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

August 29th 2018

University of Pennsylvania, Security Seminar

Gabrielle De Micheli

Joint work with:

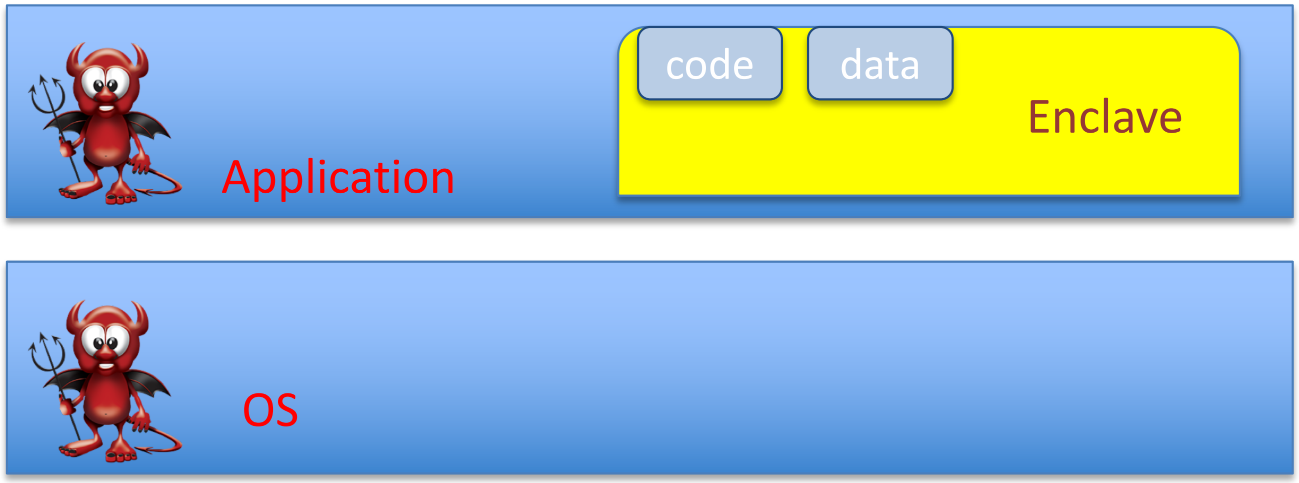
Fergus Dall, Thomas Eisenbarth, Daniel Genkin, Nadia Heninger,

Ahmad Moghimi, and Yuval Yarom

Intel Software Guard Extensions

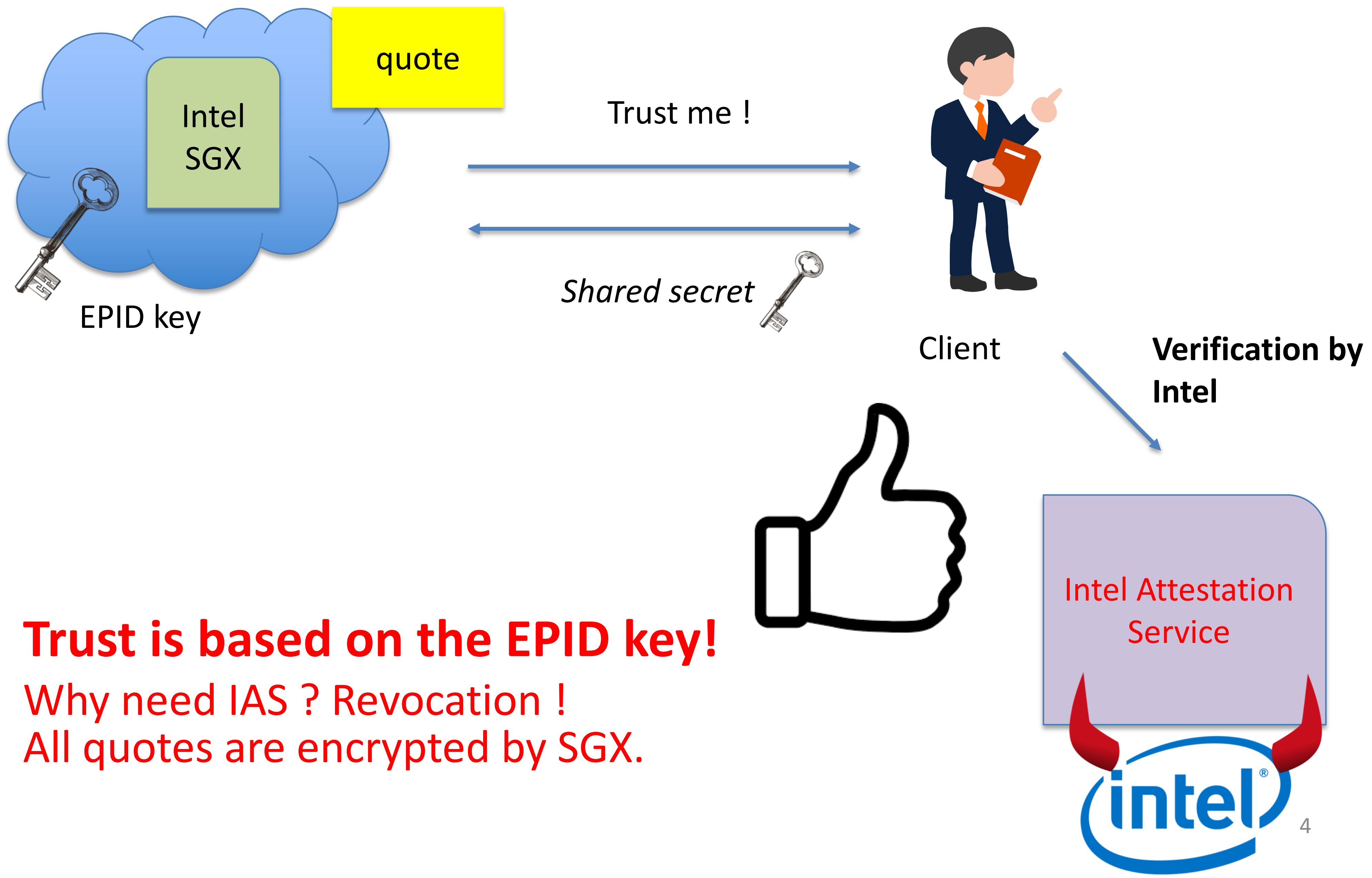
1. Set of instructions aiming to guarantee confidentiality and integrity of applications that run inside untrusted environments.
2. Protects *enclaves* of code and data

