CacheQuote: Efficiently Recovering Long-term Secrets of SGX EPID via Cache Attacks

September 11th 2018

CHES, Amsterdam, The Netherlands

Fergus Dall, **Gabrielle De Micheli,** Thomas Eisenbarth, Daniel Genkin, Nadia Heninger, Ahmad Moghimi, and Yuval Yarom

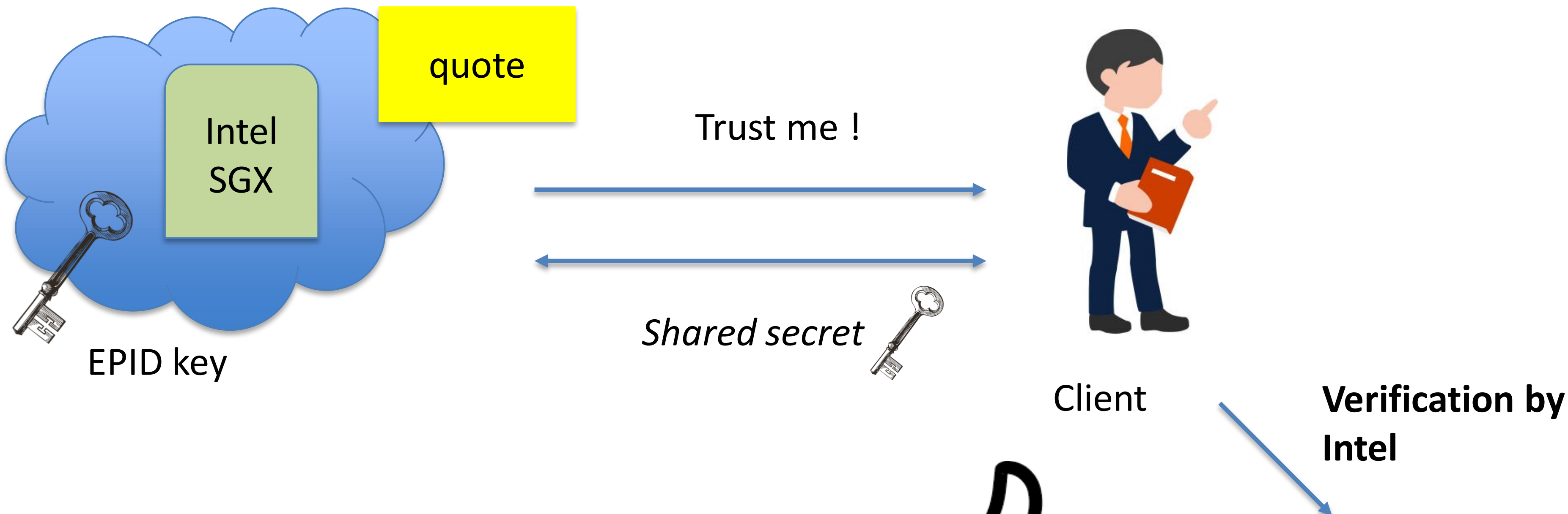
Intel Software Guard Extensions

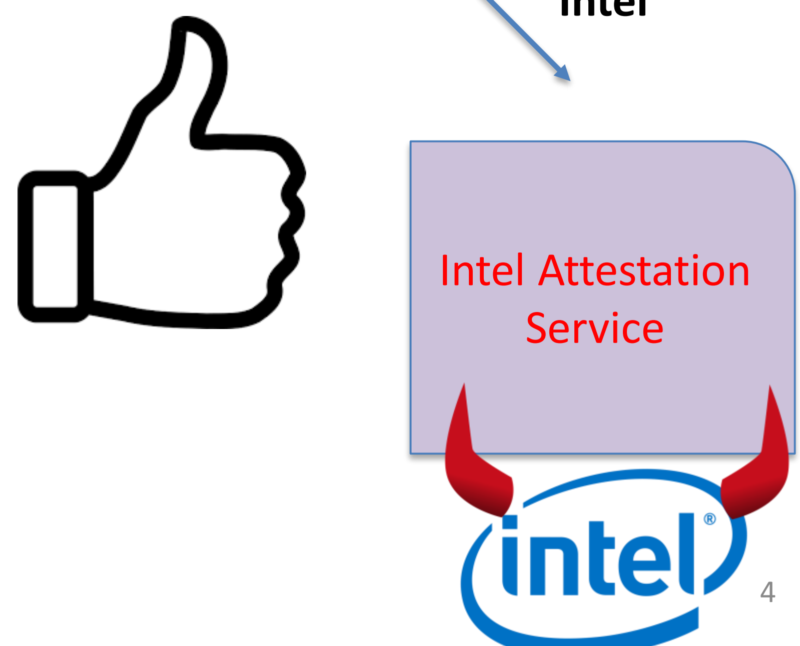
