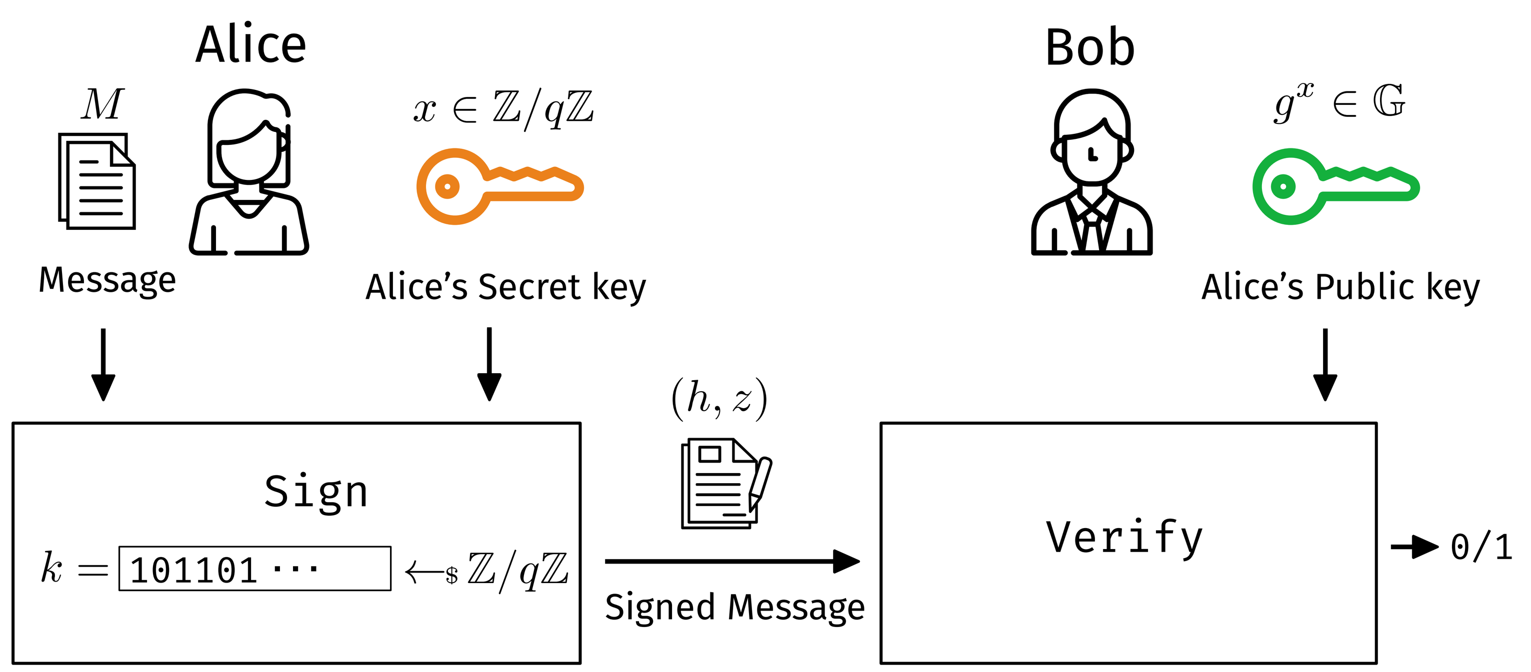
LadderLeak: Breaking ECDSA with Less than One Bit of Nonce Leakage

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

Risk of randomness failure in ECDSA-type signatures



- 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 $x = (z - z')/(h' - h) \mod q$
- What if k is **biased** or **partially leaked**? $\leadsto$ Attack possible by solving the **hidden number problem (HNP)**!
- Two different approaches to HNP: Fourier analysis vs lattice attack.

Challenges

- Can we reduce the data complexity of Fourier analysis-based attack?
- Can we attack even **less than 1-bit of nonce leakage** (i.e., top-most bit of nonce k is only leaked with prob. < 1)?
- Can we obtain such a small leakage from practical ECDSA implementations?

Our contributions

1. Novel class of cache attacks against the Montgomery ladder scalar multiplication in OpenSSL `1.0.2u` and `1.1.01`, and RELIC 0.4.0.
   - **Affected curves:** NIST P-192, P-224, P-256 (not by default in OpenSSL), P-384, P-521, B-283, K-283, K-409, B-571, `sect163r1`, `secp192k1`, `secp256k1`
2. Improved theoretical analysis of the Fourier analysis-based attack on HNP (originally established by Bleichenbacher)
   - Significantly reduced the required input data
   - Analysis in the presence of erroneous leakage information
3. Implemented a full secret key recovery attack against OpenSSL ECDSA instantiated over `sect163r1` and NIST P-192.