Enclaves



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

Remote attestation: EPID

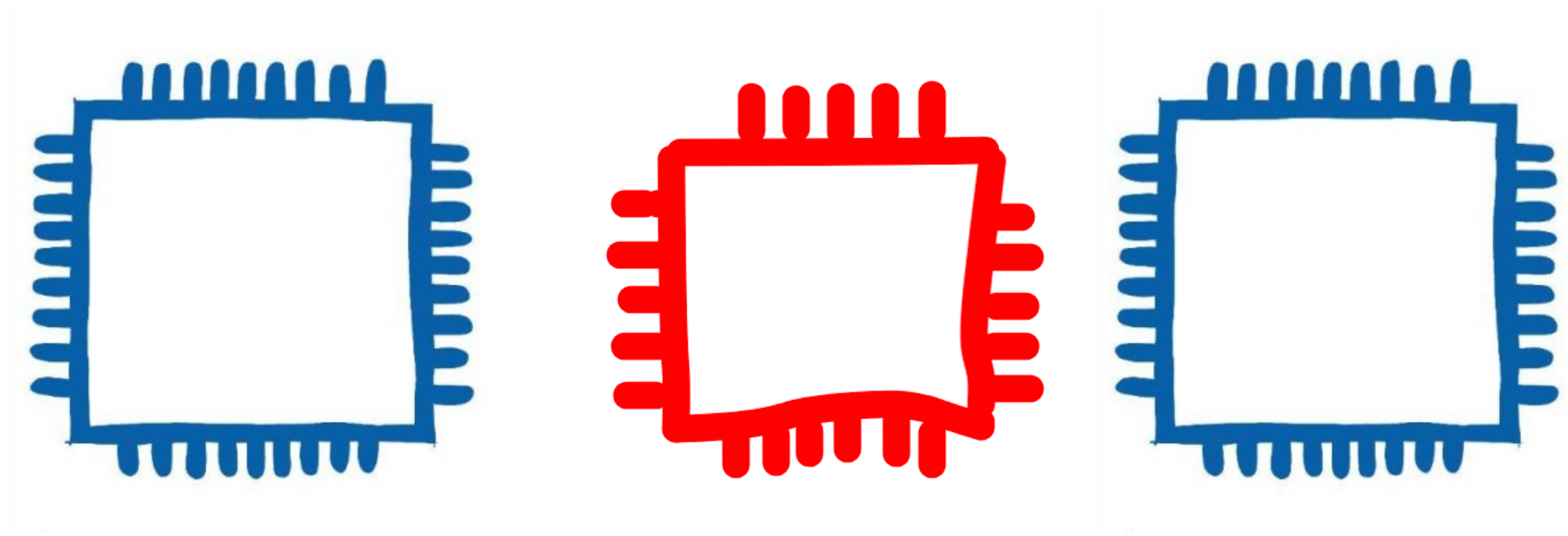


Trust is based on the EPID key!