1. Set of instructions aiming to guarantee confidentiality and integrity of applications that run inside untrusted environments.
2. Protects *enclaves* of code and data.

Enclaves

- Enclaves are isolated from the software running on the computer.
- SGX controls the entry to and exit from enclaves.

Remote attestation: EPID



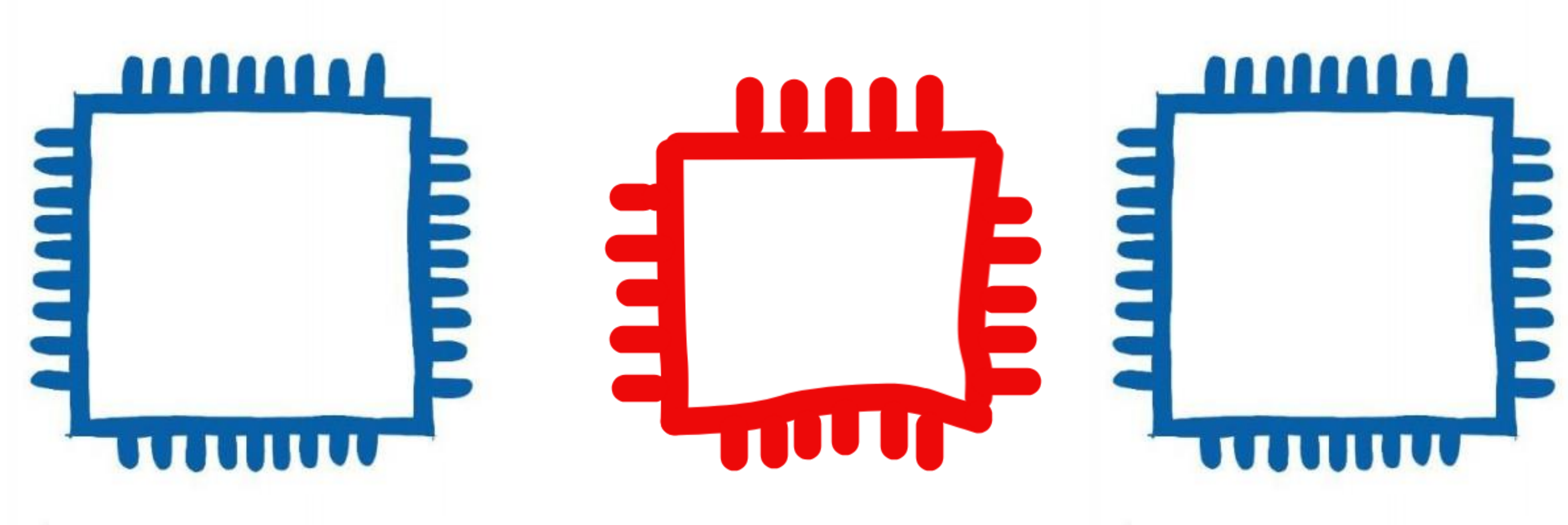


Trust is based on the EPID key!

