LadderLeak

Breaking ECDSA with Less than One Bit of Nonce Leakage

Black Hat Europe (also CCS'20 and ePrint: 2020/615)

Diego F. Aranha[1] Felipe R. Novaes[2] Akira Takahashi[1] Mehdi Tibouchi[3] Yuval Yarom[4]

[1]DIGIT, Aarhus University, Denmark

[2]University of Campinas, Brazil

[3]NTT Corporation, Japan

[4]University of Adelaide and Data61, Australia

This talk

New attacks on randomness leakage/bias from ECDSA/Schnorr-type schemes

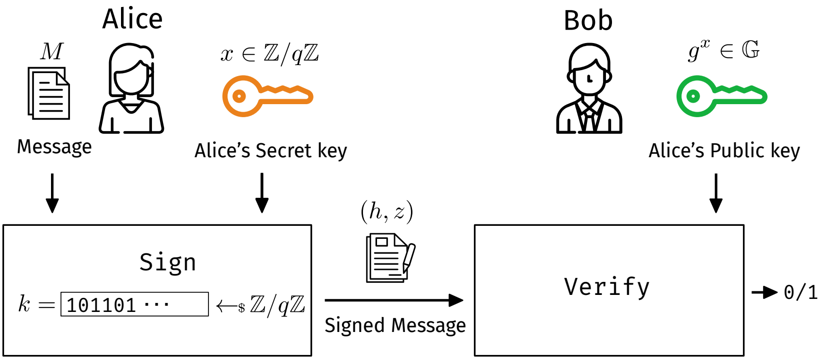
- Discovered vulnerabilities in ECDSA implementations: **OpenSSL** and **RELIC**.
- Theoretical improvements to the attack framework on the **Hidden Number Problem (HNP)**.
- Part I: How to **acquire** side-channel information.
- Part II: How to **exploit** side-channel information to recover the secret key.

Background: Attack on ECDSA Nonces

ECDSA and Schnorr Signatures

- Most popular signature schemes relying on the hardness of the (EC)DLP
- Signing operation involves **secret** randomness $k \in \mathbb{Z}/q\mathbb{Z}$, sometimes called **nonce**

Randomness in ECDSA/Schnorr-like Schemes

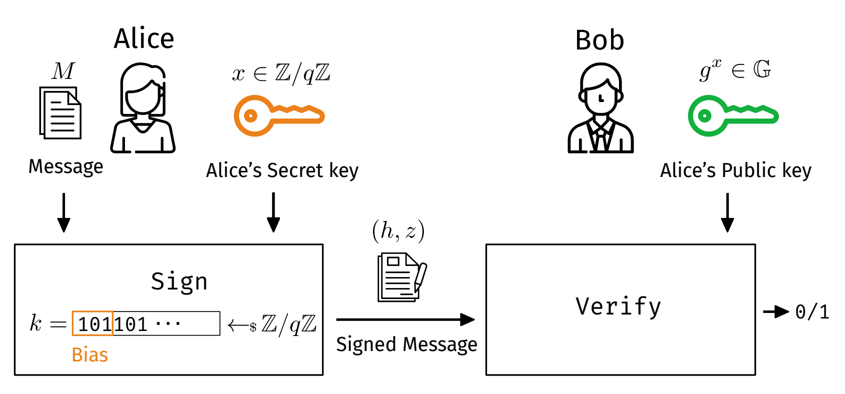


- k is a uniformly random value satisfying

$$k \equiv \underbrace{z}_{\text{public}} + \underbrace{h}_{\text{public}} \cdot x \mod q.$$

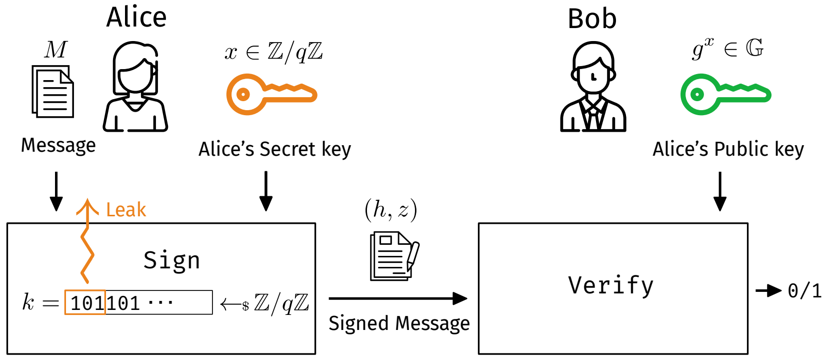
- k should **NEVER** be reused/exposed as $x = (z - z')/(h' - h) \mod q$

Risk of Biased/Leaky Randomness



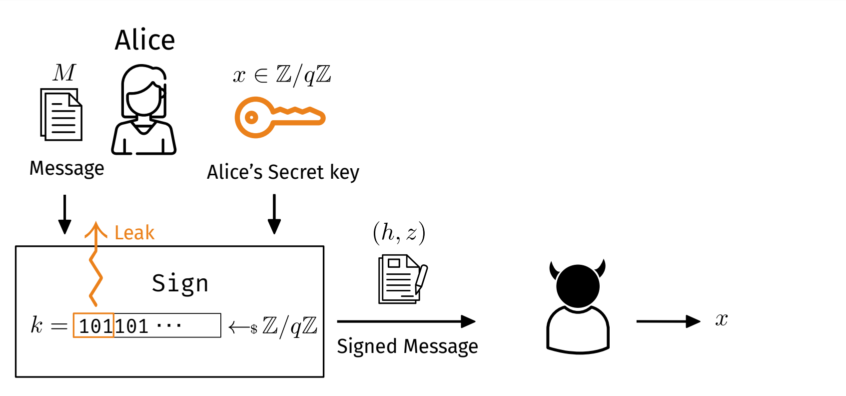
- What if k is slightly biased ?
- Secret key x is recovered by solving the hidden number problem (HNP)