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

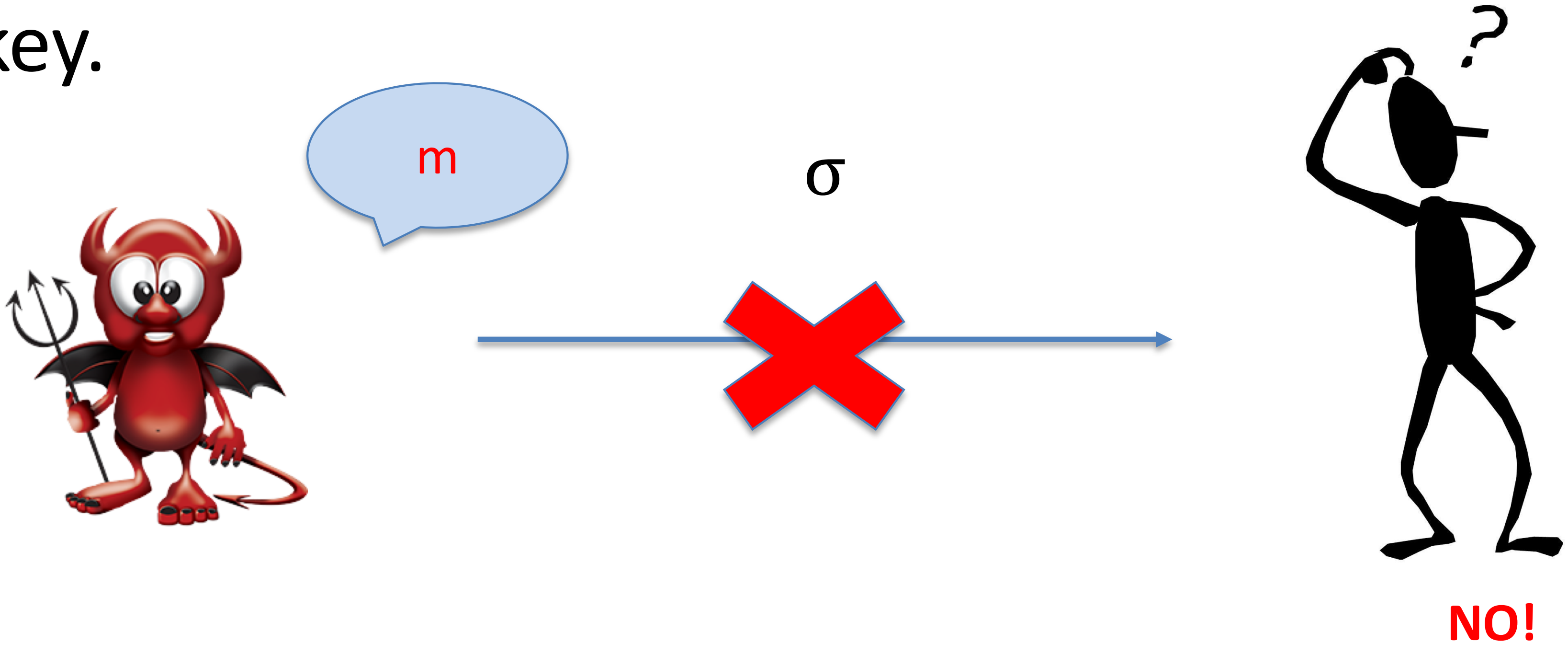
Unlinkability

➡ 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.

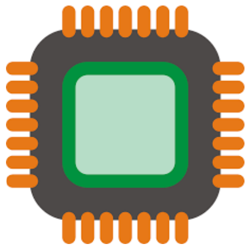
- Malicious attestation server (Intel) can break the unlinkability guarantees of SGX's remote attestation protocol.

EPID: setup

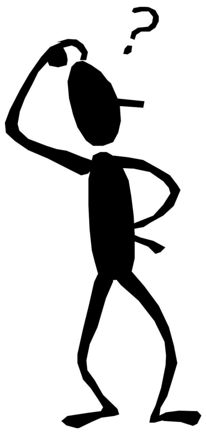
- An issuer:

- A revocation manager:

