LadderLeak: Breaking ECDSA with Less than One Bit of Nonce Leakage

Report from the paper

Edoardo Saputelli

Politecnico di Milano

November 3, 2022

Table of Contents

Introduction

What is ECDSA?

- The **ECDSA** (Elliptic Curve Digital Signature Algorithm) offers a variant of the DSA algorithm, using elliptic-curve cryptography.
- It is widely employed today, being part of many practical cryptographic protocols such as TLS and SSH.
- Its signing operation relies on an ephemeral random value called **nonce**
  - it is crucial to keep it in secret in order not to leak any information on the digital signature.
- Cryptanalysts have discovered that is possible to recover the secret key even knowing just short substrings of the nonce. For this reason, **HNP** (Hidden Number Problem) attacks have been developed in order to exploit the ECDSA algorithm.

HNP Attacks

- Lattice-based attacks
  - they perform very efficiently with sufficiently long substrings of the nonces;
  - however, they become essentially inapplicable when only a very small fraction of the nonce is known or if the retrieved information presents leakage errors.
- Fourier analysis-based attacks (by Bleichenbacher)
  - they can tackle arbitrary small nonce biases and handle erroneous inputs out of the box;
  - nevertheless, they have a limited practicality, due to the need of particular conditions, not easy to obtain.
- Therefore, the work presents the first real-world ECDSA vulnerabilities not susceptible to lattice attacks and practically exploitable with the Fourier analysis method.

Preliminaries

Cache Attacks

- In order to deliver high-level performances, modern computers employ some techniques with the aim of predicting programs' behaviour, optimizing in this way the processor's work.
  - As a consequence, program execution affects the future behaviour of the processor;
  - Therefore, by monitoring its own performance, a program can learn about execution patterns of other programs.
    In this way, an **unintended and unmonitored communication channel** is created.
- Over the years, some attacks have been developed in order to exploit leakage through the internal state of various microarchitectural components.
- An example is the **Flush+Reload attack**, which monitors victim access to a memory location.
  - It consists in evicting the monitored memory location and then accessing it, measuring the access time in order to understand if the victim has accessed that memory location.
  - Repeating the attack, an attacker can recover the memory usage patterns of the victim, allowing cryptographic attacks.

The Montgomery Ladder and its Secure Implementation

- Given the math defining elliptic curves, many cryptographic protocols are based on the **ECDLP** (Elliptic Curve Discrete Logarithm Problem), i.e. find k given inputs $(P, [k]P)$.
  - In many settings, an attacker aims to recover bits of the scalar k, in order to ease the necessary effort to solve the ECDLP.
- The **Montgomery Ladder** is a method employed to compute scalar multiples of points of elliptic curves. It was chosen for its capability of keeping regularity in the way the scalar is processed, also because complete addition laws for Weierstrass curves incur a substantial performance penalty.
- Therefore, satisying some basic preconditions, a constant-time implementation of the Montgomery Ladder protected against timing attacks.

ECDSA and Hidden Number Problem with Erroneous Input

- The signing key extraction from the nonce leakages in ECDSA signatures typically amounts to solving the so-called hidden number problem (HNP).
- Let q be a prime and $sk \in \mathbb{Z}_q$ be a secret.
  Let h_i and k_i be uniformly random elements in $\mathbb{Z}_q$ for each $i = 1, ..., M$ and define $z_i = k_i - h_i \cdot sk \bmod q$.
  Suppose some fixed distribution χ_b on $\{0,1\}^b$ for $b > 0$ and define a probabilistic algorithm $EMSB_{\chi_b}(x) = MSB_{\chi_b}(x) \oplus \epsilon$ for some error bit string ϵ sampled from χ_b.
  Given (h_i, z_i) and $EMSB_{\chi_b}(k_i)$ for $i = 1, ..., M$,
  the HNP with error distribution χ_b asks one to find sk.
- A straightforward calculation shows that a set of ECDSA signatures with leaky nonces is indeed an instance of the HNP. Indeed, since the ECDSA signature satisfies $s = \frac{H(msg)+r \cdot sk}{k} \bmod q$ for uniformly chosen $k \in \mathbb{Z}_q$, we can rearrange the terms as $\frac{H(msg)}{s} = k - (r/s) \cdot sk \bmod q$. In this way, we obtain a HNP sample if the MSB of k is leaked with some probability.