Risk of Biased/Leaky Randomness



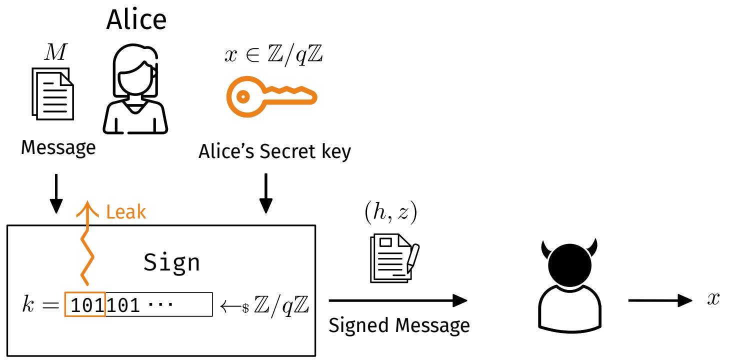
- What if k is slightly biased or partially leaked?
- Secret key x is recovered by solving the hidden number problem (HNP)

Risk of Biased/Leaky Randomness



- What if k is slightly biased or partially leaked? $\rightsquigarrow$ Attack!
- Secret key x is recovered by solving the hidden number problem (HNP)

Risk of Biased/Leaky Randomness



- What if k is slightly biased or partially leaked? $\rightsquigarrow$ Attack!
- Secret key x is recovered by solving the hidden number problem (HNP)

Randomness Failure in the Real World

- Poorly designed/implemented RNGs.
- Predictable seed (`srand(time(0))`).
- VM resets ⇝ same snapshot will end up with the same seed.
- Side-channel leakage.
- and many more...


BBC | Sign in | News | Sport | Weather | Shop | Earth | Travel | Mo

NEWS

Home | UK | World | Business | Politics | Tech | Science | Health | Family & Education | E

Technology

iPhone hacker publishes secret Sony PlayStation 3 key

By Jonathan Fildes
Technology reporter, BBC News

6 January 2011

Share

The PlayStation 3's security has been broken by hackers, potentially allowing anyone to run any software - including pirated games - on the console.

A collective of hackers recently showed off a method that could force the system to reveal secret keys used to load

BBC news. 2011. https://www.bbc.com/news/technology-12116051

Contributions

1. Novel class of cache attacks against ECDSA implemented in OpenSSL `1.0.2u` and `1.1.0l`, and RELIC 0.4.0.

   Affected curves: NIST P-192, P-224, P-256, P-384, P-521, B-283, K-283, K-409, B-571, `sect163r1`, `secp192k1`, `secp256k1`

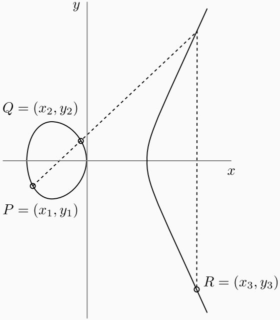
   Affected products: VMWare Photon, Chef, Wickr ?

2. Theoretical improvements to Fourier analysis-based attack on the HNP
   - Significantly reduced the required input data
   - Attack became feasible given **less than 1-bit of nonce bias/leakage** per signature

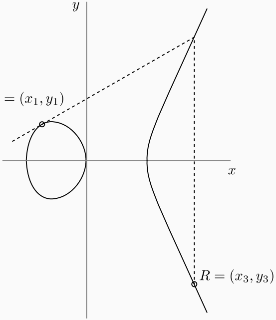
3. Implemented a full secret key recovery attack against OpenSSL ECDSA over `sect163r1` and NIST P-192.

Curve-based cryptography

Elliptic curves



(a) Point addition $R = P \oplus Q$



(b) Point doubling $R = [2]P$

Group law: Points form an additive group under the operation $\oplus$ (chord and tangent) of **order** q with ∞ as the identity.

Coordinate system: For efficiency, we represent a point in **affine coordinates** (x, y) using **projective coordinates** (X, Y, Z) such that $x = X/Z^c$ and $y = Y/Z^d$.

ECDSA signing

Scalar multiplication is critical for performance/security of ECC.

Algorithm 1 ECDSA signature generation

Input: Signing key $sk \in \mathbb{Z}_q$, message $\texttt{msg} \in \{0,1\}^*$, group order q, base point G, and cryptographic hash function $H : 0, 1^* \to \mathbb{Z}_q$.

Output: A valid signature (r, s)

1: $k \leftarrow_\$ \mathbb{Z}_q^*$

2: $R = (r_x, r_y) \leftarrow [k]G$

3: $r \leftarrow r_x \mod q$

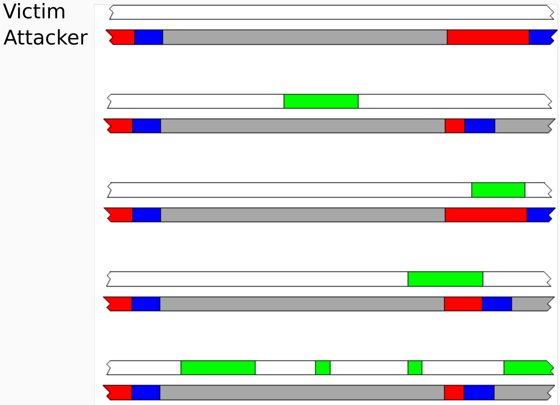
4: $s \leftarrow (H(\texttt{msg}) + r \cdot sk)/k \mod q$

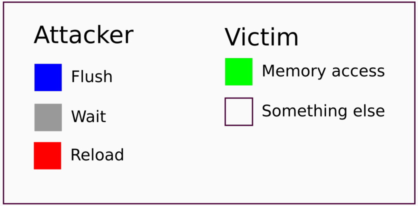
5: **return** (r, s)

Critical: Should be implemented in **constant time** to avoid timing leakage about k.

Cache-timing attacks

Modern CPUs have instructions (`cflush`) that can reveal **secrets** through cache data eviction. When programs share a library, a Flush+Reload attack is possible:





Algorithm 2 Left-to-right Montgomery ladder

Input: $P = (x, y)$, $k = (1, k_{t-2}, \dots, k_1, k_0)$

Output: $Q = [k]P$

```
R_0 ← P, R_1 ← [2]P
for i ← t − 2 downto 0 do
    if k_i ← 1 then
        R_0 ← R_0 ⊕ R_1; R_1 ← [2]R_1
    else
        R_1 ← R_0 ⊕ R_1; R_0 ← [2]R_0
    end if
end for
return Q = R_0
```

For constant-time:

- Fixed number of iterations
- Accumulators R_i in the same order.
- Group law is implemented in constant time.

