_j x \bmod q. \end{aligned} \tag{2}$$

If the original K_j are randomly and uniformly generated on $[1, \ldots, q-1]$, then denoting $q_b = (q-1)/2^b$, the $K_{j,hi}$ will be randomly and almost uniformly distributed on $[0, \ldots, \lfloor q_b \rfloor]$*.

It simplifies our analysis and improves the attack to center the $K_{j,hi}$ around zero. See Sect. 4 for details. Subtracting $\lfloor q_{b+1} \rceil$ from both sides of Eq. (2) gives

$$\begin{aligned} K_{j,hi} - \lfloor q_{b+1} \rceil = 2^{-b}\left(s_j^{-1} H_j - K_{j,lo}\right) \\ + 2^{-b} s_j^{-1} r_j x \bmod q - \lfloor q_{b+1} \rceil. \end{aligned} \tag{3}$$

Let $k_j = K_{j,hi} - \lfloor q_{b+1} \rceil$, $h_j = 2^{-b}\left(s_j^{-1} H_j - K_{j,lo}\right) - \lfloor q_{b+1} \rceil \bmod q$, and $c_j = 2^{-b} s_j^{-1} r_j \bmod q$, Eq. (3) becomes

$$k_j = h_j + c_j x + \alpha_j q, \tag{4}$$

*$Pr\{K_{j,hi} == \lfloor q_b \rfloor\}$ will be less than for all other values of $K_{j,hi}$ in the interval.

where the k_j are almost uniformly distributed on $[-\lfloor q_{b+1} \rceil, \ldots, \lfloor q_{b+1} \rfloor]$ for appropriate multipliers α_j[†]. We can therefore recover the secret x by solving the following version of the hidden number problem:

Hidden Number Problem: Let $x \in [0, \ldots, q-1]$ be unknown, and suppose we have an oracle which generates random, uniformly distributed $c_j \in [1, \ldots, q-1]$ and $k_j \in [-\lfloor q_{b+1} \rceil, \ldots, \lfloor q_{b+1} \rfloor]$, computes $h_j = (k_j - c_j x) \bmod q$, and outputs the pairs (c_j, h_j). The goal is to recover x.

Lattice-based solutions have been studied extensively and will not be covered here. We only briefly note our own results with these techniques for a 384-bit modulus. Using both the CVP and SVP approaches we were able to attack 6-bit leaks using both LLL and BKZ (fplll v.4.0.1 [7]) for lattice reduction. We could attack 4 and 5-bit leaks with BKZ, but not LLL. The 4-bit attack succeeded twice in 583 trials over a range of 100-200 points per lattice.

3.2 Bias Definition and Properties

Let X be a random variable over $\mathbb{Z}/q\mathbb{Z}$. Bleichenbacher defines the bias of X as

$$B_q(X) = E(e^{2\pi i X/q}) = B_q(X \bmod q). \tag{5}$$

For a set of points $V = (v_0, v_1, \ldots, v_{L-1})$ in $\mathbb{Z}/q\mathbb{Z}$, he defines the sampled bias as

$$B_q(V) = \frac{1}{L} \sum_{j=0}^{L-1} e^{2\pi i v_j/q}. \tag{6}$$

Some properties of the bias are listed in Lemma 1 below, taken from [5].

Lemma 1 *Let $0 < T \leq q$ such that X is uniformly distributed on the interval $[-(T-1)/2, \ldots, (T-1)/2]$. Then:*

a. For independent random variables X and X', $B_q(X + X') = B_q(X)B_q(X')$.
b. $B_q(X) = \frac{1}{T}\frac{\sin(\pi T/q)}{\sin(\pi/q)}$. Hence $B_q(X)$ is real-valued with $0 \leq B_q(X) \leq 1$.
c. If X is uniformly distributed on $[0 \ldots q-1]$, then $B_q(X) = 0$.
d. Let a be an integer with $|a|T \leq q$, and $Y = aX$. Then $B_q(Y) = \frac{1}{T}\frac{\sin(\pi a T/q)}{\sin(\pi a/q)}$.
e. $B_q(Y) \leq B_q(X)^{|a|}$.

[†]We wrote Eq. (4) as an equality because the k_j can take on negative values. With this understanding, for the remainder of the paper we will simply write 'mod q'.

Proof

a. This formula follows from the fact that the probability distribution of the sum of independent variables is the convolution of the variables' distributions, together with the relationship between Fourier transforms and convolutions.
b. This formula can be computed directly using the standard formula for geometric sequences. The value is real because we centered the distribution of points about zero, and the resulting values on the unit circle are symmetric about the x-axis. Without centering the bias would be complex, with the same absolute value. Also, if T is even, then the formulas still hold, with the shifted points taking on half-integer values.
c. Follows immediately from setting $T = q$ in part b.
d. Same as part b.
e. Write

$$B_q(X) = \frac{1}{T}\frac{\sin(\pi T/q)}{\sin(\pi/q)} = \frac{\frac{\sin(\pi T/q)}{\pi T/q}}{\frac{\sin(\pi/q)}{\pi/q}} \tag{7}$$

Setting $y = \pi/q$ and $F(y) = \log(\sin(y)/y)$ we want to show that

$$F(aTy) - F(ay) \leq a\,(F(Ty) - F(y))\,. \tag{8}$$

This will be true if F is concave down. Taking the second derivative gives $F''(z) = 1/z^2 - 1/\sin^2(z)$, which is negative for $z \in (0, \pi)$. Hence Eq. (8) holds and the result is proved.