Bleichenbacher's Attack Framework (premises)

- The **Fourier analysis-based approach** to the HNP was first proposed by Bleichenbacher and it has been used to break ECDSA with small nonce leakages that are hard to exploit with the lattice-based method.
- The essential idea of the method is to quantify the **modular bias of nonce k** using the bias functions in the form of inverse discrete Fourier transform.
- Let K be a random variable over $\mathbb{Z}_q$.

  The modular bias $B_q(K)$ is defined as $B_q(K) = E[e^{(\frac{2\pi K}{q})i}]$,
  while the sampled bias of a set of points $K = \{k_i\}_{i=1}^{M}$ in $\mathbb{Z}_q$ is defined by $B_q(K) = \frac{1}{M}\sum_{i=1}^{M} e^{(\frac{2\pi k_i}{q})i}$.
- If the k_i's follow the uniform distribution over $\mathbb{Z}_q$, then the mean of the norm of sampled bias is estimated as $\frac{1}{\sqrt{M}}$.

Bleichenbacher's Attack Framework (consequences)

- From the premises, it would be straightforward to retrieve the secret value sk. For each candidate secret key $w \in \mathbb{Z}_q$, compute the corresponding set of candidate nonces $K_w = \{z_i + h_i w \bmod q\}_{i=1}^{M}$ and then conclude w = sk if the sampled bias $|B_q(K)|$ shows a peak value.
- This because, as De Mulder et al. explain in their paper, we wish to find the secret and presumably unique w for which the set of candidate nonces K_w all fall near 0 or q, while for any value different from w, the set of values would show a relatively small sampled bias. So, in the former case we would have $B_q(w)$ close to 1, while in the latter one the modular bias would show a value closer to 0.
- However, since it would be no better than the exhaustive search over the entire $\mathbb{Z}_q$, the so-called collision search of input samples allows to expand the peak width.

Collision search goals

- The first goal of the collision search phase is to find sufficiently **small linear combinations** of the samples so that the FFT on the table of size L_{FFT} becomes practically computable.
  - This result can be obtained by taking linear combinations of input samples $\{(h_i, z_i)\}_{i=1}^{M}$,to generate new samples $\{(h'_j, z'_j)\}_{i=1}^{M'}$ such that $h'_j < L_{FFT}$, broadening the peak to approximately $\frac{q}{L_{FFT}}$. In this way, the broadened peak can be hit by only checking the sampled biases at L_{FFT} candidate points over $\mathbb{Z}_q$.
- The second goal of the collision search phase is the **sparsity of the linear combinations**. Indeed, the collision search operation can't be computed to the bitter end, because it has a price: in exchange of the broader peak width, the peak height gets reduced exponentially.
  - This can result in a complication, since the peak height should be significantly larger the noise value (which is $\frac{1}{\sqrt{M'}}$ on average, as mentioned before).

Timing attacks on Montgomery Ladder

Montgomery Ladder Algorithm

Given the scalar $k \in \mathbb{Z}_q$ and the point P on $E(\mathbb{F})$, it ensures the results of the scalar multiplication $R = [k]P$.

1: $(R_0, R_1) \leftarrow (P, 2P)$
2: for $i = \lfloor lg(k) \rfloor - 1$ downto 0 do
3: if $k_i = 0$ then
4: $(R_0, R_1) \leftarrow ([2]R_0, R_0 \oplus R_1)$
5: else
6: $(R_0, R_1) \leftarrow (R_0 \oplus R_1, [2]R_1)$
7: endif
8: endfor
9: return R_0