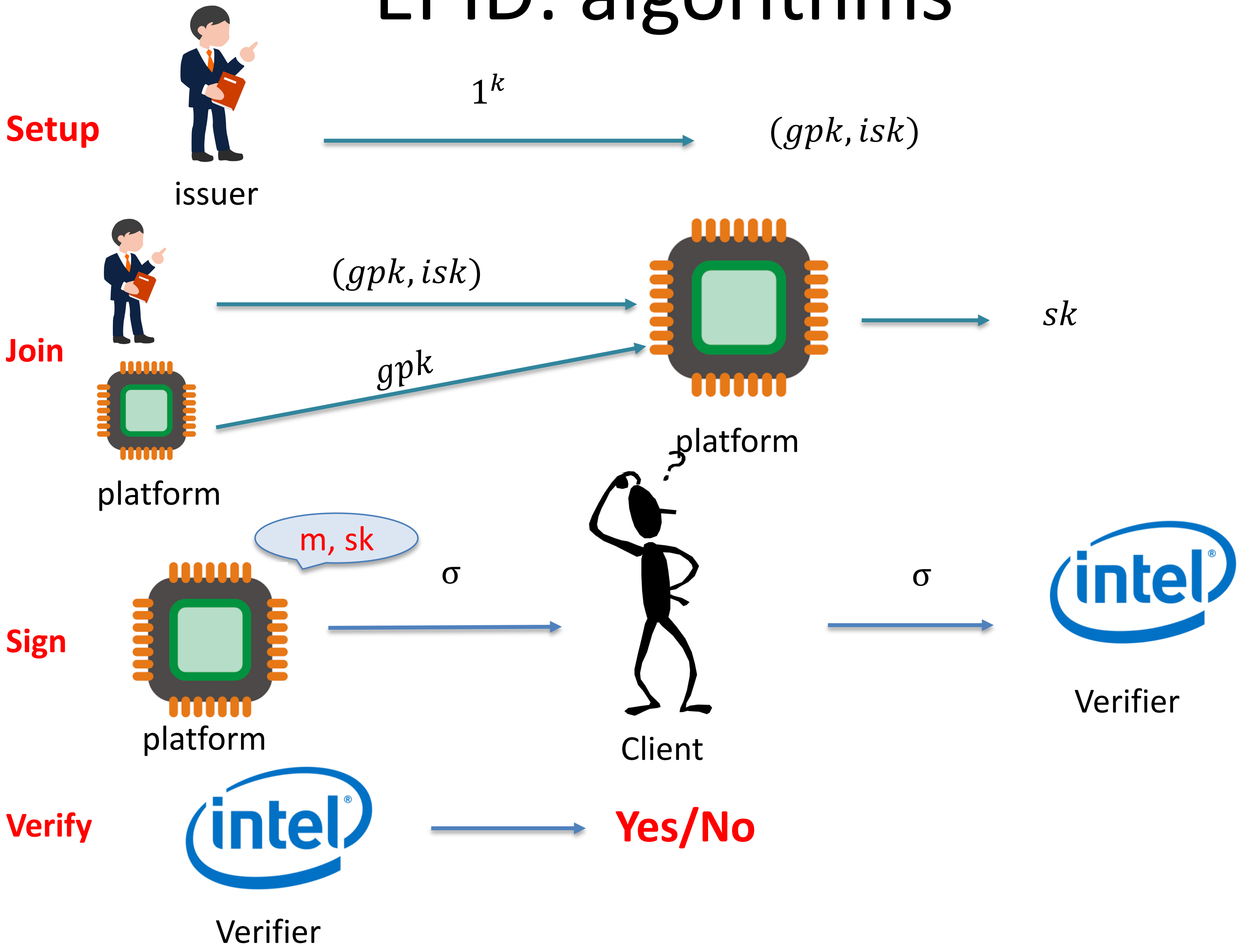
- A platform:



- A verifier:



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_f}, where A is some value.
- Signature σ has the values *K*, *B* and $s_f \leftarrow r_f + Hf$

Attack idea

- Recover side-channel information about the length of the nonce r_f from A^{r_f}.

- 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 $\sigma = (K, B, \ldots)$ 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.

Side-channel attacks

- Attacks based on information obtained from leakage between software and hardware.
- Timing side-channel attacks: exploit timing variation in execution time of cryptographic algorithms.
- Example: execution time of square and multiply algorithm used in modular exponentiation depends linearly on the number of non-zero bits in the key.

Square and multiply

Goal: fast computation of large positive integer powers of a number

Algorithm

```
Input: a, b
Output: $c = a^b$
Convert exponent to binary: $b = b_{t-1} \cdots b_0$
$c = 1$
For $j = t - 1 \; \ldots 0$ to 0, do:
    If $b_j = 0$: square $c \leftarrow c^2$
    If $b_j = 1$: square and multiply $c \leftarrow c^2 \times a$
Return $c$
```

Example

Input: 3, 5

Output: 3^5

1. 5 = 101
2. 3
3. 3^2
4. $(3^2)^2 \times 3$

We did 3 computations instead of 5!

1024: in binary: 10000000000 $\longrightarrow$ 10 calculations

Time of execution depends on number of multiplication, which depends on the number of 1's.

Cache attacks

- **Memory accesses are not always performed in constant time!**

⇒ Cache attacks: analysis of the cache behavior.