We can find convenient approximations to the formulas in Lemma 1 by taking limits as $q \to \infty$.

Lemma 2 *Suppose $R = T/q$ remains fixed as q varies, with random variables X_q uniformly distributed on $[-(T-1)/2, ..., (T-1)/2]$ for each q. Let $Y_q = aX_q$. Finally define $B_\infty(X) = lim_{q\to\infty} B_q(X_q)$ and $B_\infty(Y) = lim_{q\to\infty} B_q(Y_q)$. Then:*

a. $B_\infty(X) = \sin(\pi R)/\pi R$.
b. $B_\infty(Y) = \sin(a\pi R)/a\pi R$.

Proof L'Hôpital's rule.

Some example bias values for $R = T/q = 2^{-b}$, for large q, are shown in Table 1.

3.2.1 Most Significant vs. Least Significant Bit Leaks.

In [23] the authors noted that depending on the modulus, the most significant bit can carry less information than lower-order bits. This difference can be quantified in terms of the bias. We illustrate this by comparing 5-bit leaks for NIST P-384 and brainpoolP384r1.

The base point for the NIST curve has order $q =$ 0xFFFFFFFF..., and for the Brainpool curve $q =$

Table 1: Example bias values for $R = 2^{-b}$

b	1	2	3	4	5	6	7	8
$\mathbf{B_q(X)}$	0.6366198	0.9003163	0.9744954	0.9935869	0.9983944	0.9995985	0.9998996	0.9999749

0x8CB91E82... If the low-order 5 bits leak, then for either prime we get $T = \lfloor q/2^5 \rfloor$ and $R = T/q \approx 2^{-5}$, for a bias of 0.9984.

If the high-order 5 bits leak, then $T = 2^{379}$. For the NIST prime, we still have $R = T/q \approx 2^{379}/2^{384} = 2^{-5}$. Hence the work to attack a 5-bit leak is the same whether the MSBs or LSBs are recovered. On the other hand, for the Brainpool prime we have $R = T/q \approx$ 0x8/0x8C =1/17.5 and a resulting bias of 0.9946. This is much closer to the value for a 4-bit LSB leak.

Our experiments confirm these calculations. For the NIST prime the work factor for the attack does not depend on whether the MSBs or LSBs are leaked. On the other hand, for the Brainpool prime the work required to attack a 5-bit leak of the MSBs is on par with the work to attack a 4-bit leak of the LSBs. Given the form of the Brainpool prime, about 8/9 of the time the high-order bit of a randomly generated nonce is zero. Hence when the MSBs are leaked, we gain on average very little additional information about the high-order bit.

3.3 Connecting the Hidden Number Problem to the Bias

In an instance of the HNP, we are given a modulus q and a set of pairs (c_j, h_j), $0 \le j < L$, and we wish to find the secret and presumably unique x for which the set of values $V_x = \{h_j + c_j x \bmod q\}_{j=0}^{L-1}$ all fall near 0 or q. If they do, then this set of values will show a significantly nonzero sampled bias. Furthermore, for any w different from x, we expect that the values $V_w = \{h_j + c_j w \bmod q\}_{j=0}^{L-1}$ would show a relatively small sampled bias. To see why, for $0 \le w < q$ let us define[‡]

$$\begin{aligned} B_q(w) &= \frac{1}{L}\sum_{j=0}^{L-1} e^{2\pi i(h_j + c_j w)/q} \\ &= \sum_{t=0}^{q-1}\left(\frac{1}{L}\sum_{\{j|c_j=t\}} e^{2\pi i h_j/q}\right) e^{2\pi i t w/q} \\ &= \sum_{t=0}^{q-1}\left(\frac{1}{L}\sum_{\{j|c_j=t\}} e^{2\pi i(h_j + c_j x)/q}\right) e^{2\pi i t(w-x)/q} \\ &= \sum_{t=0}^{q-1}\left(\frac{1}{L}\sum_{\{j|c_j=t\}} e^{2\pi i k_j/q}\right) e^{2\pi i t(w-x)/q}. \end{aligned} \tag{9}$$

[‡]We acknowledge the abuse of notation in writing $B_q(w)$ instead of $B_q(V_w)$, but this is consistent with Bleichenbacher's notes and will simplify the exposition.

If $w = x$, then $B_q(w) = \frac{1}{L}\sum_{j=0}^{L-1} e^{2\pi i k_j/q}$ is just the sampled bias of the points $k = (k_0, k_1, \ldots, k_{L-1})$. Given a b-bit nonce leak ($R = 2^{-b}$ in Table 1) and enough samples, $B_q(x)$ will have a value close to 1, as all the terms $e^{2\pi i k_j/q}$ are confined to the part of the unit circle with phase $-\pi/2^b < \theta < \pi/2^b$.

If $w \ne x$, then $B_q(w)$ will be close to zero. The inner sums in the last line of Eq. (9) get rotated around the unit circle by the $e^{2\pi i t(w-x)/q}$ terms, and largely cancel each other out. This effect is illustrated by Figures 5 and 6.

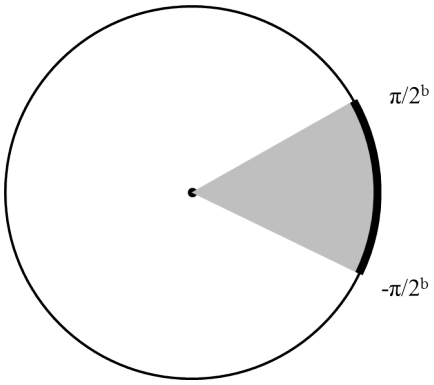

Fig. 5: When $w = x$, all the summands in Eq. (9) have bounded phase $-\pi/2^b < \theta < \pi/2^b$.