Comparison with previous HNP records

	< 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]	—

LadderLeak: Tiny timing leakage from the Montgomery ladder

Algorithm 1 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 $Q = R_0$

Conditions for the attack to work:

1. Group order is $2^n - \delta$ with small δ.
2. Accumulators (R_0, R_1) are in **projective coordinates**, but initialized with the base point in **affine coordinates**.
3. Group law is non-constant time wrt handling Z coordinates $\leadsto$ **Weierstrass model**

Cache-timing attack experiments

Experiments *were carried out with Flush+Reload cache attack technique $\leadsto$ MSB of k was detected with > 99 % accuracy.*

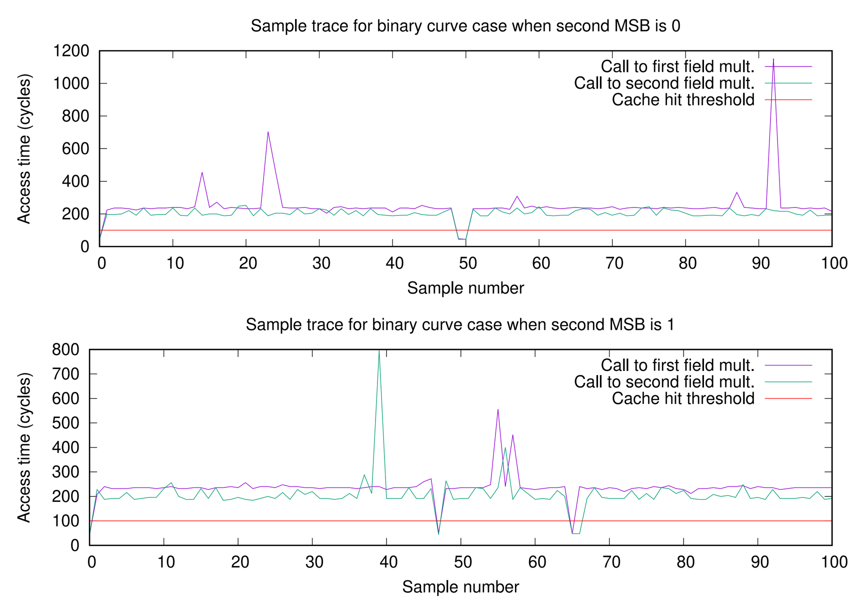


Figure 1. Pattern in traces collected by `FR-trace` for the **binary curve** case.