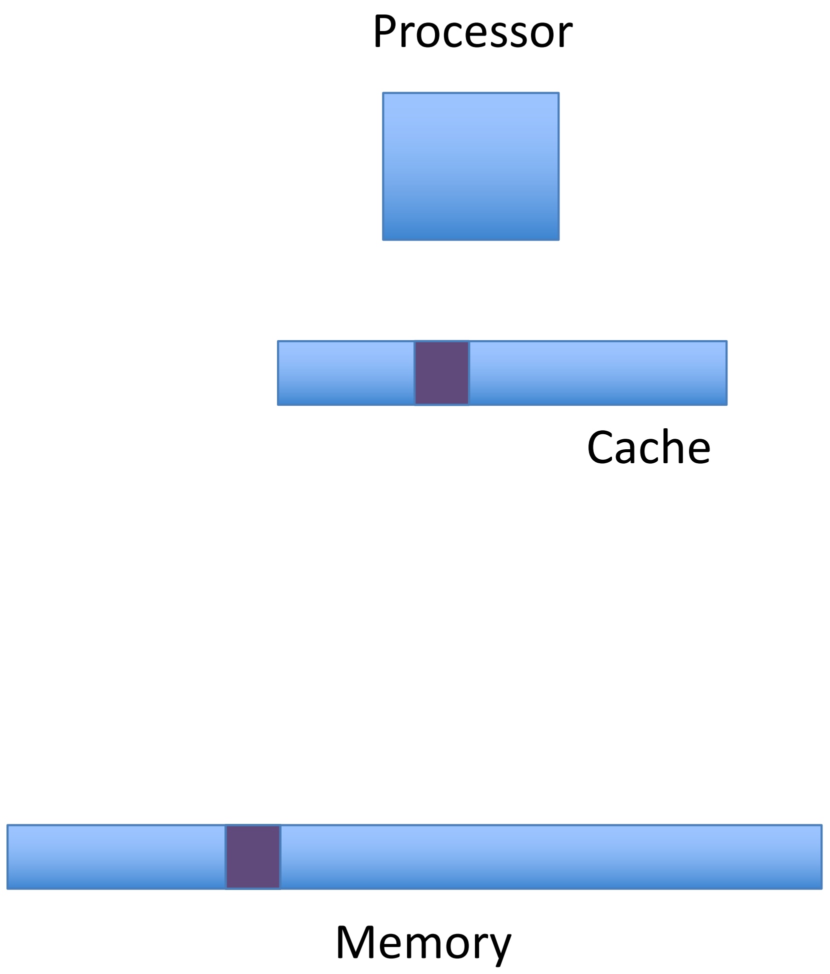
- Attacks: Prime and Probe [Per05, OST06]

CPU vs. Memory

Cache are used to bridge the gap

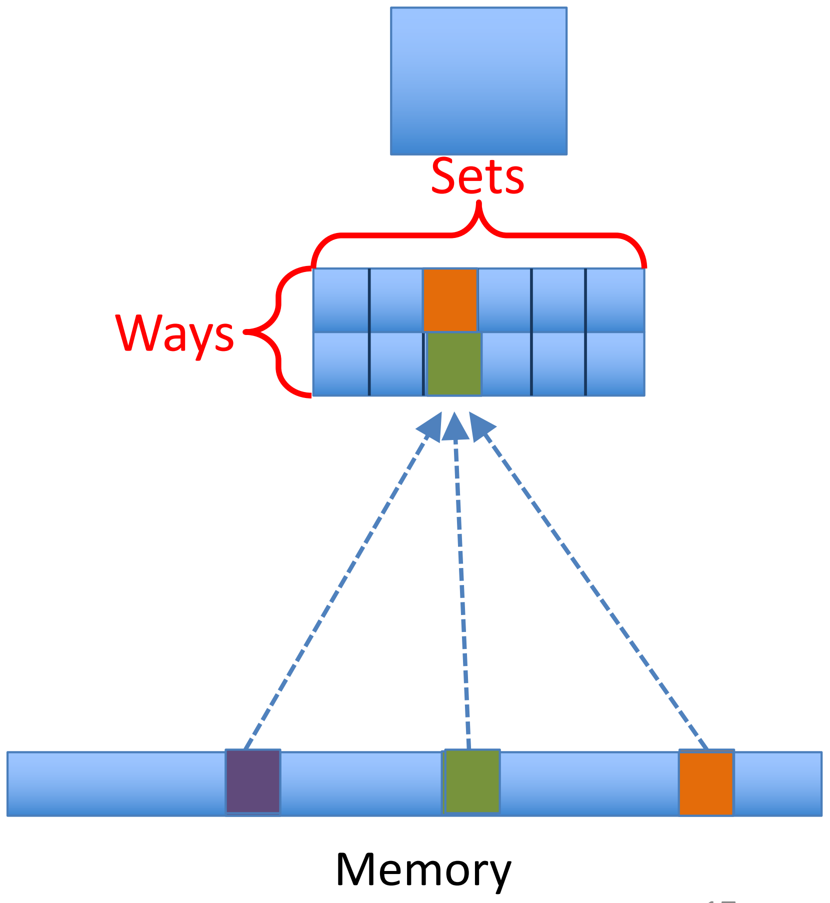
- Divides memory into *lines*
- Stores recently used lines