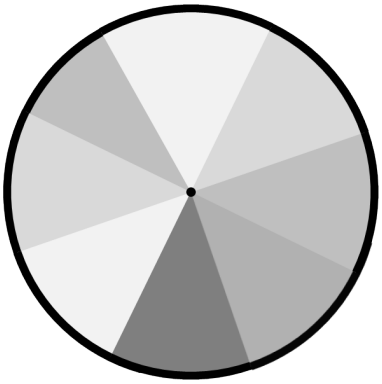

Fig. 6: When $w \ne x$, the inner sums in the last line of Eq. (9) get rotated around the unit circle by different angles and cancel each other out.

Thus, the bias calculation gives us a way to score putative solutions to the HNP, allowing us to search for the correct value x which maximizes $B_q(w)$. Evaluating

it for all w in $[0, ..., q-1]$ is, of course, impractical for large q. [§]

Bleichenbacher's insight [5] was that if the c_j are small enough, the peak in $B_q(w)$ centered about x will be broad enough to discern even if we only calculate $B_q(w)$ for an extremely sparse set of w. Furthermore, we can synthesize (c_j, h_j) pairs with small c_j by taking linear combinations (mod q) of the original (c_j, h_j) pairs.

To see why the peak of $B_q(w)$ is broad when the c_j are small, note that in the second line of Eq. (9), $B_q(w)$ is a weighted sum of terms $e^{2\pi itw/q}$, with frequencies $t/q = c_j/q$. If those frequencies are all much smaller than 1, the peak of $B_q(w)$ will be broad, reducing the search work proportionally. How small must the c_j be to make the peak centered at x broad enough to find?

Suppose we have a bound $0 < C \ll q$ such that all the c_j satisfy $0 \le c_j < C$. Then the inner sums in the last line of Eq. (9) will be combined after rotation by angles $\theta_t = 2\pi t(w-x)/q$ satisfying $|\theta_t| < 2\pi C|w-x|/q$, as only those terms for which $t < C$ will be nonzero. If C is small enough, then for w close to x these rotations will be small. There will be no cancellation of terms during summation, and $B_q(w)$ will approximate $B_q(x)$. This situation is illustrated in Figure 7.

At what values of θ_t do we expect the terms to begin canceling out? Referring again to Figure 7, we can see cancellation starts occurring once $|\theta_t| > \pi/2$, as the real parts of those terms begin to take on negative values. Accordingly, $B_q(w)$ begins falling significantly below its peak when $2\pi C|w-x|/q \approx \pi/2$, or equivalently $|w-x| \approx q/4C$. This implies the peak of $B_q(w)$ centered at x should have a width of approximately $q/2C$.

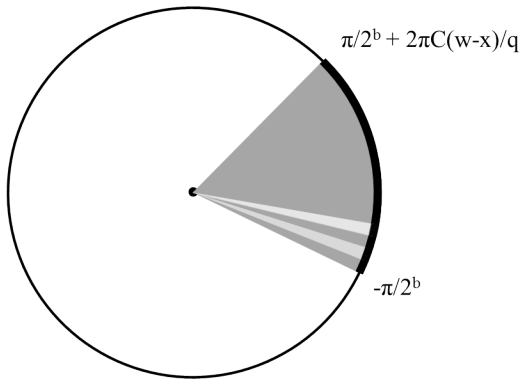

Fig. 7: When the c_j are bounded and w is close to x, there is little or no cancellation of the inner sums in the last line of Eq. (9).

[§]Throughout the paper we will refer to the size of $B_q(w)$, by which we mean $|B_q(w)|$. With this understanding, for the remainder of the paper we will leave off the absolute value bars.

Therefore, if the c_j are bounded by C we can find an approximation to x by searching for the peak value in $B_q(w)$ over $n = 2C$ evenly-spaced values of w between 0 and q. To do so, set $w_m = mq/n$, $m \in [0, n-1]$, in Eq. (9). Then

$$\begin{aligned} B_q(w_m) &= \frac{1}{L}\sum_{j=0}^{L-1} e^{2\pi i(h_j + (c_j mq/n))/q} \\ &= \frac{1}{L}\sum_{j=0}^{L-1} e^{(2\pi i h_j/q) + (2\pi i c_j m/n)} \\ &= \sum_{t=0}^{n-1}\left(\frac{1}{L}\sum_{\{j|c_j=t\}} e^{2\pi i h_j/q}\right) e^{2\pi itm/n} \\ &= \sum_{t=0}^{n-1} Z_t e^{2\pi itm/n} \end{aligned} \tag{10}$$

where $Z_t = \frac{1}{L}\sum_{\{j|c_j=t\}} e^{2\pi i h_j/q}$.

The observant reader may recognize the above formula as the inverse FFT of $Z = (Z_0, Z_1, \ldots, Z_{n-1})$. Note that since $n = 2C$ and all the c_j are less than C, the coefficients $(Z_C, \ldots, Z_{n-1})$ are all zero. Hence the $B_q(w_m)$ can be efficiently computed by first computing the vector Z, and then taking the inverse FFT. In practice, we are limited in the number of $B_q(w)$ we can evaluate at a given time by the maximum FFT size we can efficiently compute. Hence, we require (c_j, h_j) pairs with sufficiently small c_j. We will discuss range reduction in Sect. 4. For the next section we will assume the c_j are appropriately bounded.