Why need IAS ? Revocation !
All quotes are encrypted by SGX.

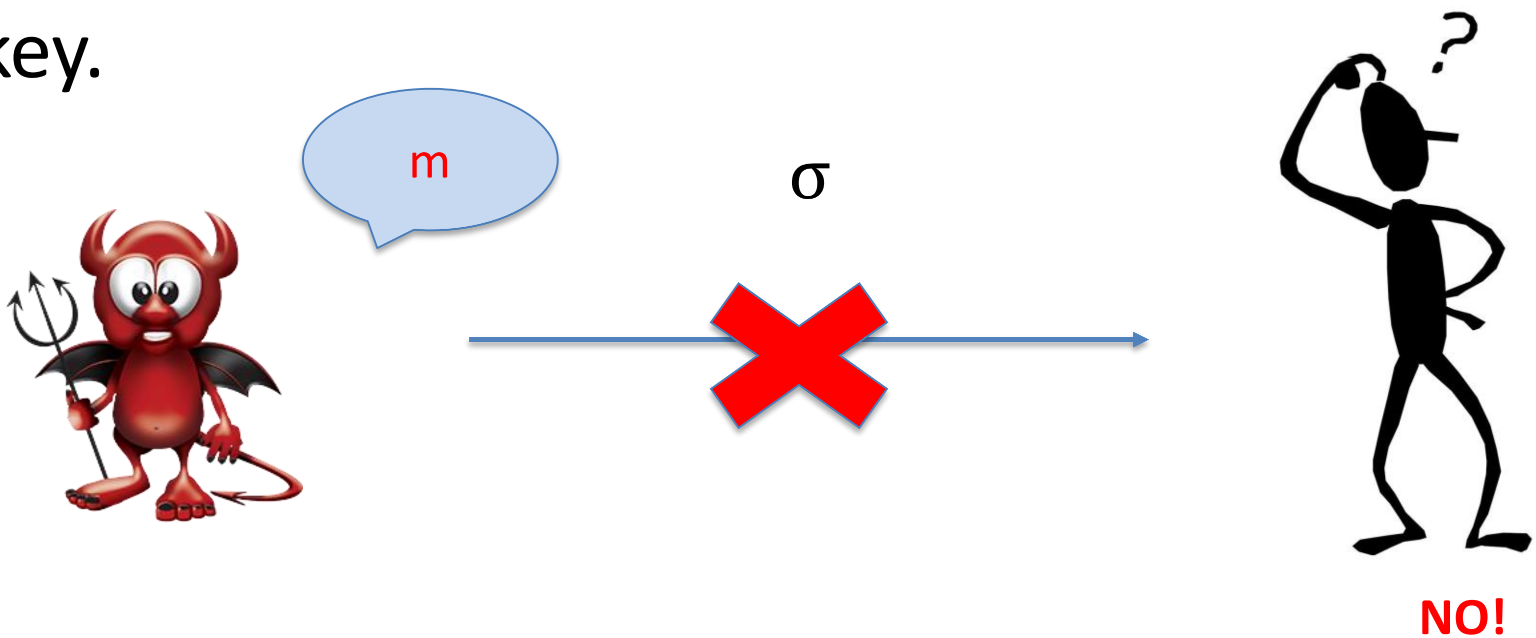
Unlinkability

➡ impossible to identify the platform that produced a signature on some message m.



Unforgeability

impossible for an attacker to forge a valid signature on some previously-unsigned message, without knowing a non-revoked secret key.