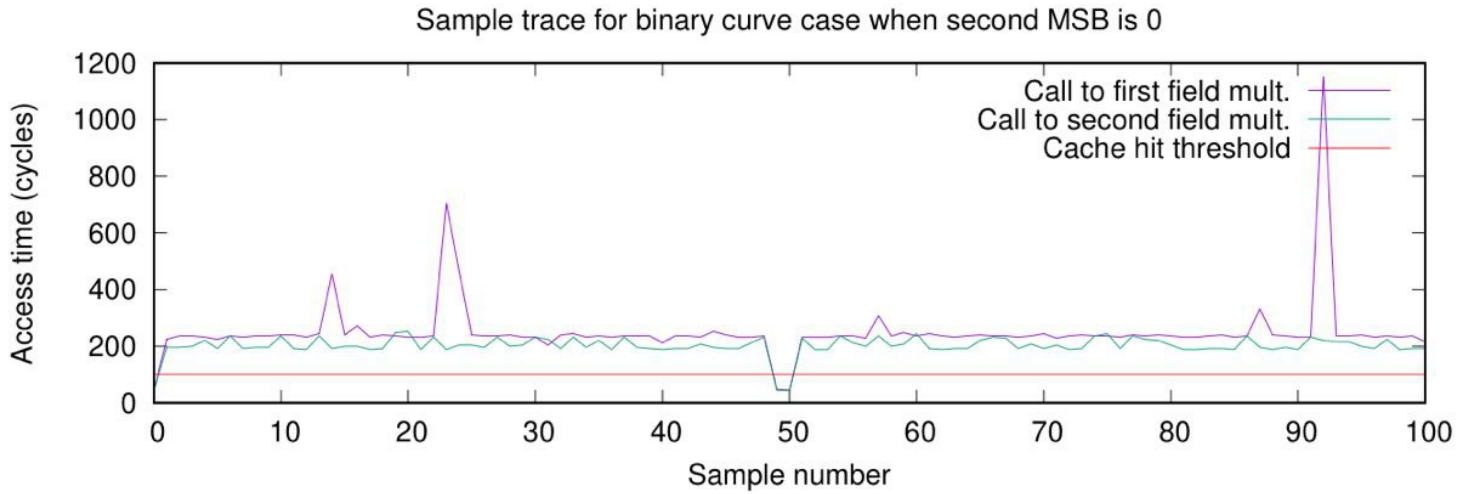
Montgomery Ladder Algorithm Exploitation

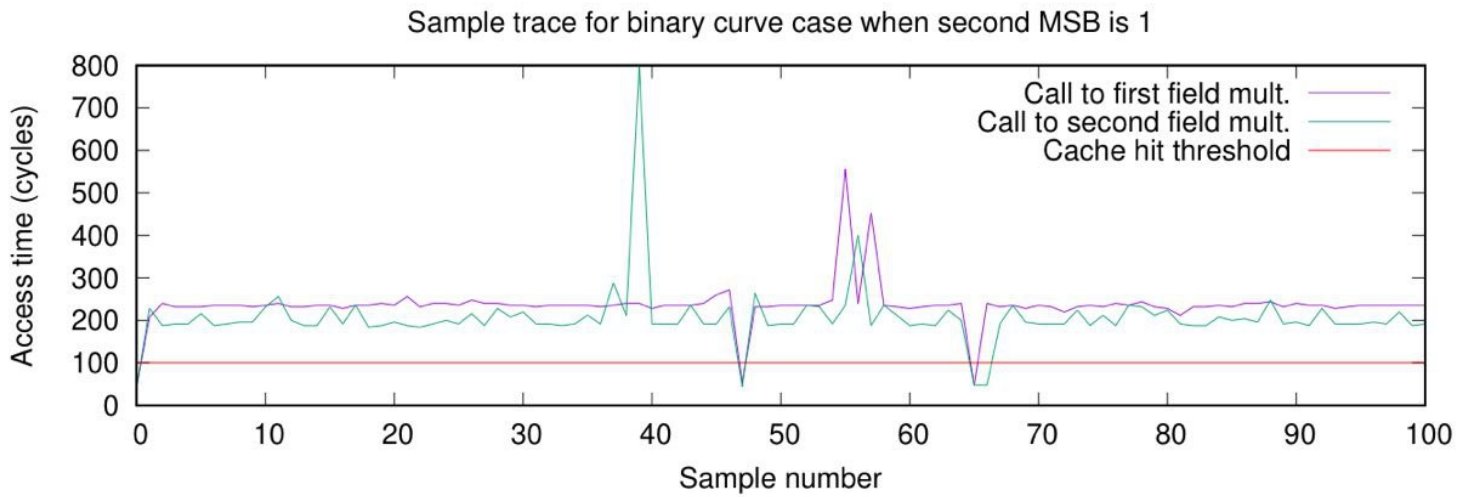
- The Montgomery Ladder Algorithm must be built on top on a constant-time implementation in order to guarantee a strong side-channel resistance.
  - Indeed, any minor deviation from this implementation could leak information about which of the two branches of a certain iteration are being evaluated, leaking in this way information about the key bit.
- This work exploited a vulnerability in the line 1 of the algorithm, which consists in a **coordinate mismatch**. Indeed, the implementation that employs projective coordinates will have accumulator R_0 in affine coordinates and R_1 in projective coordinates. This allows the attacker to mount a cache-timing attack, revealing the second MSB of the scalar.
- The implementations of the attack in this work have employed the different implementations of the Montgomery Ladder in the codebase of OpenSSL, differentiating between binary and prime curves.