3.4 Recovering the Secret x with Bounded c_j

Suppose we can compute an $n = 2^N$-point inverse FFT. Then we can recover the high-order N bits of the x as follows.

1. Zero the vector Z.
2. Loop over all L pairs (c_j, h_j). For each pair add $e^{2\pi i h_j/q}$ to the appropriate Z_t, namely $t = c_j$.
3. Compute the inverse FFT of Z and find the m for which $B_q(w_m)$ is maximal.
4. The most significant N bits of x are $msb_N(x) = msb_N(\lfloor mq/n \rfloor)$.

We can repeat the process iteratively to recover the remaining bits of x. Let $x = 2^u x_{hi} + x_{lo}$, where x_{hi} are the known bits previously recovered, and x_{lo} is u bits in length and unknown. We first rewrite Eq. (4) to absorb the known bits x_{hi} into h_j:

$$\begin{aligned} k_j &= (h_j + c_j x) \bmod q \\ &= ((h_j + 2^u c_j x_{hi}) + c_j x_{lo}) \bmod q \\ &= (h'_j + c_j x_{lo}) \bmod q. \end{aligned} \tag{11}$$

The computation proceeds as before, except we evaluate $B_q(w)$ over n evenly spaced values of w between 0 and 2^u, since only u bits remain unknown. Mimicking the previous computation, set $w_m = 2^u m/n$, $m \in [0, n-1]$ in Eq. (10):

$$\begin{aligned} B_q(w_m) &= \frac{1}{L}\sum_{j=0}^{L-1} e^{2\pi i(h'_j + (2^u c_j m/n))/q} \\ &= \frac{1}{L}\sum_{j=0}^{L-1} e^{(2\pi i h'_j/q) + (2\pi i (2^u c_j m/qn))} \\ &= \sum_{t=0}^{n-1}\left(\frac{1}{L}\sum_{\{j|(2^u c_j/q)=t\}} e^{2\pi i h'_j/q}\right) e^{2\pi i t m/n} \\ &= \sum_{t=0}^{n-1} Z_t e^{2\pi i t m/n} \end{aligned} \tag{12}$$

where $Z_t = \frac{1}{L}\sum_{\{j|\lfloor 2^u c_j/q\rfloor = t\}} e^{2\pi i h'_j/q}$. As before, compute the $B_q(w_m)$ by taking the inverse FFT of Z, and find the m with the maximum value for $B_q(w_m)$. The most significant N bits of x_{lo} are $msb_N(\lfloor 2^u m/n \rfloor)$.

If the c_j remain bounded by $C = n/2$ as the attack proceeds, then $2^u c_j/q$ would always be zero as soon as $2^u < 2q/n$, and the only nonzero coefficient would be Z_0. Instead, we want $2^u c_j/q$ to range between 0 and $n/2$, so we relax the bound on the c_j and set $C = nq/2^{u+1}$ as additional bits of x are recovered.

Note that in the above description, and in the attack we implemented, we rounded mq/n and $2^u m/n$ down, and used those values as the recovered bits of x. This strategy was probably not ideal, and was undoubtedly helped by the windowing we also performed (see Sect. 5). It would be a more robust strategy to round those values both up and down, and then use both in the next iteration. The incorrect value should be easily identified and discarded, as using it in the next iteration should result in all the calculated biases being small.

4 Range Reduction

The original c_j will be uniformly distributed in $[1, q-1]$ and not nicely constrained as required for the attack described above. It remains for us to demonstrate how to find (c_j, h_j) pairs whose c_j values lie in an extremely narrow range. Suppose we have L pairs (c_j, h_j) such that the corresponding values $k_j = h_j + c_j x \bmod q$ are biased. Then the $L^2 - L$ values

$$k_{i,j} = (h_i + h_j) + (c_i + c_j)x \bmod q, \quad i \neq j, \tag{13}$$

will also be biased. Assuming independence of the new pairs (an untrue but useful assumption) Lemma 1 tells us that the bias of the new $k_{i,j}$ will be the square of the bias of the original k_j.

We demonstrate the effect of taking linear combinations on the bias using a toy example. We set the modulus to $q = 2^{16} + 1$ and selected the secret x. We also generated 2^{12} random k_j bounded by 2^{13}, to simulate a 3-bit leak, and 2^{12} random c_j in $[1, ..., q-1]$. We then solved for the h_j such that $k_j = h_j + c_j x \bmod q$. The pdf of the k_j and their bias is shown in the top pair of graphs in Figure 8.

We then randomly added or subtracted pairs of $c_j \bmod q$, and took the the corresponding linear combinations of the h_j and k_j, until we had 2^{12} new c_j which were bounded by 2^{14}. This process was repeated, reducing the c_j by two bits each iteration, until we had 2^{12} c_j bounded by $2^4 = 16$. The effects on the probability distributions and biases of the corresponding k_j are shown on the left and right hand sides of Figure 8.

As the k_j get combined, their probability distributions get increasingly flattened and spread out over the entire interval $[0, ..., q-1]$. The bias for the correct value x also decreases, but the biases for w near x also start increasing, as the range of the c_j decreases. In the final row we see a shorter but broad and clearly dominant peak centered around the correct value x. We can therefore find the high-order 4 bits of x by searching over $2^4 = 16$ equally-spaced values in $[0, ..., q-1]$, rather than find all 16 bits of x with a 2^{16} search.