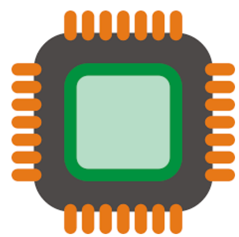


Our results

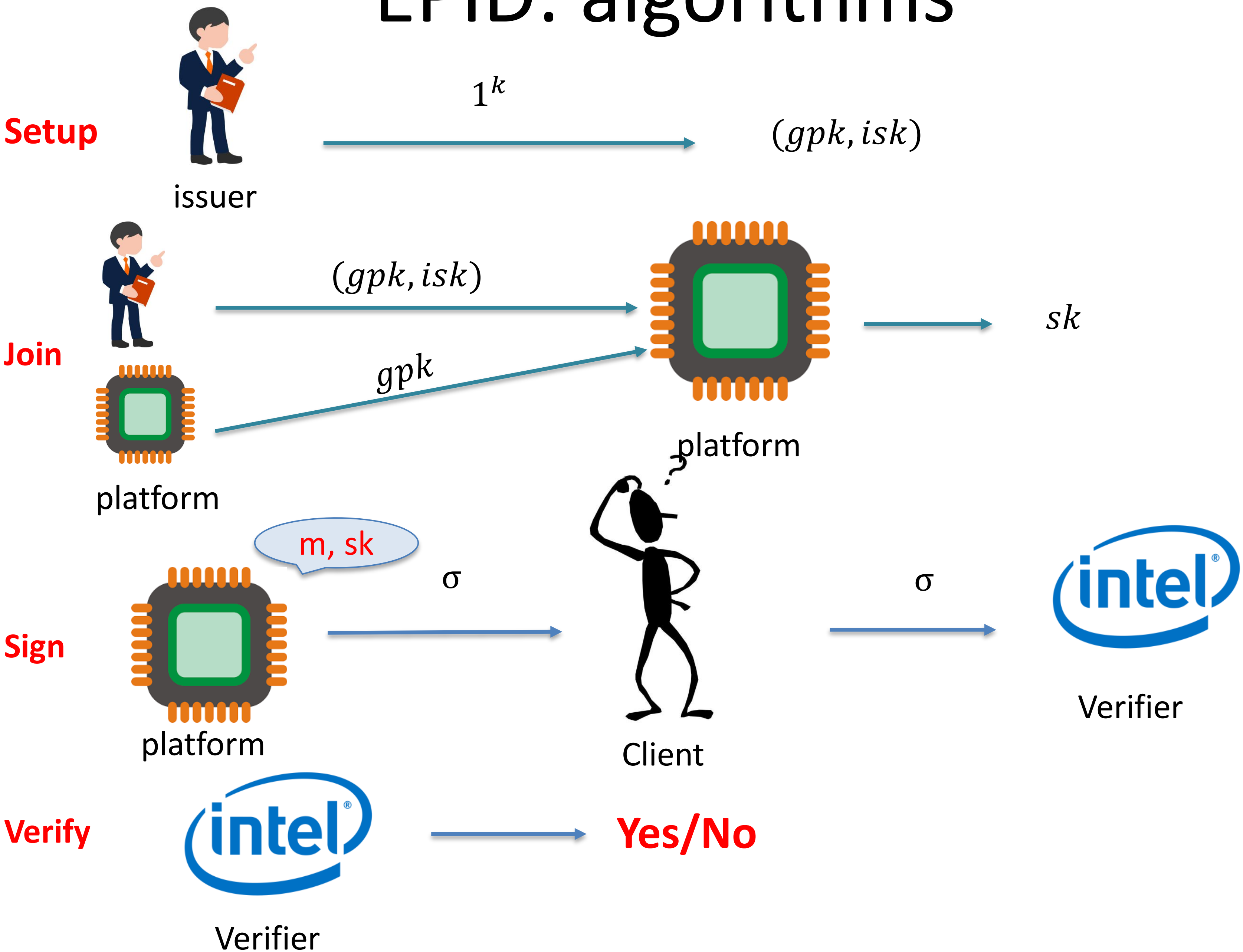
- First cache attacks on Intel’s EPID protocol implemented inside SGX.

- Recover part of the enclave’s long term secret key.
- Malicious attestation server (Intel) can break the unlinkability guarantees of SGX’s remote attestation protocol.