Side-channel attacks in scalar multiplication

Algorithm 3 Left-to-right Montgomery ladder

Input: $P = (x, y)$, $k = (1, k_{t-2}, \ldots, k_1, k_0)$

Output: $Q = [k]P$

$k' \leftarrow$ Select $(k + q, k + 2q)$
$R_0 \leftarrow P$, $R_1 \leftarrow [2]P$
for $i \leftarrow \lg(q) - 1$ **downto** 0 **do**
    Swap (R_0, R_1) if $k'_i = 0$
    $R_0 \leftarrow R_0 \oplus R_1$; $R_1 \leftarrow [2]R_1$
    Swap (R_0, R_1) if $k'_i = 0$
end for
return $Q = R_0$

For constant-time:

- Fixed iterations by **adding 1 or 2 multiples** of q (preserves MSB of k in second MSB of k' when q is just below power of 2.
- Replace branch with **conditional swap** (ideally implemented in ASM).
- **Careful** implementation of group law!

Side-channel attacks in scalar multiplication

Algorithm 4 Left-to-right Montgomery ladder

Input: $P = (x, y)$, $k = (1, k_{t-2}, \ldots, k_1, k_0)$

Output: $Q = [k]P$

$k' \leftarrow$ Select $(k + q, k + 2q)$

$R_0 \leftarrow P$, $R_1 \leftarrow [2]P$

for $i \leftarrow \lg(q) - 1$ **downto** 0 **do**

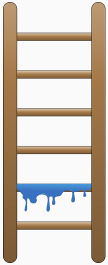
  Swap (R_0, R_1) if $k'_i = 0$

  $R_0 \leftarrow R_0 \oplus R_1$; $R_1 \leftarrow 2R_1$

  Swap (R_0, R_1) if $k'_i = 0$

end for

return $Q = R_0$



Critical: Leakage in k allows to build set of **biased** signatures.

Experimental setup

Target platforms:

- Broadwell CPUs (Core i7-5500U @ 2.4GHz and i7-3520M @ 2.9GHz)
- TurboBoost **disabled** for reducing noise
- Binaries executed in userland runtime, **no privileges**
- OpenSSL built using default configuration, debugging symbols

Tooling:

- `FR-Trace` from Mastik side-channel analysis toolkit
- Flush+Reload **slot** selected as the 5,000 cycles
- Other cores evict code from cache (**performance degradation**)

Cache-timing attacks on prime curves

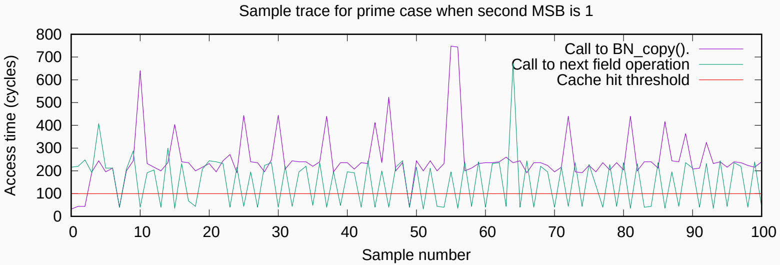
We can detect if R_1 is in affine coordinates in point doubling ($k_i' = 0$).

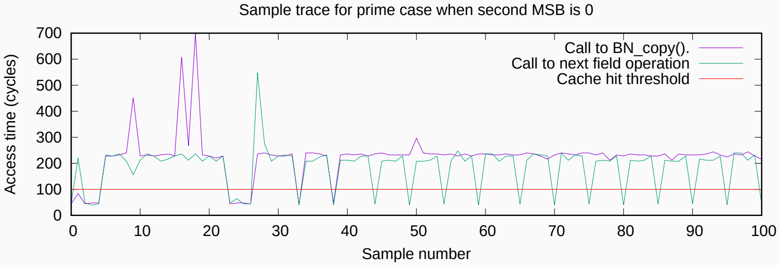
```
    (...)
    if (a->Z_is_one) {
        if (!BN_copy(n0, &a->Y))
            goto err;
    } else {
        if (!field_mul(group, n0, &a->Y, &a->Z, ctx))
            goto err;
    }
    (...)
```

Performance degradation can amplify the difference to $\approx$ 15,000 cycles.

Attack: Flush+Reload can detect if `BN_copy()` is called with $>$ 99% precision.

Cache-timing attacks on prime curves





Cache-timing attacks on binary curves

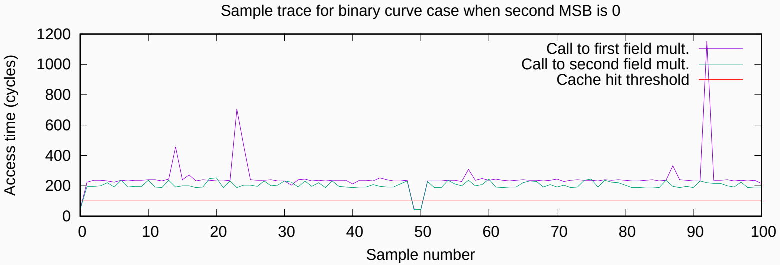
We can detect if R_1 has projective coordinates in point addition ($k'_i = 1$).

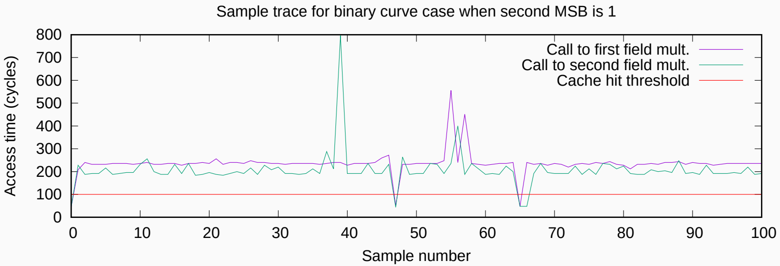
```
(...)
if (!BN_copy(t1, x))
        goto err;
    if (!group->meth->field_mul(group, x1, x1, z2, ctx))
        goto err;
    if (!group->meth->field_mul(group, z1, z1, x2, ctx))
        goto err;
(...)
```

Performance degradation can amplify difference to $\approx$ 100,000 cycles.

Attack: Flush+Reload can detect if `z2`$= 1$ with $>$ 99% precision.

Cache-timing attacks on binary curves





Software countermeasures

There are **at least** three possible fixes:

1. **Randomize** Z coordinates at the beginning of scalar multiplication.
2. Implement group law in constant time, for example using **complete addition formulas** (no branches).
3. Implement ladder over co-Z arithmetic to **not handle** Z directly.