The goal of range reduction is to broaden the peak centered about x so that it can be found with an exhaustible search, without flattening it so much that it becomes indistinguishable from noise. If successful, this allows a divide-and-conquer strategy in which blocks of bits of x are recovered in an iterative fashion.

Bleichenbacher's original analysis was for very small, even fractional, bit leaks, and he used millions of signatures, and large FFT sizes. His range reduction strategy was to look for collisions in the high-order bits of the c_j and take their differences. For example, if you have L points in $[0, q-1]$, sort them by size, and subtract each point from the next largest one, you get a list of $L-1$ points, which are on average about $\log(L)$ bits smaller. This process can be repeated until the points are in the desired ranges.

In our case we are working with a larger bit leak, but we only have a few thousand signatures, a 384-bit modulus instead of a 160-bit modulus, and a 2^{28}-point FFT. If we had 2^{30} signatures, we could employ the sort-and-difference method to the c_j for 12 rounds, taking differences of the corresponding h_j as well. This would result in a large number of c'_j satisfying $0 \le c'_j < 2^{24}$. The sampled bias for the corresponding points $k'_j = h'_j + c'_j x \bmod q$ would be about $(0.9984)^{2^{12}} = 0.0014$, which is large enough for the attack to succeed.

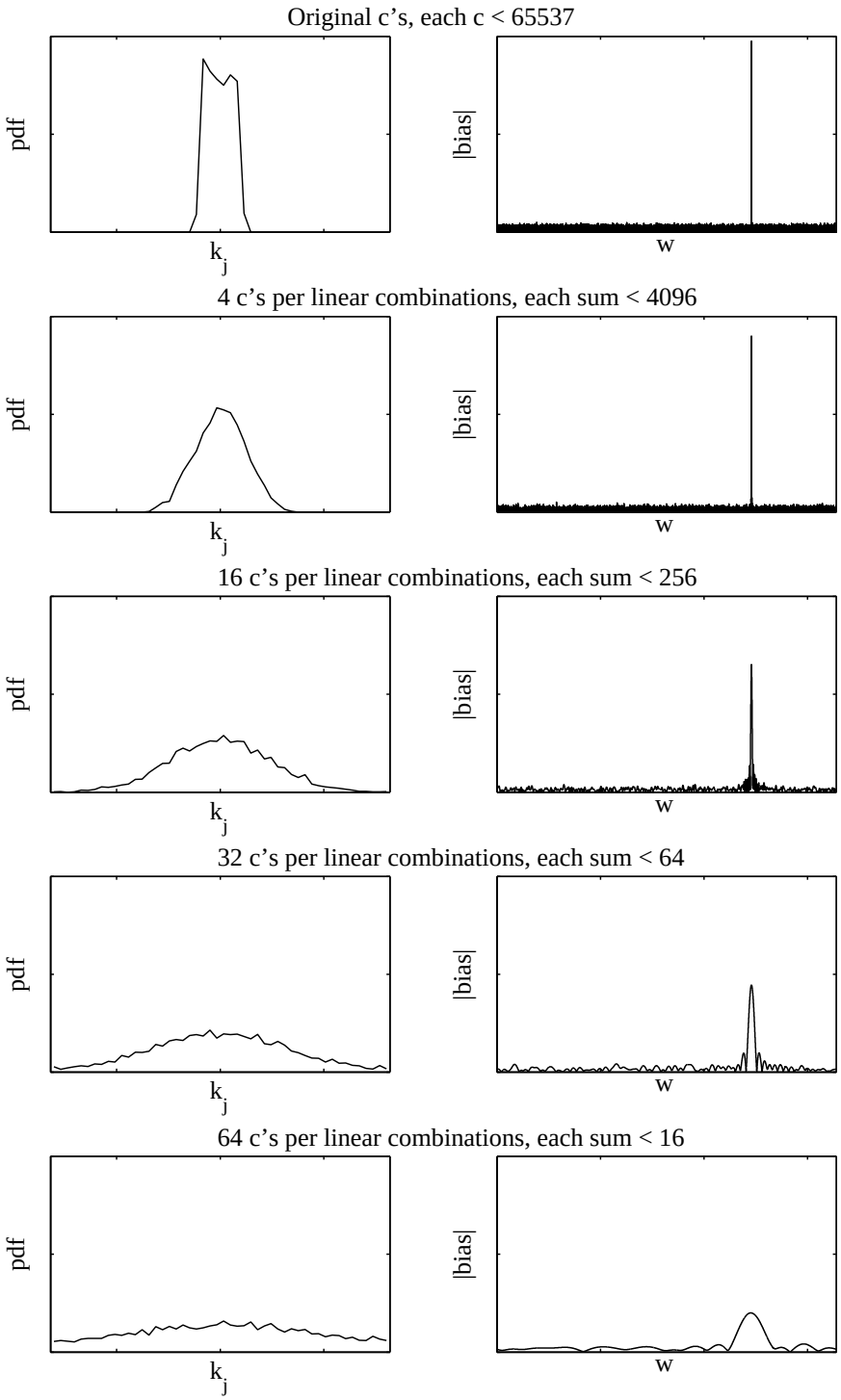

Fig. 8: Effect of taking linear combinations of the c_j, on the distribution of the nonces and their biases.

If we had 2^{20} signatures and applied the sort-and-difference method for 18 rounds, the resulting c'_j would also be in the same range, but the sampled bias would be about 10^{-182}, far too small to be useful. With about 4 000 signatures available, we looked for another strategy to reduce the range of the c_j without reducing the bias too much.