EPID: setup

- An issuer: 
- A revocation manager: 
- A platform: 
- A verifier: intel

EPID: algorithms



The signing algorithm

- Secret key: f + Intel's signature on f
- Randomly choose: $B \in G$ and compute

$$K := B^f$$

- How to sign ?

Non-interactive zero knowledge proof of knowledge:

"I know an unrevoked f such that $K := B^f$*"*

- Requires computing A^r , where A is some value.
- Signature σ has the values *K, B* and *s* $\leftarrow r + Hf$

Attack idea

- Recover side-channel information about the length of the nonce r from A^r.

- After many observations, use length data to mount a lattice attack to recover the value of f.

- Break unlinkability.

How unlinkability is broken?

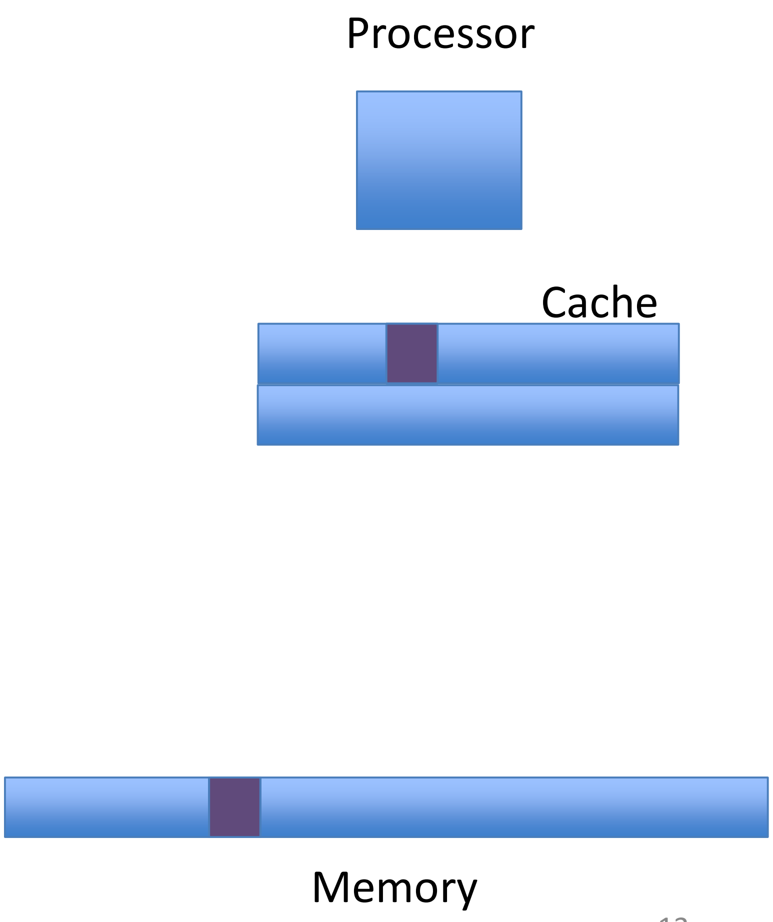
- f is unique per platform and private.
- The attacker knowns a signature $\sigma = (K, B, \dots)$ on some message m and f.
- He can check if $K = B^f$.
- If yes, then the signature was issued by the platform whose key is f.

CPU vs memory

Caches are used to bridge the gap.