Coordinated disclosure: reported in December 2019, fixed in April 2020 with the first countermeasure.

Main takeaways

- Securely implementing brittle cryptographic algorithms is still **hard**.
- Do not underestimate timing leakage without careful analysis, even if **tiny**.
- **Upgrade** OpenSSL to 1.1.1 (or 3.0 when available) as soon as possible!

How to Exploit Nonce Leakage

Overview

- Recover the ECDSA secret by solving the **hidden number problem (HNP)** [BV96]
- **Fourier analysis-based attack** (Bleichenbacher ’00)
    - Allows us to recover the secret using only **1-bit** of nonce info per signature.
    - Analysis considers side-channel attacker’s **misdetection of nonce bits**
    - The techniques in principle apply to other sources of bias/leakage

The problem we tackle

Definition (Hidden Number Problem)

Let h_i and k_i be uniformly random elements in $\mathbb{Z}_q$ for each $i = 1, \ldots, M$ and

$$z_i = k_i - h_i \cdot sk \mod q.$$

The HNP asks to find sk, given the pairs (h_i, z_i) and $\mathtt{MSB}_\ell(k_i)$ for all i (the ℓ most significant bits of k_i).

The problem we tackle

Definition (Hidden Number Problem)

Let h_i and k_i be uniformly random elements in $\mathbb{Z}_q$ for each $i = 1, \ldots, M$ and

$$z_i = k_i - h_i \cdot sk \mod q.$$

The HNP asks to find sk, given the pairs (h_i, z_i) and $\mathtt{MSB}_\ell(k_i)$ for all i (the ℓ most significant bits of k_i).

* (h_i, z_i) can be computed from ECDSA signature:

$$h_i = r/s \pmod q$$
$$z_i = H(\mathtt{msg})/s \pmod q$$

Chronology of HNP: a 24-year retrospective

1996 Boneh–Venkatesan defined the HNP

1999 Howgrave-Graham–Smart proposed the lattice attack against HNP

2000 Bleichenbacher announced the Fourier analysis attack

⋮

2018 CacheQuote on SGX EPID; PortSmash on SMT/Hyper-Threading; ROHNP

2019 TPM-FAIL; Minerva

2020 Dé jà Vu attack on Mozilla's NSS; Raccoon attack on TLS 1.2

Still at the heart of **many** recent real-world vulnerabilities in ECDSA/Diffie–Hellman key exchange implementations!

Chronology of HNP: a 24-year retrospective

1996 Boneh–Venkatesan defined the HNP

1999 Howgrave-Graham–Smart proposed the lattice attack against HNP

2000 Bleichenbacher announced the Fourier analysis attack

⋮

2018 CacheQuote on SGX EPID; PortSmash on SMT/Hyper-Threading; ROHNP

2019 TPM-FAIL; Minerva

2020 Dé jà Vu attack on Mozilla's NSS; Raccoon attack on TLS 1.2

Still at the heart of **many** recent real-world vulnerabilities in ECDSA/Diffie–Hellman key exchange implementations!

Chronology of HNP: a 24-year retrospective

1996 Boneh–Venkatesan defined the HNP

1999 Howgrave-Graham–Smart proposed the lattice attack against HNP

2000 Bleichenbacher announced the Fourier analysis attack

⋮

2018 CacheQuote on SGX EPID; PortSmash on SMT/Hyper-Threading; ROHNP

2019 TPM-FAIL; Minerva

2020 Dé jà Vu attack on Mozilla's NSS; Raccoon attack on TLS 1.2

Still at the heart of **many** recent real-world vulnerabilities in ECDSA/Diffie–Hellman key exchange implementations!

Chronology of HNP: a 24-year retrospective

1996 Boneh–Venkatesan defined the HNP

1999 Howgrave-Graham–Smart proposed the lattice attack against HNP

2000 Bleichenbacher announced the Fourier analysis attack

⋮

2018 CacheQuote on SGX EPID; PortSmash on SMT/Hyper-Threading; ROHNP

2019 TPM-FAIL; Minerva

2020 Dé jà Vu attack on Mozilla's NSS; Raccoon attack on TLS 1.2

Still at the heart of **many** recent real-world vulnerabilities in ECDSA/Diffie–Hellman key exchange implementations!

Chronology of HNP: a 24-year retrospective

1996 Boneh–Venkatesan defined the HNP

1999 Howgrave-Graham–Smart proposed the lattice attack against HNP

2000 Bleichenbacher announced the Fourier analysis attack

⋮

2018 CacheQuote on SGX EPID; PortSmash on SMT/Hyper-Threading; ROHNP

2019 TPM-FAIL; Minerva

2020 Dé jà Vu attack on Mozilla's NSS; Raccoon attack on TLS 1.2

Still at the heart of **many** recent real-world vulnerabilities in ECDSA/Diffie–Hellman key exchange implementations!

Chronology of HNP: a 24-year retrospective

1996 Boneh–Venkatesan defined the HNP

1999 Howgrave-Graham–Smart proposed the lattice attack against HNP

2000 Bleichenbacher announced the Fourier analysis attack

⋮

2018 CacheQuote on SGX EPID; PortSmash on SMT/Hyper-Threading; ROHNP

2019 TPM-FAIL; Minerva

2020 Dé jà Vu attack on Mozilla's NSS; Raccoon attack on TLS 1.2

Still at the heart of **many** recent real-world vulnerabilities in ECDSA/Diffie–Hellman key exchange implementations!

Chronology of HNP: a 24-year retrospective

1996 Boneh–Venkatesan defined the HNP

1999 Howgrave-Graham–Smart proposed the lattice attack against HNP

2000 Bleichenbacher announced the Fourier analysis attack

⋮

2018 CacheQuote on SGX EPID; PortSmash on SMT/Hyper-Threading; ROHNP

2019 TPM-FAIL; Minerva

2020 Dé jà Vu attack on Mozilla's NSS; Raccoon attack on TLS 1.2

Still at the heart of **many** recent real-world vulnerabilities in ECDSA/Diffie–Hellman key exchange implementations!