Given subsets $c_J = (c_{J,0}, c_{J,1}, ..., c_{J,d-1})$ of the c_j, we want to find sets of integer coefficients

$$A_J = (a_{J,0}, a_{J,1}, ..., a_{J,d-1}),$$

such that

$$c_{A_J} = \langle A_J, c_J \rangle \bmod q = \sum_{t=0}^{d-1} a_{J,t} c_{J,t} \bmod q \tag{14}$$

satisfies $0 \le c_{A_J} < C$. Applying the A_J to Eq. (4) gives

$$k_{A_J} = (h_{A_J} + c_{A_J} x) \bmod q, \tag{15}$$

where $k_{A_J} = \langle A_J, k_J \rangle \bmod q$, $h_{A_J} = \langle A_J, h_J \rangle \bmod q$, and the c_{A_J} are small enough to be used in the FFT calculation. The expected bias of the k_{A_J} can be approximated by applying the rules from Lemma 1 and Lemma 2.

It is actually a trivial task to find such A_J. The difficulty lies in finding them subject to the condition that the bias of the resulting k_{A_J}still be large enough to detect against background noise.

The most relevant metrics are the L_1 norm $\|A_J\|_1$, and the L_∞ norm $\|A_J\|_\infty$, which must be sufficiently bounded. Finding bounds G_1 and G_∞ for these norms for which the attack succeeds is discussed in Sect. 4.1. Given these bounds, however, we can use BKZ for range reduction and keep only those points which satisfy them.

Consider the lattice spanned by the rows of the following matrix:

$$\begin{bmatrix} W & 0 & 0 \cdots & 0 & c_{J,0} \\ 0 & W & 0 \cdots & 0 & c_{J,1} \\ \vdots & \vdots & \vdots \; \vdots & \ddots & \vdots \\ 0 & 0 & 0 \cdots & W & c_{J,d-1} \\ 0 & 0 & 0 \cdots & 0 & q \end{bmatrix}. \tag{16}$$

The $c_{J,l}$ are randomly chosen from our list of points, and W is a weight factor to balance the reduction of the $c_{J,l}$, and the size of the resulting coefficients. Applying BKZ to the matrix gives

$$\begin{bmatrix} a_{J,0,0}W & a_{J,0,1}W & a_{J,0,2}W & \cdots & a_{J,0,d-1}W & c_{A_{J,0}} \\ a_{J,1,0}W & a_{J,1,1}W & a_{J,1,2}W & \cdots & a_{J,1,d-1}W & c_{A_{J,1}} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{J,d,0}W & a_{J,d,1}W & a_{J,d,2}W & \cdots & a_{J,d,d-1}W & c_{A_{J,d}} \end{bmatrix} \tag{17}$$

where $A_{J,l} = (a_{J,l,0}, a_{J,l,1}, ..., a_{J,l,d-1})$ for $0 \le l \le d$ and $c_{A_{J,l}} = \langle A_{J,l}, c_J \rangle \bmod q = \sum_{t=0}^{d-1} a_{J,l,t} c_{J,t} \bmod q$. To simplify notation we will drop the second index l for the rest of the paper.

We want the above lattices to contain points $c_{A_J} \in (-C, C)$ for which $\|A_J\|_1 \le G_1$, and the $\|A_J\|_\infty \le G_\infty$. The number of good points per lattice depends not only on those bounds, but also the dimension d, the weight W, and the BKZ parameters. We determined these experimentally. For the first iteration of the attack we used $d = 128$, a BKZ blocksize of 20, $W = 2^{25}$, $C = 2^{28}$, $G_1 = 325$ and $G_\infty = 8$.

Recall from the discussion in Sec. 3.3 that C should be 2^{27}, not 2^{28}. However, we increased it by a bit in order to find more reduced points. This doubled the size of the maximum rotation angles to $|\theta_t| \approx \pi$ and accordingly decreases $B_q(w)$ for w near the endpoints of the interval containing x. Nevertheless, the peak in the bias was still broad and tall enough to identify the correct interval.

We can now explain the main reason for centering the k_j around zero. It mitigates the reduction of the bias when taking linear combinations. To see why, suppose the range of two independent variables X and X' is $[0, ..., T-1]$. Then the range of $\pm(X \pm X')$ is $[-2(T-1), ..., 2(T-1)]$. On the other hand, if the range of X and X' is $[-(T-1)/2, ..., (T-1)/2]$, then the range of $\pm(X \pm X')$ is $[-(T-1), ..., T-1]$. Hence if the original k_j are centered about zero, then the k_{A_J} in $[0, ..., q-1]$ are more densely clustered near 0 and q, and therefore have a larger bias. In fact, centering the k_j improves the performance of the attack by about a bit in number of leaked nonce bits.

4.1 Finding BKZ Parameters for Range Reduction

The weights W are required to balance the reduction of the c_j with the size of the coefficients. If W is too small, say $W = 1$, then the lattice would contain many $c_{A_J} \in (-C, C)$ but the coefficients would be too large. On the other hand, if W is too large, say $C \le W$, the coefficient norms would be smaller, but there would be very few if any $c_{A_J} \in (-C, C)$. The values for W which appear to work best are those which are just a few bits smaller than C. In our lattices we set $W = 2^{\lfloor log(C) \rfloor - 3}$.

We now turn to the question of the coefficient bounds G_1 and G_∞. We need to know for which bounds the attack will work, and how many (c_{A_J}, h_{A_J}) pairs are required. Ideally we would run experiments using reduced points output by BKZ. Unfortunately, the lattices which worked best were large, containing 128 or more points. Each lattice reduction took 1-2 minutes, and yielded only a few, if any useful points. This made

it impractical to use BKZ outputs of our real data to analyze coefficient bounds.