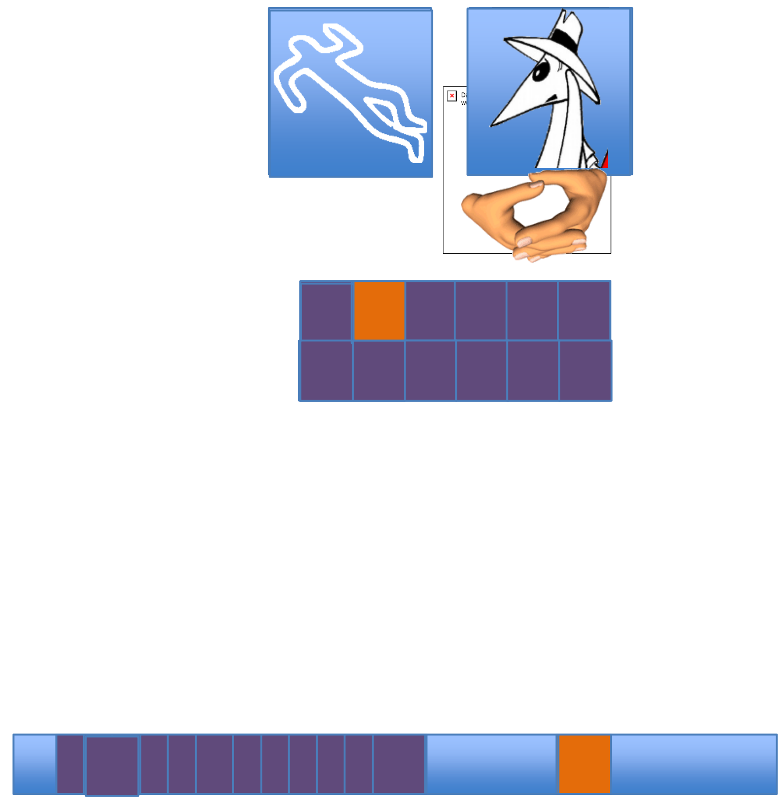
- Divides memory into *lines*
- Stores recently used lines

- In a *cache hit*, data is retrieved from the cache
- In a *cache miss*, data is retrieved from memory and inserted to the cache



The Prime+Probe Attack

- Allocate a cache-sized memory buffer
- *Prime:* fills the cache with the contents of the buffer
- *Probe:* measure the time to access each cache set
  - Slow access indicates victim access to the set



In our attack

- The signing algorithm requires computing A^r.
- Exponentiation uses some variant of square and multiply with fixed windows of bits.
- Quoting enclave recodes the nonce r to have fewer non-zero bits.

Scalar multiplication algorithm

MultPoint(point P, window size w , scalar r):

Initialize P : $P_0 \leftarrow O$

For $i \leftarrow 1$ to 2^{w-1} do:

$\quad P_i \leftarrow P \cdot P_{i-1}$

$i \leftarrow \max(j : r_j \neq 0)$ ← Start with MSB $\neq 0$

$s \leftarrow P_{r_i}$

$i \leftarrow i - 1$

Main loop:

While $i \geq 0$ do:

$\quad s \leftarrow r^{2^w}$ ← w squaring operations

$\quad s \leftarrow s \cdot P_{r_i}$ ← Multiplication with precomputed value P_{r_i} (selected in constant-time)