Chronology of HNP: a 24-year retrospective

1996 Boneh–Venkatesan defined the HNP

1999 Howgrave-Graham–Smart proposed the lattice attack against HNP

2000 Bleichenbacher announced the Fourier analysis attack

⋮

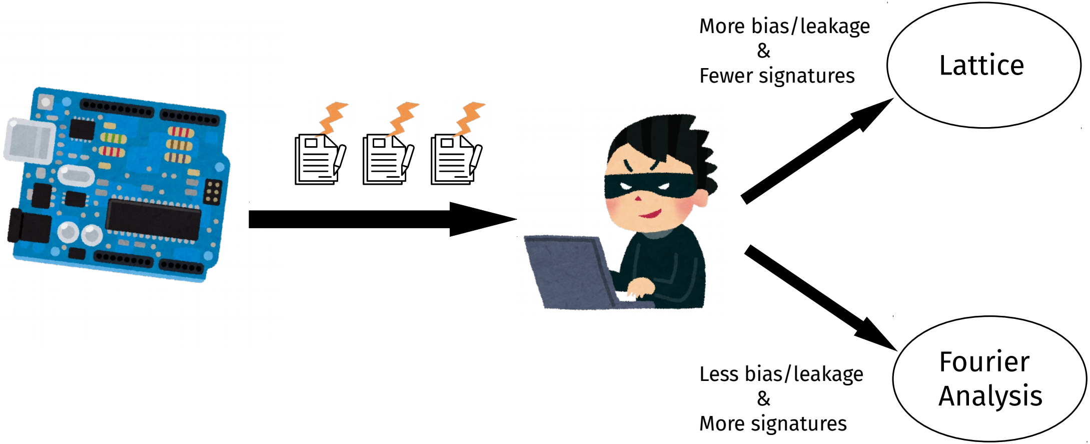
2018 CacheQuote on SGX EPID; PortSmash on SMT/Hyper-Threading; ROHNP

2019 TPM-FAIL; Minerva

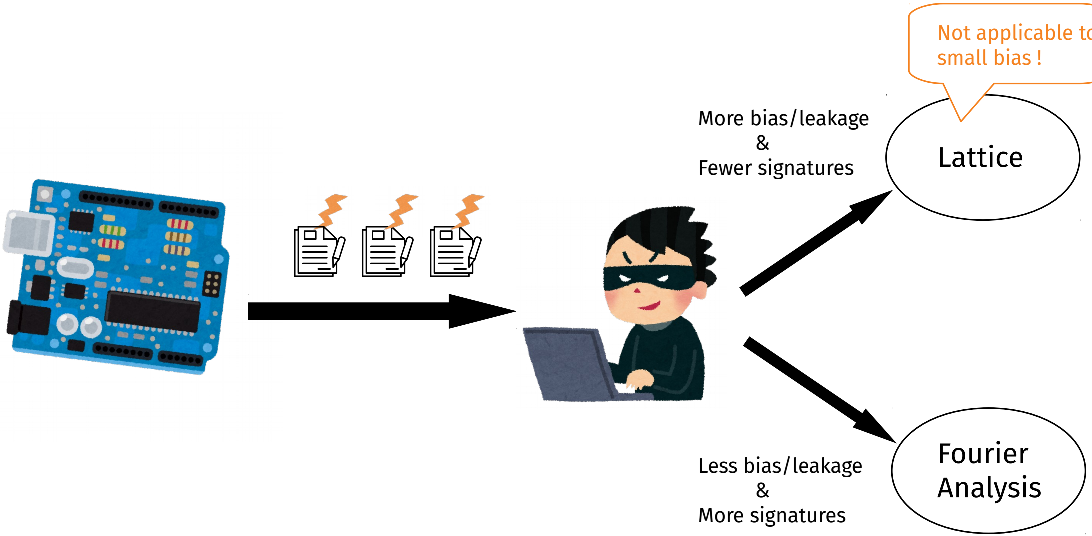
2020 Dé jà Vu attack on Mozilla's NSS; Raccoon attack on TLS 1.2

Still at the heart of **many** recent real-world vulnerabilities in ECDSA/Diffie–Hellman key exchange implementations!

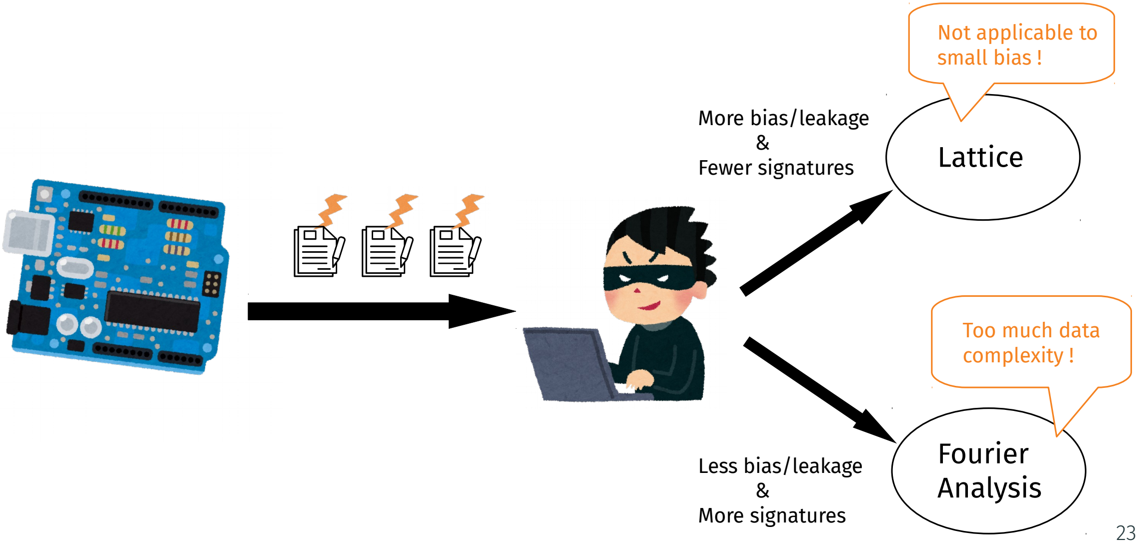
How to solve the HNP: Lattice vs Fourier analysis



How to solve the HNP: Lattice vs Fourier analysis



How to solve the HNP: Lattice vs Fourier analysis



Challenges

- Can we reduce the data complexity of Fourier analysis-based attack?
- Can we attack even **less than 1-bit of nonce leakage** (= MSB is only leaked with prob. < 1)?
- Is there such a small leakage from practical ECDSA implementations?