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



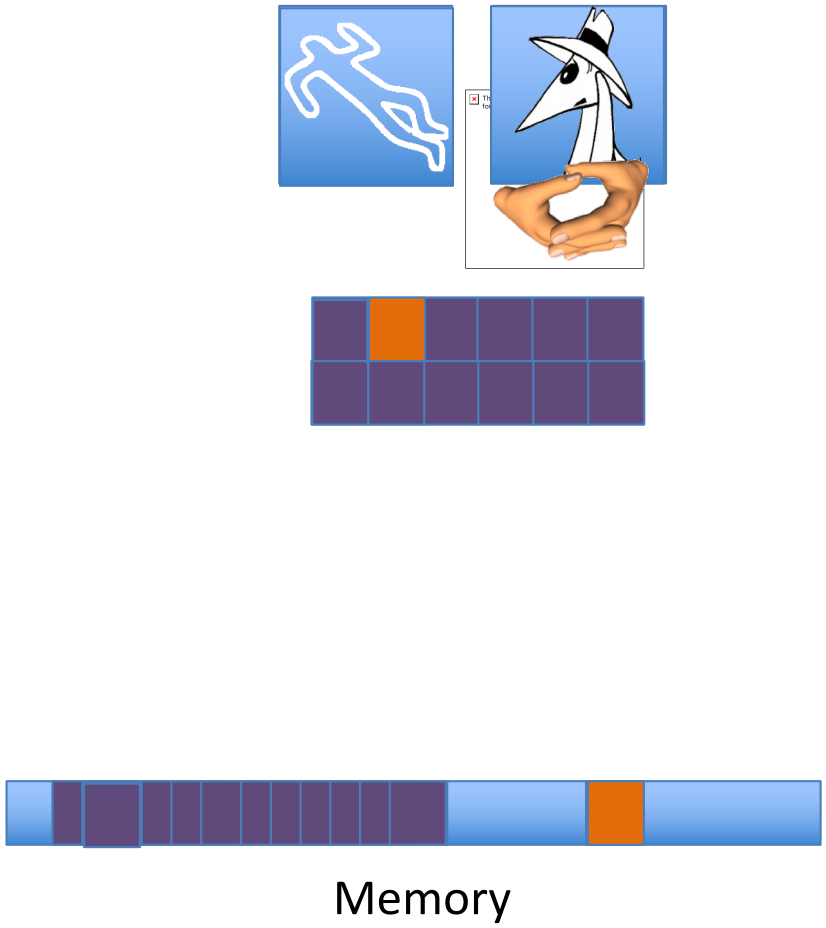
Set Associative Caches

- Memory lines map to *cache sets*. Multiple lines map to the same set.
- Sets consist of *ways*. A memory line can be stored in **any** of the ways of the set it maps to.
- When a cache miss occurs, one of the lines in the set is *evicted*.



The Prime+Probe Attack [Per05, OST06]

- 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



Prime+Probe attack examples

- RSA (OpenSSL 0.9.7c), Percival 2005
- AES (OpenSSL 0.9.8), Osvik, Shamir, and Tromer. 2005 Tromer, Osvik, and Shamir. 2010
- DSA (OpenSSL 0.9.8d) Onur Acıic¸mez, Brumley, and Grabher. 2010
- ECDSA (OpenSSL 0.9.8k) Brumley and Hakala. 2009
- ElGamal (GnuPG v.2.0.19,libgcrypt v.1.5.0) Zhang, Juels, Reiter, and Ristenpart. 2012

Countermeasures

- Constant-time techniques:
  - remove conditional execution (two conditions can have different execution time)
  - no secret dependent memory access …

In our attack

- The signing algorithm requires computing: A^{r_f}
- Use some variant of square and multiply which uses windows of bits.
- Exponentiation faster with fewer non-zeros bits (fewer multiplications)
- Recode the nonce r_f to have fewer non-zero bits.

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. $-2^w - 1 \le r_i \le 2^w - 1.$