$\quad i \leftarrow i - 1$

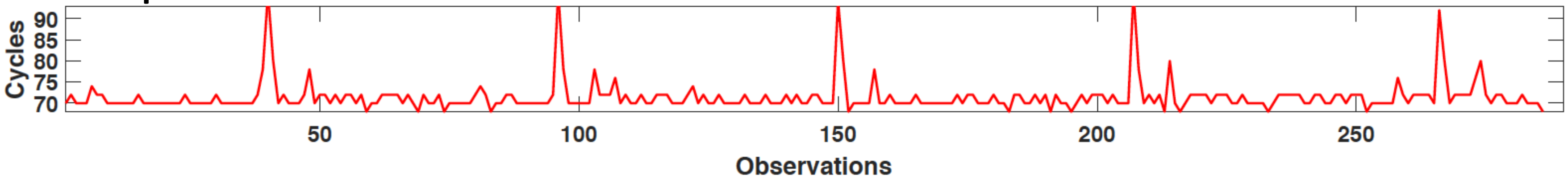
End while

Output: s

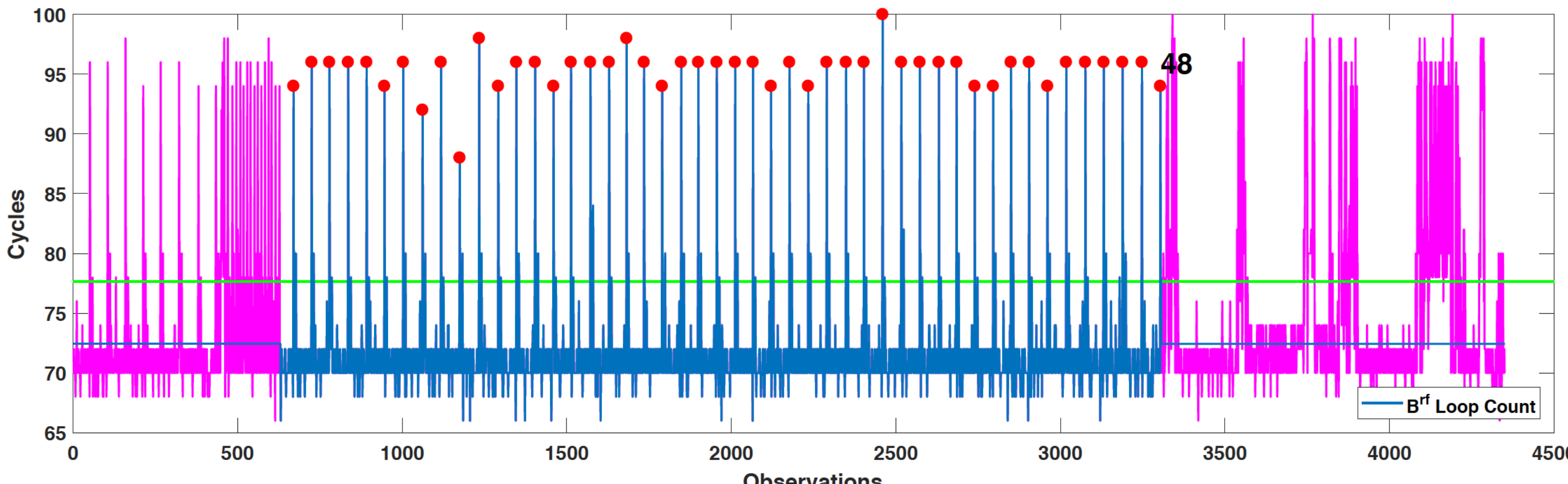
- Scalar of length 256 bits ⇨ recoded scalar of length 52 ⇨ 51 loop iterations.
- Bits 256 and 255 are 0 ⇨ recoded scalar of length 51 ⇨ 50 loop iterations.

Counting loops

- Monitor cache access patterns during the computation of the main loop.



- One period corresponds to one loop iteration.
- Number of periods gives us information on the number of iterations.



A lattice attack

Side channel ⟶ information about the length of r.

Goal: Solve for f.

- Many samples $\{(s, H)\}_i$ such that:

$$s \equiv r + Hf \bmod p$$

- Information about the number l_i of most significant zero bits in r_i.
- We learn $|s_i - H_i f| = |r_i| < \frac{p}{2^{l_i}}$

⟶ hidden number problem and obtain *f*.

Recovering *f*

10 600 signatures required if only using 49-loop samples to get 37 error-free samples.

Signatures	48-loop	49-loop	50-loop	BKZ block size	BKZ time
10300	2	35	0	2	0.1s
10000	2	31	10	20	0.2s
9000	2	29	21	30	1.4s
8000	2	25	35	30	4.5s

- Use samples of different loop lengths

- Reduce the number of signatures with manual inspection: less than 7 500 observed signatures to obtain enough 49-loop observations for a full key recovery.