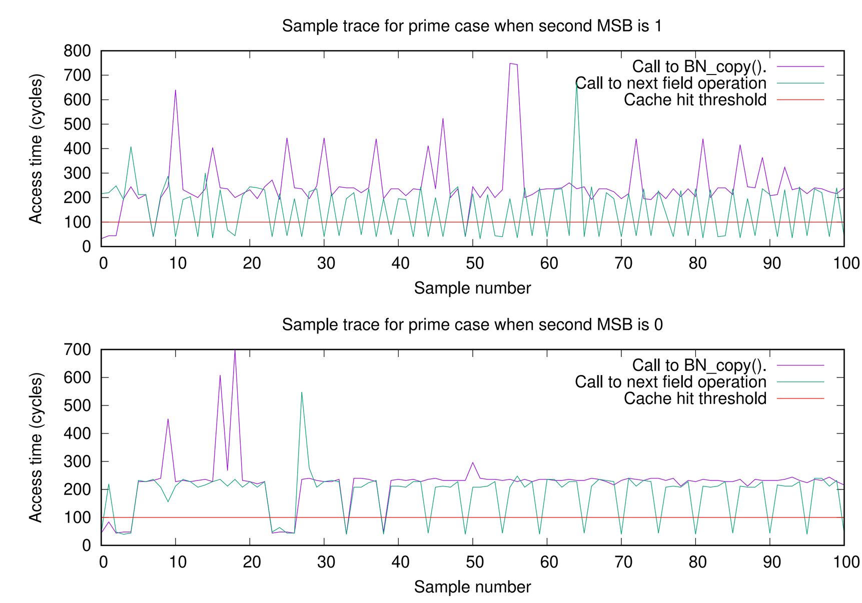


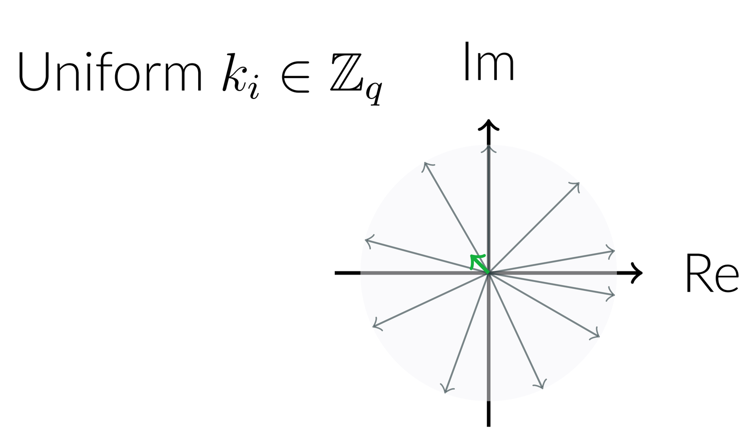
Figure 2. Pattern in traces collected by `FR-trace` for the **prime curve** case.

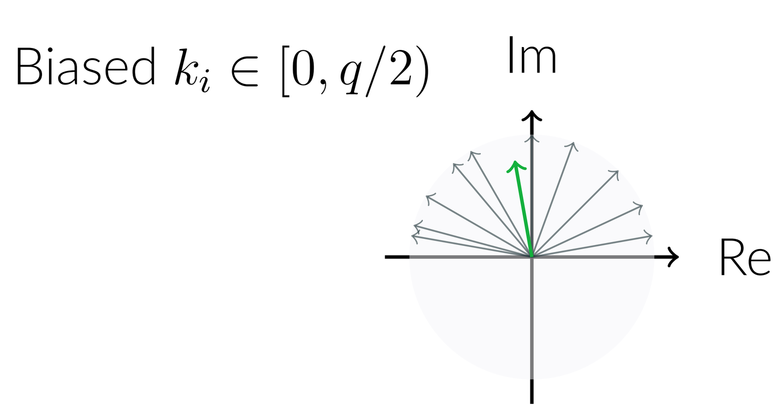
How to quantify the nonce bias

Bias function

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

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





Bleichenbacher's Fourier analysis-based attack

- **Step 1.** Quantify the modular bias of randomness K by defining a bias function $\mathrm{Bias}_q(K)$.
  - Improvement 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
  - Improvement 2 Established **unified time-memory-data tradeoffs** by applying $\mathcal{K}$-list sum algorithm for the GBP!

Tradeoff graphs for 1-bit bias

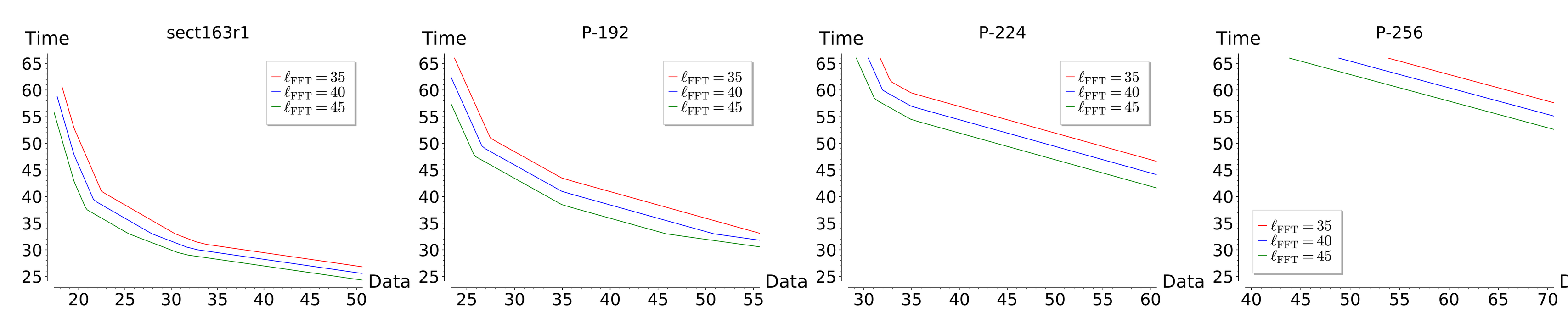


Figure 3. Time–Data tradeoffs when memory is fixed to 2^{35}.

- Optimized data complexity obtained by solving the linear programming problem.
- Paper has various tradeoff graphs and improved complexity estimates for 2-3 bits bias.

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.
- Attacks on P-224 with 1-bit bias or P-256 with 2-bit bias are also tractable.

Main takeaways

- Securely implementing brittle cryptographic algorithms is still **hard**.
- **Don't** underestimate even less than 1-bit of nonce leakage!
- Interesting connection between the HNP and GBP (from symmetric key crypto)
- Future work:
  - More list sum algorithms and tradeoffs?
  - Improvements to FFT computation?
  - Other sources of small leakage?

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