Montgomery Ladder Algorithm Exploitation - Binary Curves Case

- For curves defined over $\mathbb{F}_{2^m}$, the Montgomery Ladder starts by initializing two points $(X_1, Z_1) = (x, 1)$ and $(X_2, Z_2) = [2]P = (x^4 + b, x^2)$.
- After a conditional swap function that exchanges the two points based on the value of the second MSB, the first function called within the first iteration, employed for point addition, starts by multiplying by value Z_1.
- However, the finite field arithmetic is not implemented in constant-time for binary fields. Therefore, there is a timing difference between multiplying by (1) or (x^2), since the latter case would require modular reduction.
  Since it would happen only after the conditional swap (i.e. when the second MSB is 1), a cache-timing attack can monitor when the modular reduction code is called, knowing then which is the current case.

Montgomery Ladder Algorithm Exploitation - Binary Curves Case

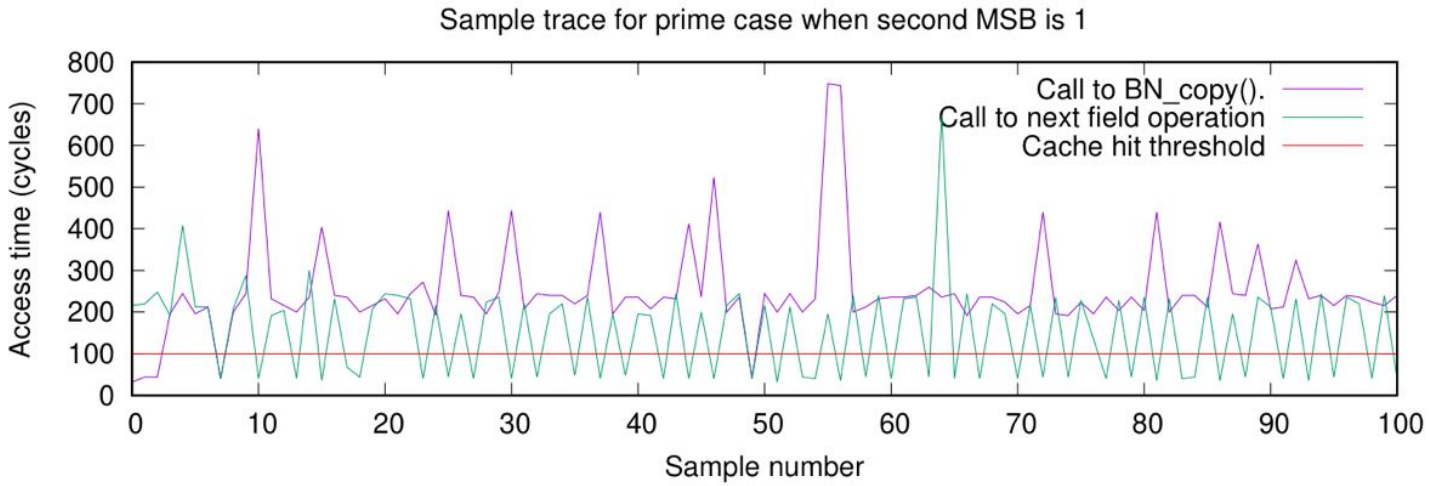


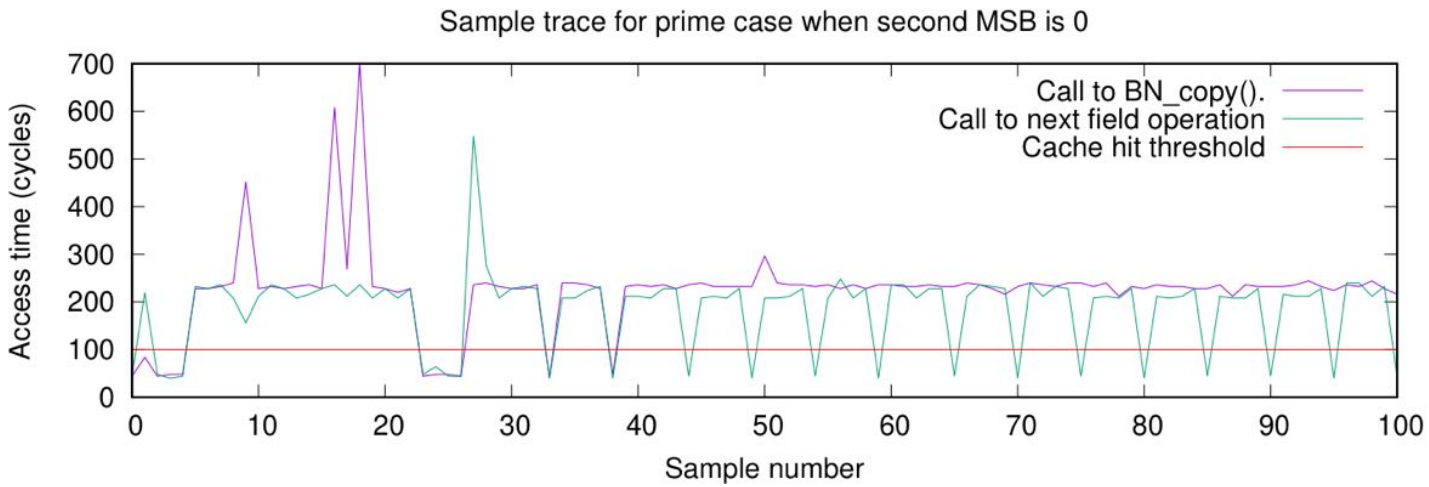


Montgomery Ladder Algorithm Exploitation - Prime Curves Case

- In curves defined over $\mathbb{F}_p$ for large prime p, OpenSSL implements the Montgomery ladder by using optimized formulas for elliptic curve arithmetic in the Weierstrass model.
- The ladder starts by initializing two accumulators $R_0 = P$ (in affine coordinates) and $R_1 = 2P$ (in projective coordinates).
  After a conditional swap based on the key bit, the first loop iteration computes a point addition and a point doubling.
  When the input point for the doubling function is in affine coordinates (i.e. when the two accumulators are not swapped), a field multiplication by Z is replaced by a faster call to the function $BN\_Copy()$.
  This happens when the two accumulators are not swapped in the ladder, which means that the second MSB is 0, and it can be detected with a cache-timing attack.

Montgomery Ladder Algorithm Exploitation - Prime Curves Case





Implementation of the attacks

- The cache-timing attacks were implemented using the Flush+Reload technique. Since the observed timing difference was very small in both cases, it was amplified using **performance degradation**, which consisted in make multiple threads run in the background penalizing the targeted pieces of code, by constantly evicting their addresses from the cache.
  - In the binary case the target parameter was the curve sect163r1, where the amplification due to performance degradation reached around 100,000 cycles with a precision of 99.00% of detecting the second MSB.