Conclusion

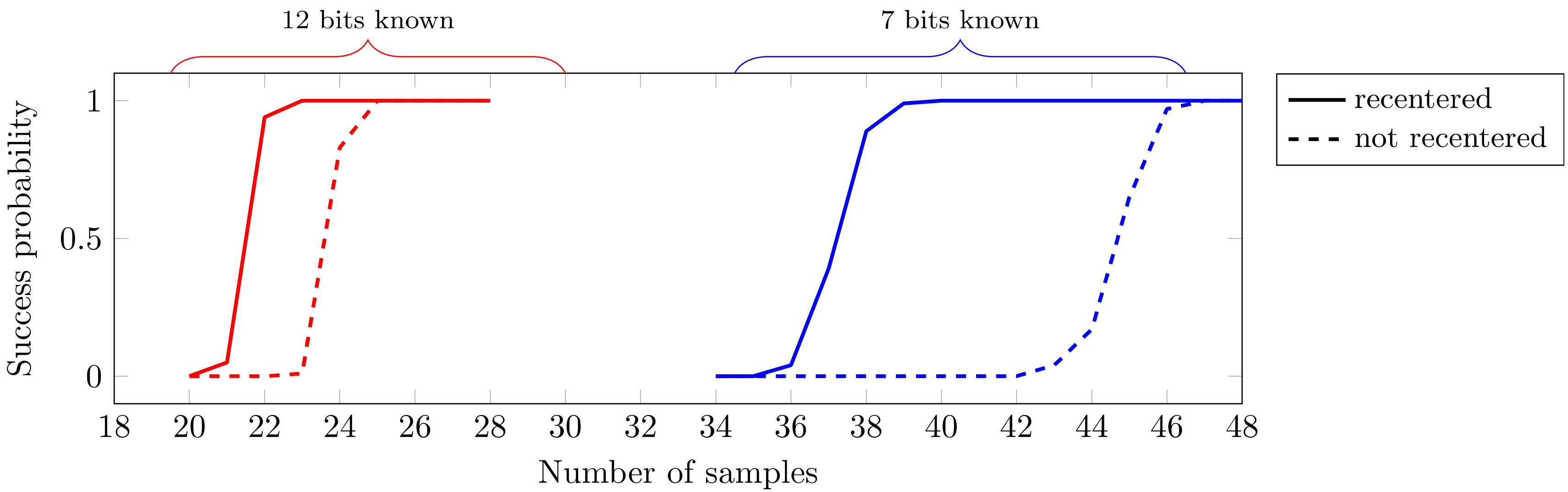
- We finally have *f*.
- Limitations: we can’t run the attack ourselves as all the EPID signatures are encrypted with Intel’s public key !
- A malicious Intel could break the unlinkability guarantee.
- Thank you !

Thank you !

CacheQuote: Efficiently Recovering Long-term Secrets of SGX EPID via Cache Attacks

Fergus Dall, Gabrielle De Micheli, Thomas Eisenbarth, Daniel Genkin, Nadia Heninger, Ahmad Moghimi, and Yuval Yarom

Key recovery with the hidden number problem



In our experiments, 8000 signatures necessary to enough error-free samples for key recovery.

Recoding the nonces

- Non-adjacent form (NAF) encoding:

  a. no two sequential non-zero digits.

  b. signed digits
- Example:

  a. binary: $(0,1,1,1) = 2^2 + 2^1 + 2^0 = 7$

  b. 2-NAF: $(1,0,0,-1) = 2^3 - 2^0 = 7$

- Generalization to *w*-NAF: work in base 2^w.
- The quoting enclave *recodes* the scalar r_f using some variant of *w*-NAF.

$r_f = (r_1, \cdots r_n)$ s.t.:

1. $r_f = \sum_i 2^{w \cdot i} r_i$
2. $-2^w - 1 \leq r_i \leq 2^w - 1.$

- Example: $(0, 0, 1, -25) = 2^{5 \cdot 1} \cdot 1 + 2^{5 \cdot 0} \cdot (-25) = 7$