We therefore analyzed the distribution of coefficients output by BKZ in order to simulate them. For our analysis and attack, we used BKZ with $d = 128$, and a blocksize of 20. We randomly populated and reduced lattices, and sorted the c_{A_J} based on the bounds C, G_1 and G_∞ they satisfied. Once we had enough c_{A_J} for each set of bounds, we examined the distribution of nonzero coefficients. An example is shown in Figure 9, for $C = 2^{28}$, $G_1 = 325$ and $G_\infty = 8$. The distribution strongly resembles a normal distribution, and a normal fit also appears to match the data.[¶] We therefore modeled the output of BKZ using normal distributions, after getting estimates of the coefficient standard deviations for different sets of bounds.

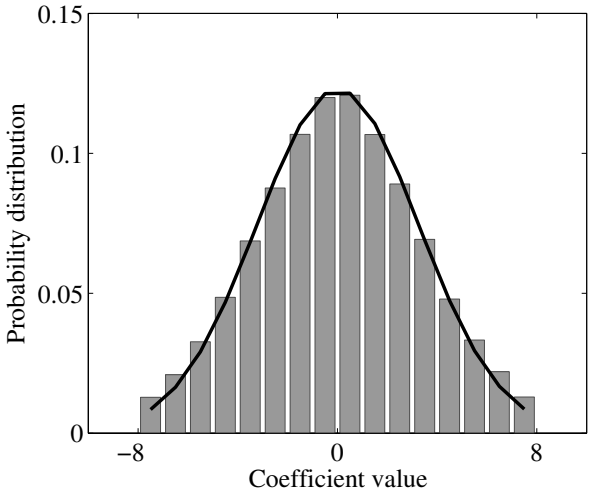

Fig. 9: Distribution and normal fit of the nonzero coefficients output by BKZ.

Note that this was not intended as a rigorous mathematical analysis. We only needed a reasonable model of the coefficient distribution for our simulations. Once we had that, we generated simulated data points (c_j, h_j) and coefficients A_J with $\|A_J\|_1 \leq G_1$ and $\|A_J\|_\infty \leq G_\infty$ such that the $c_{A_J} \in (-C, C)$. We then performed the FFT phase of the attack in order to determine the number of (c_{A_J}, h_{A_J}) pairs required for success. The simulations were accurate, and successfully predicted the number of points required by the attack on actual data. For example, our simulations predicted that for a 5-bit leak with bounds $G_1 = 325$ and $G_\infty = 8$, the first phase of the attack would succeed with about 3 000 reduced pairs (c_{A_J}, h_{A_J}), matching what occurred in the real attack.

[¶]Curiously, the coefficient distributions output by LLL were better modeled by geometric distributions.

5 Attack Details and Observations

The attack consists of multiple iterations, in which additional bits of x are recovered in each iteration. Each iteration consists of two phases: range reduction using BKZ, followed by the inverse FFT calculation. The first iteration, with the smallest value for C, is the most difficult. As the attack proceeds and C increases, we can find more points $c_{A_J} \in (-C, C)$ with smaller coefficient bounds G_1 and G_∞, so fewer points are required for the FFT phase.

We kept a short list of the top 10 scoring candidates for x from each iteration. We chose to keep 10 candidates based on our experiments with the given bias and the number of points available for the inverse FFT. The correct answer x was not always the top candidate, but was always in the top 10, and was the top candidate after the final iteration.

We also used overlapping windows as we successively recovered bits of x, keeping the high-order 20 bits out of the 28 recovered. We did this for two reasons. First, the results of the FFT are sometimes off by a few of the low order bits. This is more of an issue when the number of points available is barely sufficient. The second reason is that we used this 8-bit value to round the current approximation of x for the next iteration. We found this rounding essential for getting the correct result in the next iteration. After the next block of bits of x is recovered, the rounding from the previous iteration is undone.

The attack succeeded using 3 000 reduced points for each iteration, derived from the original 4 000 signatures. However, the work factor and time required was worse than the standard lattice attacks. For the BKZ phase of the first iteration, we used the bounds and lattice parameters discussed above. Each lattice reduction took about 2 minutes, and returned on average 2 usable points. This phase is easy to parallelize, and took about 4 hours to complete on 12 cores. Each 2^{28}-point FFT took 30 seconds, for a total of 5 minutes. The second iteration was similar, as the increase in C did not improve the BKZ outputs much. The remaining iterations were significantly easier, and the rest of the attack took a few hours to complete.

6 Conclusions

In this paper we described an attack against a nonce leak in 384-bit ECDSA running on a smart card. We used a template attack to recover a few low-order bits from each nonce. We then used Bleichenbacher's solution to the HNP, where we had a much larger modulus and far fewer signatures than in his original analysis. Without enough signatures to perform his collision searches, we used BKZ for range reduction.

Our attack succeeded against a 5-bit leak with about 4 000 signatures, although the time and resources required is worse than what can be done with standard lattice-based attacks. However, our technique will continue to scale with fewer bits. For example, our simulations also show that we could attack a 4-bit leak with 500 000 reduced points satisfying $G_1 = 250$ and $G_\infty = 5$. Finding these points does not appear feasible with the lattice reduction software we used. However, it may be possible to find them using improved implementations such as BKZ 2.0 [9]. There is still a lot of room for improvement in our results, and we hope this paper spurs more research on Bleichenbacher's method.