  - In the prime case the target parameter was the curve P-192/secp192k1, where the amplification due to performance degradation reached around 15,000 cycles with a precision of 99.53% of detecting the second MSB.

Performance Degradation technique

- Allan et al. examine in detail how the **performance degradation method** is able to amplify side-channel attacks, choosing as target the ECDSA signature algorithm itself.
- Indeed, **sharing hardware resources** between different clients can help to achieve higher resource utilization. On the other hand, it paves the way for interferences between different clients, which could be exploited to harm a chosen victim as it is performed in this paper, against the scalar multiplication code employed for ECDSA.
- The performance degradation attack consists in **evicting the frequently executed code** of the victim from the processor cache, which forces the victim to reload the evicted code from the memory, introducing significant delays to the process. This repeated action would negate the performance benefits of the cache, slowing the victim down by a factor of over 150.

Improved Analysis of Bleichenbacher's attack

Unified time-space-data tradeoff

- As we mentioned, Bleichenbacher's Fourier analysis-based attack can tackle small nonce biases, being at the same time able to handle erroneous inputs.
- However, the FFT-based approach to the HNP was thought to require 8 billions of signatures as input to attack 1-bit of nonce leakage, which is not practically easy to collect.
- Nevertheless, this work proved that it is possible to significantly reduce the data complexity by carefully choosing the inputs to a given tradeoff formula.
  - This brings the side-channel attacker to recover the ECDSA key given only thousands (sometimes even hundreds) signatures in many cases.

K-list Sum Problem

- The ***K*-list sum problem** is a sub-problem of the Generalized Birthday Problem which is considered in order to instantiate the trade-off mentioned above.
  - Given *K* sorted list $L_1, ..., L_K$, each of which consists of 2^a uniformly random l-bit integers, the *K*-list problem asks one to find a non-empty list *L'* consisting of $x' = \sum_{i=1}^{K} \omega_i x_i$, where *K*-tuples $x_1, ..., x_K \in L_1 \times ... \times L_K$ and $(\omega_1, ..., \omega_K) \in \{-1, 0, 1\}^K$ satisfy $MSB_n(x') = 0$ for some target parameter $n \leq l$.
- There can be a practical situation where the adversary may want to trade the "online" side-channel detection cost (i.e. data complexity) for the "offline" resources required by Bleichenbacher's attack (i.e. time and space complexities).
  - In order to get the tradeoff formula, we analyze the instance of the *K*-list sum problem for $K = 4$.