YES!

Challenges

- Can we reduce the data complexity of Fourier analysis-based attack?
- Can we attack even **less than 1-bit of nonce leakage** (= MSB is only leaked with prob. < 1)?
- Is there such a small leakage from practical ECDSA implementations?

YES!

Challenges

- Can we reduce the data complexity of Fourier analysis-based attack?
- Can we attack even **less than 1-bit of nonce leakage** (= MSB is only leaked with prob. < 1)?
- Is there such a small leakage from practical ECDSA implementations?

YES!

Challenges

- Can we reduce the data complexity of Fourier analysis-based attack?
- Can we attack even **less than 1-bit of nonce leakage** (= MSB is only leaked with prob. < 1)?
- Is there such a small leakage from practical ECDSA implementations?

YES!

New attack records for the HNP!

Comparison with the previous records of solutions to the HNP: Fourier analysis vs Lattice

	< 1	1	2	3	4
256-bit	—	—	[TTA18]	[TTA18]	[Rya18, Rya19, MSEH19, WSBS20]
192-bit	This work	This work	—	—	—
160-bit	This work	This work (less data), [AFG^{+}14, Ble05]	[Ble00][LN13]	[NS02]	—

- Require fewer input signatures to attack 160-bit HNP with 1-bit leak!
- First attack records for 192-bit HNP with (less than) 1-bit leak!

Bleichenbacher’s Fourier Analysis Attack

Bleichenbacher's Attack: High-level Overview

- Step 1. Quantify the bias of nonce $K = \{k_i\}_{i \in \{1,\ldots,M\}}$
  - $\text{Bias}_q(K) \approx 0$ if k is uniform in $\mathbb{Z}_q$
  - $\text{Bias}_q(K) \approx 1$ if k is biased in $\mathbb{Z}_q$
  - Contribution-1 Analyzed the behavior $\text{Bias}_q(K)$ when k's MSB is biased with probability < 1!
- Step 2. Find a candidate secret key which leads to the peak of $\text{Bias}_q(K)$ (by computing FFT)
- Critical intermediate step: collision search of integers h
  - Detect the bias peak correctly and efficiently
  - Contribution-2 Established **unified time-memory-data tradeoffs** by applying $\mathcal{K}$-list sum algorithm for the GBP!

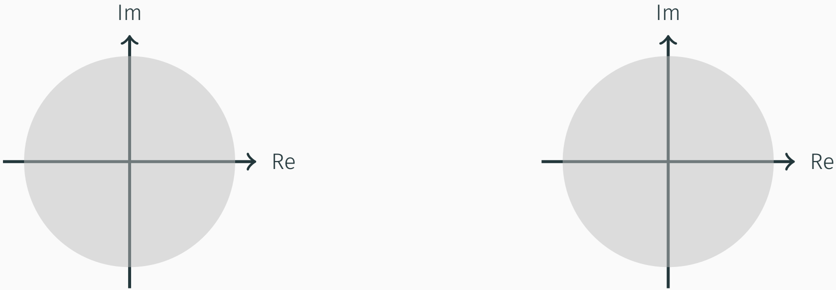
Bleichenbacher's Attack: High-level Overview

- Step 1. Quantify the bias of nonce $K = \{k_i\}_{i \in \{1,\ldots,M\}}$
  - $\mathrm{Bias}_q(K) \approx 0$ if k is uniform in $\mathbb{Z}_q$
  - $\mathrm{Bias}_q(K) \approx 1$ if k is biased in $\mathbb{Z}_q$
  - Contribution-1 Analyzed the behavior $\mathrm{Bias}_q(K)$ when k's MSB is biased with probability < 1!
- Step 2. Find a candidate secret key which leads to the peak of $\mathrm{Bias}_q(K)$ (by computing FFT)
- Critical intermediate step: collision search of integers h
  - Detect the bias peak correctly and efficiently
  - Contribution-2 Established **unified time-memory-data tradeoffs** by applying $\mathcal{K}$-list sum algorithm for the GBP!

Bleichenbacher's Attack: High-level Overview

- Step 1. Quantify the bias of nonce $K = \{k_i\}_{i \in \{1,\dots,M\}}$
  - $\mathrm{Bias}_q(K) \approx 0$ if k is uniform in $\mathbb{Z}_q$
  - $\mathrm{Bias}_q(K) \approx 1$ if k is biased in $\mathbb{Z}_q$
  - Contribution-1 Analyzed the behavior $\mathrm{Bias}_q(K)$ when k's MSB is biased with probability < 1!
- Step 2. Find a candidate secret key which leads to the peak of $\mathrm{Bias}_q(K)$ (by computing FFT)
- Critical intermediate step: collision search of integers h
  - Detect the bias peak correctly and efficiently
  - Contribution-2 Established **unified time-memory-data tradeoffs** by applying $\mathcal{K}$-list sum algorithm for the GBP!

Bias Function (Essentially DFT)

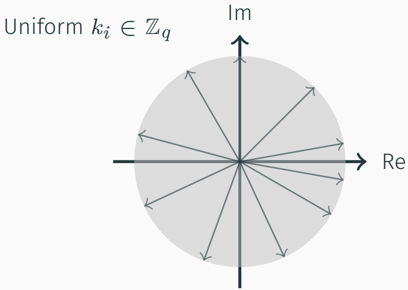


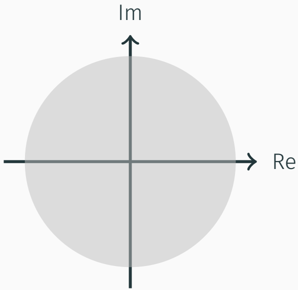
Definition

The **sampled bias** of a set of points $K = \{k_i\}_{i\in\{1,\ldots,M\}}$ in $\mathbb{Z}_q$ is defined by

$$\mathrm{Bias}_q(K) = \frac{1}{M}\sum_{i=1}^{M} e^{2\pi \mathrm{i} k_i/q}.$$

Bias Function (Essentially DFT)