Acknowledgements. We would like to thank Pankaj Rohatgi and Mike Hamburg for many fruitful discussions and valuable suggestions.

References

1. Minutes from the IEEE P1363 Working Group for Public-Key Cryptography Standards, November 15 2000.
2. ANSI X9.62:2005: Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA), 2005.
3. L. Babai. On Lovász' Lattice Reduction and the Nearest Lattice Point Problem. *Combinatorica*, 6(1):1–13, 1986.
4. D. Bleichenbacher. On The Generation of One-Time Keys in DL Signature Schemes. Presentation at IEEE P1363 Working Group meeting, November 2000.
5. D. Bleichenbacher. On the Generation of DSA One-Time Keys. Presentation at Cryptography Research, Inc., San Francisco, CA, 2007.
6. D. Boneh and R. Venkatesan. Hardness of Computing the Most Significant Bits of Secret Keys in Diffie-Hellman and Related Schemes. In N. Koblitz, editor, *CRYPTO 1996*, volume 1109 of *LNCS*, pages 129–142, 1996.
7. D. Cadé, X. Pujol, and D. Stehlé. fplll-4.0.1 Lattice Reduction Library, 2012.
8. S. Chari, J. R. Rao, and P. Rohatgi. Template Attacks. In B. S. Kaliski Jr., Ç. K. Koç, and C. Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 13–28. Springer, 2002.
9. Y. Chen and P. Q. Nguyen. BKZ 2.0: Better Lattice Security Estimates. In *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2011.
10. G. Hachez and J.-J. Quisquater. Montgomery Exponentiation with no Final Subtractions: Improved Results. In Ç. K. Koç and C. Paar, editors, *CHES 2000*, volume 1965 of *LNCS*, pages 91–100. Springer, 2000.
11. M. Hamburg. Fast and Compact Elliptic-Curve Cryptography. IACR Cryptology ePrint Archive 2012: 309.
12. M. Hedabou, P. Pinel, and L. Beneteau. A Comb Method to Render ECC Resistant Against Side Channel Attacks. IACR Cryptology ePrint Archive 2004: 342.
13. N. Howgrave-Graham and N. P. Smart. Lattice Attacks on Digital Signature Schemes. *Designs, Codes and Cryptography*, 23(3):283–290, August 2001.
14. M. Hutter, M. Medwed, D. Hein, and J. Wolkerstorfer. Attacking ECDSA-Enabled RFID Devices. In M. Abdalla, D. Pointcheval, P.-A. Fouque, and D. Vergnaud, editors, *ACNS 2009*, volume 5536 of *LNCS*, pages 519–534. Springer, 2009.
15. M. Joye and M. Tunstall. Exponent Recoding and Regular Exponentiation Algorithms. In B. Preneel, editor, *AFRICACRYPT 2009*, volume 5580 of *LNCS*, pages 334–349, 2009.
16. P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In N. Koblitz, editor, *CRYPTO 1996*, volume 1109 of *LNCS*, pages 104–113. Springer, 1996.
17. P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In M. J. Wiener, editor, *CRYPTO 1999*, volume 1666 of *LNCS*, pages 388–397, 1999.
18. A. K. Lenstra, H. Lenstra, and L. Lovász. Factoring Polynomials with Rational Coefficients. *Mathematische Annalen*, 261:515–534, 1982.
19. M. Liu and P. Q. Nguyen. Solving BDD by Enumeration: An Update. In E. Dawson, editor, *Topics in Cryptology - CT-RSA 2013*, volume 7779 of *LNCS*, pages 293–309. Springer, 2013.
20. M. Lochter and J. Merkle. Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation. RFC 5639 (Informational), Mar. 2010.
21. D. Naccache, P. Q. Nguyen, M. Tunstall, and C. Whelan. Experimenting with Faults, Lattices and the DSA. In S. Vaudenay, editor, *PKC 2005*, volume 3386 of *LNCS*, pages 16–28. Springer, 2005.
22. National Institute of Standards and Technology (NIST). FIPS-186-2 (+Change Notice): Digital Signature Standard (DSS), January 2000. Available online at `http://www.itl.nist.gov/fipspubs/`.
23. P. Q. Nguyen and I. Shparlinski. The Insecurity of the Digital Signature Algorithm with Partially Known Nonces. *J. Cryptology*, 15(3):151–176, 2002.
24. P. Q. Nguyen and I. Shparlinski. The Insecurity of the Elliptic Curve Digital Signature Algorithm with Partially Known Nonces. *Des. Codes Cryptography*, 30(2):201–217, 2003.
25. J.-J. Quisquater and F. Koene. DSA Security Evaluation of the Signature Scheme and Primitive. Technical report, Math RiZK, K2Crypt, February 2002.
26. C.-P. Schnorr and M. Euchner. Lattice Basis Reduction: Improved Practical Algorithms and Solving Subset Sum Problems. *Mathematical Programming*, 66:181–199, 1994.
27. V. Shoup. NTL: A Library for doing Number Theory, 2012.
28. S. Vaudenay. Evaluation Report on DSA. IPA Work Delivery 1002, 2001.
29. C. D. Walter. Montgomery Exponentiation needs no Final Subtractions. *Electronics Letters*, 35:1831–1832, 1999.
30. C. D. Walter and S. Thompson. Distinguishing Exponent Digits by Observing Modular Subtractions. In D. Naccache, editor, *CT-RSA 2001*, volume 2020 of *LNCS*, pages 192–207. Springer, 2001.