4-list Sum Problem tradeoff formula

This work generalizes the tradeoff formula presented by Dinur in its paper about the Generalized Birthday Problem, for which the following tradeoff holds:

$$2^4 M' N = T M^2$$

or put differently

$$m' = 3a + v - n.$$

Where each parameter is defined as follows:

- $N = 2^n$, where n is the number of bits to be nullified;
- $M = 2^m = 4 \times 2^a$ is the number of input samples, where 2^a is the lenght of each sublist;
- $M' = 2^{m'}$ is the number of output samples s.t. the top n bits are 0;
- $v \in [0, a]$ is a parameter deciding how many iterations of the collision search to be executed;
- $T = 2^t = 2^{a+v}$ is the time complexity.

Bias Function in Presence of Misdetection

- As we mentioned, the strength of the techniques disclosed in this work also consists in **being resistant to errors** in the MSB information of HNP input samples (i.e. $\epsilon > 0$).
  It is a concrete case, since the side-channel detection is not 100 percent accurate. Therefore, it is showed how to calculate biases when there are ϵ errors in the input.
  - Note that the extreme case where $\epsilon = \frac{1}{2}$ simply means that the samples are not biased at all, and therefore the bias $|B_q(K)|$ degenerates to 0. This means that the attacker gains no side-channel information about nonces, making theoretically impossible to solve the HNP.
- For $b \in \{0, 1\}$, any $\epsilon \in [0, \frac{1}{2}]$ and even integer $q > 0$ the following holds.
  Let K be a random variable following the weighted uniform distribution over $\mathbb{Z}_q$ below.

$$Pr[K = k_i] = (1 - b) \cdot \frac{1-\epsilon}{q/2} + b \cdot \frac{\epsilon}{q/2} \text{ if } 0 \leq k_i < \frac{q}{2}$$

$$Pr[K = k_i] = b \cdot \frac{1-\epsilon}{q/2} + (1 - b) \cdot \frac{\epsilon}{q/2} \text{ if } \frac{q}{2} \leq k_i < q$$

  Then the modular bias of K is $B_q(K) = (1 - 2\epsilon)B_q(K_b)$,
  where K_b follows the uniform distribution over $[0 + \frac{bq}{2}, \frac{q}{2} + \frac{bq}{2})$

Conclusions

Software countermeasures

The main countermeasure to defend against the attack presented in this work is **enforcing constant-time behavior** in the implementation of scalar multiplication. There are three suggested options:

- 1. Randomization of Z-coordinates: it is the easiest solution to implement and it would guarantee all intermediate points in projective coordinates. Additional care must be taken when converting from projective to affine coordinates;
- 2. Satisfying constant-time guarantees;
- 3. Employing alternative scalar multiplication algorithms which don't penalize performance as much.

References

- Aranha, D.F., Novaes, F.R., Takahashi, A., Tibouchi, M., Yarom, Y.: LadderLeak: breaking ECDSA with less than one bit of nonce leakage. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, pp. 225–242 (2020);
- Thomas Allan, Billy Bob Brumley, Katrina E. Falkner, Joop van de Pol, and Yuval Yarom. 2016. Amplifying side channels through performance degradation. In ACSAC. 422–435.
- Elke De Mulder, Michael Hutter, Mark E. Marson, and Peter Pearson. 2013. Using Bleichenbacher's Solution to the Hidden Number Problem to Attack Nonce Leaks in 384-Bit ECDSA. In CHES 2013 (LNCS, Vol. 8086), Guido Bertoni and JeanSébastien Coron (Eds.). Springer, Heidelberg, 435–452. https://doi.org/10.1007/978-3-642-40349-1_25
- David Wagner. 2002. A Generalized Birthday Problem. In CRYPTO 2002 (LNCS, Vol. 2442), Moti Yung (Ed.). Springer, Heidelberg, 288–303. https://doi.org/10.1007/3-540-45708-9_19
- Itai Dinur. 2019. An algorithmic framework for the generalized birthday problem. Des. Codes Cryptogr. 87, 8 (2019), 1897–1926. https://doi.org/10.1007/s10623-018-00594-6