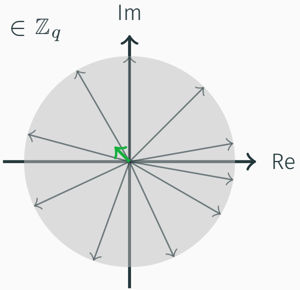
Definition

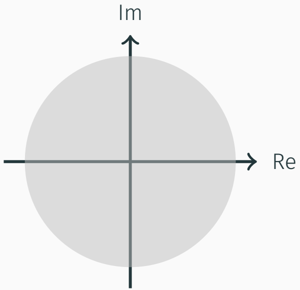
The **sampled bias** of a set of points $K = \{k_i\}_{i \in \{1,\ldots,M\}}$ in $\mathbb{Z}_q$ is defined by

$$\mathrm{Bias}_q(K) = \frac{1}{M} \sum_{i=1}^{M} e^{2\pi \mathrm{i} k_i / q}.$$

Bias Function (Essentially DFT)

Uniform $k_i \in \mathbb{Z}_q$

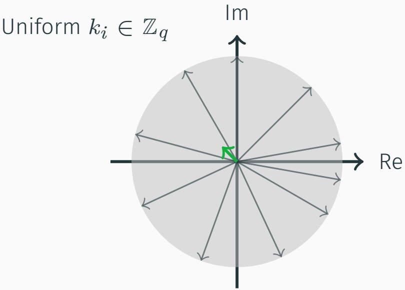


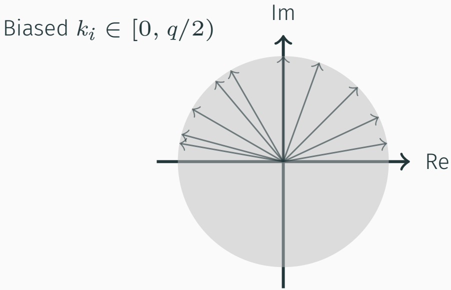


Definition

The **sampled bias** of a set of points $K = \{k_i\}_{i \in \{1,\ldots,M\}}$ in $\mathbb{Z}_q$ is defined by

$$\mathrm{Bias}_q(K) = \frac{1}{M} \sum_{i=1}^{M} e^{2\pi \mathrm{i} k_i / q}.$$



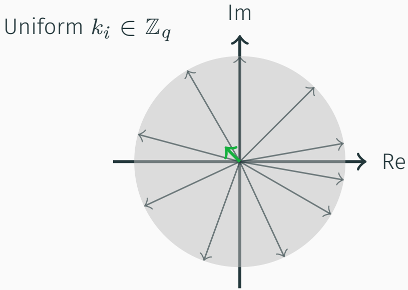


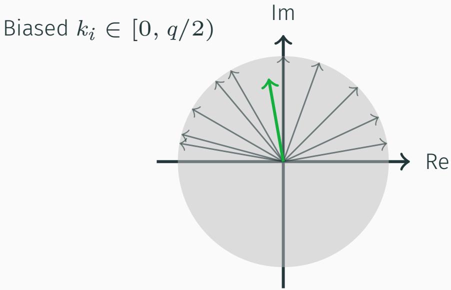
Definition

The **sampled bias** of a set of points $K = \{k_i\}_{i \in \{1,\ldots,M\}}$ in $\mathbb{Z}_q$ is defined by

$$\mathrm{Bias}_q(K) = \frac{1}{M} \sum_{i=1}^{M} e^{2\pi \mathrm{i} k_i / q}.$$

Bias Function (Essentially DFT)





Definition

The **sampled bias** of a set of points $K = \{k_i\}_{i \in \{1,\ldots,M\}}$ in $\mathbb{Z}_q$ is defined by

$$\mathrm{Bias}_q(K) = \frac{1}{M} \sum_{i=1}^{M} e^{2\pi \mathrm{i} k_i / q}.$$

Analyzing misdetection of nonce bits

When the MSB of k_i is leaked, then the attacker can collect biased signatures

$$
\begin{aligned}
k_1 &= 011101\ldots \\
k_2 &= 001010\ldots \\
k_3 &= 010110\ldots \\
k_4 &= 000011\ldots \\
&\vdots
\end{aligned}
$$

Analyzing misdetection of nonce bits

When the MSB of k_i is leaked, then the attacker can collect biased signatures

$$k_1 = 0 11101\ldots$$
$$k_2 = 0 01010\ldots$$
$$k_3 = 0 10110\ldots$$
$$k_4 = 0 00011\ldots$$
$$\vdots$$

But sometimes the side-channel attacker makes mistakes..

$$k_1 = 0 11101\ldots$$
$$k_2 = 1 01010\ldots$$
$$k_3 = 0 10110\ldots$$
$$k_4 = 1 00011\ldots$$
$$\vdots$$

Analyzing misdetection of nonce bits

When the MSB of k_i is leaked, then the attacker can collect biased signatures

$$k_1 = 011101\ldots$$
$$k_2 = 001010\ldots$$
$$k_3 = 010110\ldots$$
$$k_4 = 000011\ldots$$
$$\vdots$$

But sometimes the side-channel attacker makes mistakes..

$$k_1 = 011101\ldots$$
$$k_2 = 101010\ldots$$
$$k_3 = 010110\ldots$$
$$k_4 = 100011\ldots$$
$$\vdots$$

Our analysis covers the behavior of $\mathsf{Bias}_q(K)$ under misdetection!

$$|\mathsf{Bias}_q(K)| \approx (1 - 2\epsilon) \times |\mathsf{Bias}_q(K_0)|$$

where $\epsilon \in [0, 1/2)$ is an error rate and $\mathsf{Bias}_q(K_0)$ is a bias without errors.

Time–Data tradeoffs for 1-bit leakage

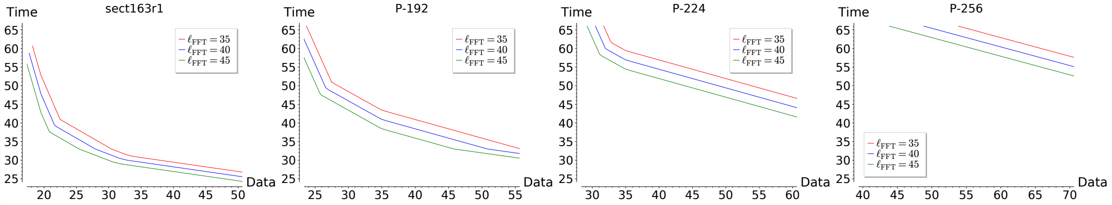


Figure 2: Time–Data tradeoff graphs (in a $\log_2$ scale) when memory is fixed to 2^{35}

* Optimized data complexity by solving the linear programming problem
* Much smaller amount of signatures needed if 2 or 3-bit leakage is available!

Experimental Results on Full Key Recovery

Target	Facility	Error rate	Input	Output	Thread (Collision)	Time (Collision)	RAM (Collision)	L_{FFT}	Recovered MSBs
NIST P-192	AWS EC2	0	2^{29}	2^{29}	96×24	113h	492GB	2^{38}	39
NIST P-192	AWS EC2	1%	2^{35}	2^{30}	96×24	52h	492GB	2^{37}	39
`sect163r1`	Cluster	0	2^{23}	2^{27}	16×16	7h	80GB	2^{35}	36
`sect163r1`	Workstation	2.7%	2^{24}	2^{29}	48	42h	250GB	2^{34}	35

- Attack on P-192 is made possible by our highly optimized parallel implementation.
- Attack on `sect163r1` is even feasible with a laptop.
- Recovering remaining bits is much cheaper in Bleichenbacher's framework.

Main takeaways

- ECDSA nonce is extremely sensitive
  - Even < 1-bit leakage/signature is exploitable!
- HNP is still relevant nowadays, even in 2020's!
- Open questions:
  - Can we further improve time–data tradeoffs?
  - Other sources of small leakage (e.g., 2 or 3-bit leakage under errors)?

Thank you! & Questions?

More details at `https://ia.cr/2020/615`

Main takeaways

- ECDSA nonce is extremely sensitive
  - Even < 1-bit leakage/signature is exploitable!
- HNP is still relevant nowadays, even in 2020's!
- Open questions:
  - Can we further improve time–data tradeoffs?
  - Other sources of small leakage (e.g., 2 or 3-bit leakage under errors)?

Thank you! & Questions?

More details at `https://ia.cr/2020/615`

Main takeaways

- ECDSA nonce is extremely sensitive
  - Even < 1-bit leakage/signature is exploitable!
- HNP is still relevant nowadays, even in 2020's!
- Open questions:
  - Can we further improve time–data tradeoffs?
  - Other sources of small leakage (e.g., 2 or 3-bit leakage under errors)?

Thank you! & Questions?

More details at `https://ia.cr/2020/615`

Main takeaways

- ECDSA nonce is extremely sensitive
  - Even $<$ 1-bit leakage/signature is exploitable!
- HNP is still relevant nowadays, even in 2020's!
- Open questions:
  - Can we further improve time–data tradeoffs?
  - Other sources of small leakage (e.g., 2 or 3-bit leakage under errors)?

Thank you! & Questions?

More details at `https://ia.cr/2020/615`

References i

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

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

Daniel Bleichenbacher.
Experiments with DSA.
Rump session at CRYPTO 2005, 2005.
Available from https://www.iacr.org/conferences/crypto2005/r/3.pdf.

Dan Boneh and Ramarathnam Venkatesan.
Hardness of computing the most significant bits of secret keys in Diffie-Hellman and related schemes.
In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 129–142. Springer, Heidelberg, August 1996.

Alejandro Cabrera Aldaya, Billy Bob Brumley, Sohaib ul Hassan, Cesar Pereida García, and Nicola Tuveri.
Port contention for fun and profit.
In *2019 IEEE Symposium on Security and Privacy*, pages 870–887. IEEE Computer Society Press, May 2019.

Fergus Dall, Gabrielle De Micheli, Thomas Eisenbarth, Daniel Genkin, Nadia Heninger, Ahmad Moghimi, and Yuval Yarom.
CacheQuote: Efficiently recovering long-term secrets of SGX EPID via cache attacks.
IACR TCHES, 2018(2):171–191, 2018.
https://tches.iacr.org/index.php/TCHES/article/view/879.

Freepik.
Icons made by Freepik from Flaticon.com.
http://www.flaticon.com.

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

Jan Jancar, Vladimir Sedlacek, Petr Svenda, and Marek Sýs.
Minerva: The curse of ECDSA nonces systematic analysis of lattice attacks on noisy leakage of bit-length of ECDSA nonces.
IACR Trans. Cryptogr. Hardw. Embed. Syst., 2020(4):281–308, 2020.

References v

Mingjie Liu and Phong Q. Nguyen.
Solving BDD by enumeration: An update.
In Ed Dawson, editor, *CT-RSA 2013*, volume 7779 of *LNCS*, pages 293–309. Springer, Heidelberg, February / March 2013.

Robert Merget, Marcus Brinkmann, Nimrod Aviram, Juraj Somorovsky, Johannes Mittmann, and Jörg Schwenk.
Raccoon attack: Finding and exploiting most-significant-bit-oracles in tls-dh(e).
Cryptology ePrint Archive, Report 2020/1151, 2020.
https://eprint.iacr.org/2020/1151.

Daniel Moghimi, Berk Sunar, Thomas Eisenbarth, and Nadia Heninger.
TPM-FAIL: TPM meets timing and lattice attacks.
CoRR, abs/1911.05673, 2019.
To appear at USENIX Security 2020.

Phong Q. Nguyen and Igor Shparlinski.
The insecurity of the digital signature algorithm with partially known nonces.
Journal of Cryptology, 15(3):151–176, June 2002.

Keegan Ryan.
Return of the hidden number problem.
IACR TCHES, 2019(1):146–168, 2018.
https://tches.iacr.org/index.php/TCHES/article/view/7337.

Keegan Ryan.
Hardware-backed heist: Extracting ECDSA keys from qualcomm's TrustZone.
In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 181–194. ACM Press, November 2019.

Akira Takahashi, Mehdi Tibouchi, and Masayuki Abe.
New Bleichenbacher records: Fault attacks on qDSA signatures.
IACR TCHES, 2018(3):331–371, 2018.
https://tches.iacr.org/index.php/TCHES/article/view/7278.

Sohaib ul Hassan, Iaroslav Gridin, Ignacio M. Delgado-Lozano, Cesar Pereida García, Jesús-Javier Chi-Domínguez, Alejandro Cabrera Aldaya, and Billy Bob Brumley.
Déjà vu: Side-channel analysis of mozilla's NSS.
In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 1887–1902. ACM, 2020.

Samuel Weiser, David Schrammel, Lukas Bodner, and Raphael Spreitzer.
Big Numbers - Big Troubles: Systematically analyzing nonce leakage in (EC)DSA implementations.
In *USENIX Security 2020)*, Boston, MA, August 2020. USENIX Association.