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

Scalar multiplication algorithm

```
MultPoint(point $P$, window size $w$ , scalar $r_f = \mathrm{r}$):
Initialize $P$ : $P_0 \leftarrow O$
For $i \leftarrow 1$ to $2^{w-1}$ do:
    $P_i \leftarrow P \cdot P_{i-1}$
$i \leftarrow \max(j : r_j \neq 0)$          ← Start with MSB $\neq 0$
$s \leftarrow P_{r_i}$
$i \leftarrow i - 1$
While $i \geq 0$ do:                              Main loop
    $s \leftarrow r^{2^w}$                      ← $w$ squaring operations
    $s \leftarrow s \cdot P_{r_i}$              ← Multiplication with precomputed value $P_{r_i}$ (selected in constant-time)
    $i \leftarrow i - 1$
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.

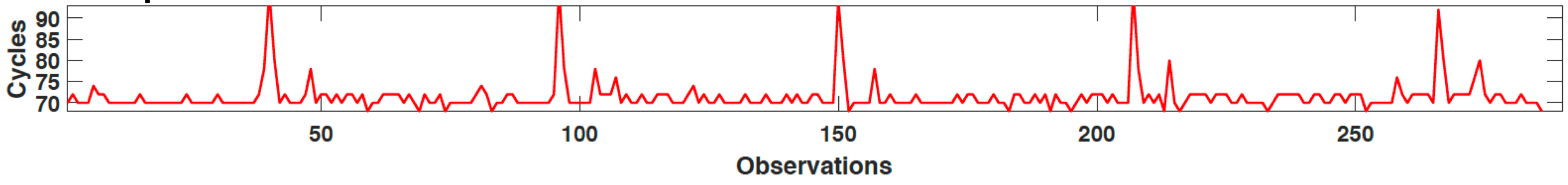
Going back to the attack

- Goal: get information about the MSB of the nonce r_f.
- Idea: we want to use Prime+Probe to count the number of iterations in the main loop of our scalar multiplication algorithm.
- How?

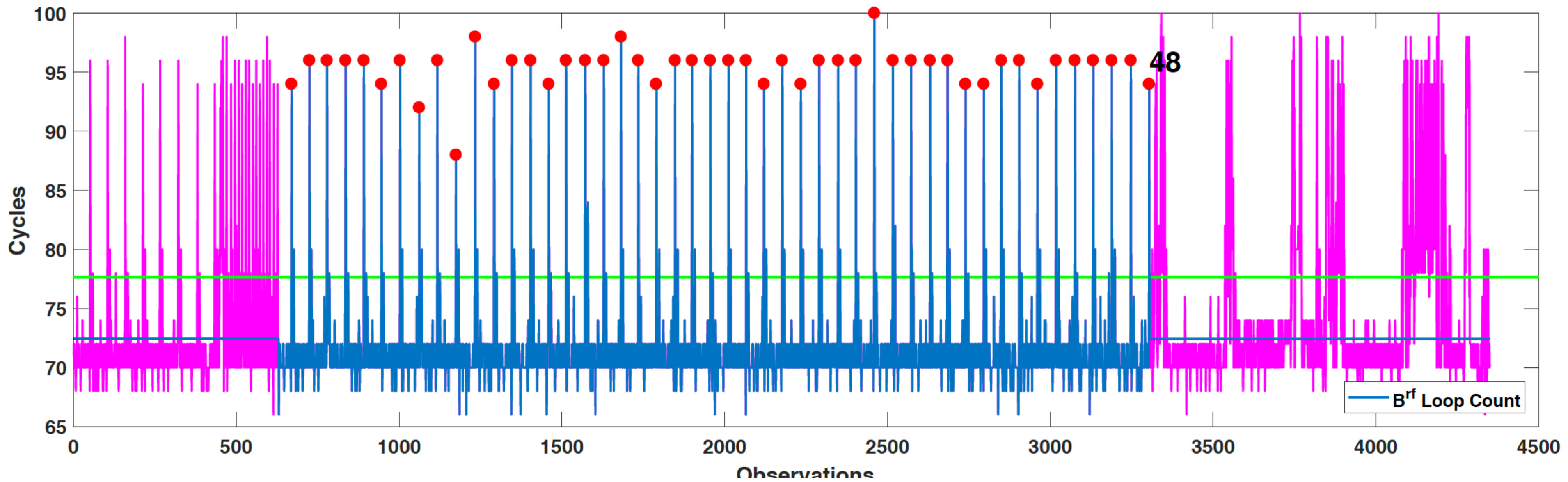
1. code is data: executing code means memory accesses (to bring the instructions from memory).
2. monitor the memory accesses needed to bringing the loop code in, which will tell us the number of iterations that the loop did.

Counting loops

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

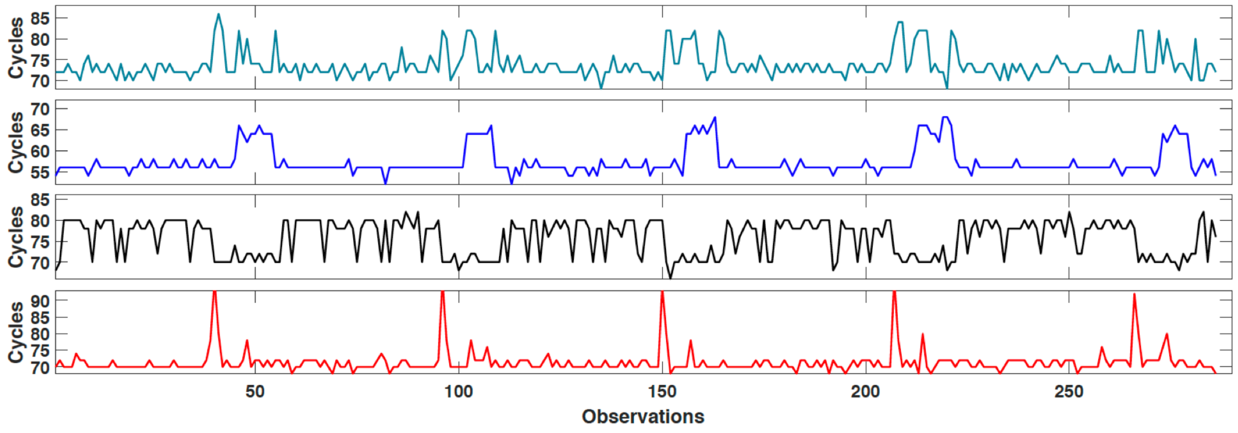


- One period corresponds to one loop iteration.
- Number of periods gives us information on the number of iterations.



Counting loop iterations automatically

- Matlab signal processing toolbox.
- Use several cache sets: the signal pattern is unique for each cache sets).
- Use five different loop counters that use information from different cache sets to count number of loops on each signature.



Handling noise

Common sources of error:

1. failing to accurately detect the beginning and the end of the multiplier window.
2. under-counting short peaks
3. over-counting occasional noises that introduce unexpected peaks or pattern.

→ if four of the five loop counters agree on the number of loop iterations, the loop counting would be error free.

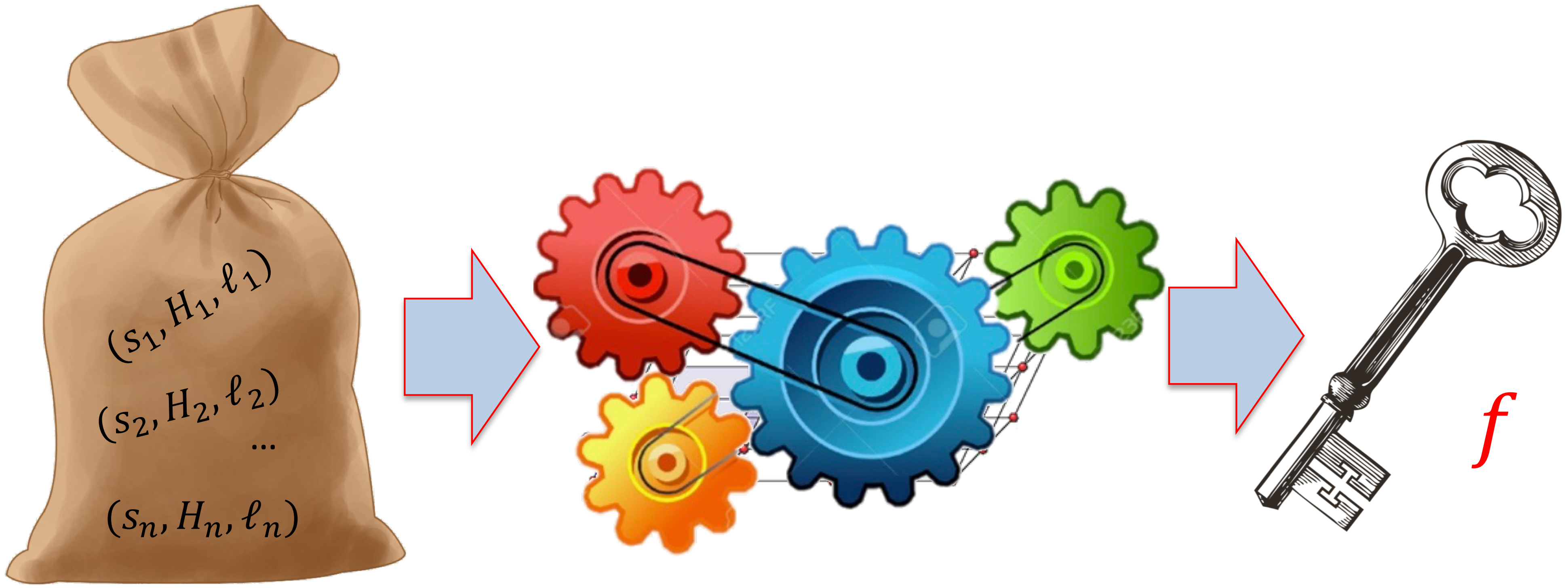
Analyzing the data

- A 49-loop period $= r_f$ with 7 MSB $= 0$.

Probability: $\frac{1}{2^7} \Longrightarrow$ many samples needed to get one signature with such a nonce.

- To reduce the number of observations, we can do some manual verification.
- Return traces where 2 or more counters agree.
- Introduces some error $\Longrightarrow$ manual post-processing needed.

The road ahead



A lattice attack

From the signing algorithm:

$$s_f = r_f + Hf \bmod p$$

with s_f, H public and p is a 256-bit order of an elliptic curve.

Side channel $\longrightarrow$ information about the length of r_f.

Goal: Solve for the secret key f.

$\Rightarrow$ hidden number problem

The hidden number problem (HNP) [BV96]

- Goal: recover some secret α

- Attacker has many samples from the ℓ MSB of random multiples of $\alpha \bmod p$.

- Given prime p and a fixed ℓ ($\approx \sqrt{\log p}$), recover the secret α in polynomial time with probability $\geq \frac{1}{2}$, under the assumption that

$$|\alpha t_i - u_i| \leq \frac{p}{2^\ell}.$$

t_i: uniformly and independently randomly chosen integers in Z_p^*.

u_i: integers representing the knowledge of the MSB of $\alpha t_i \bmod p$.

Applications of the HNP

- Boneh and Venkatesan: prove the existence of hardcore bits for the Diffie-Hellman key exchange [BV96].
- Nguyen and Shparlinski: attack DSA and ECDSA signing algorithms [NS02, NS03].
- Many attacks on implementations of the (EC)DSA algorithm.

Converting our problem to HNP

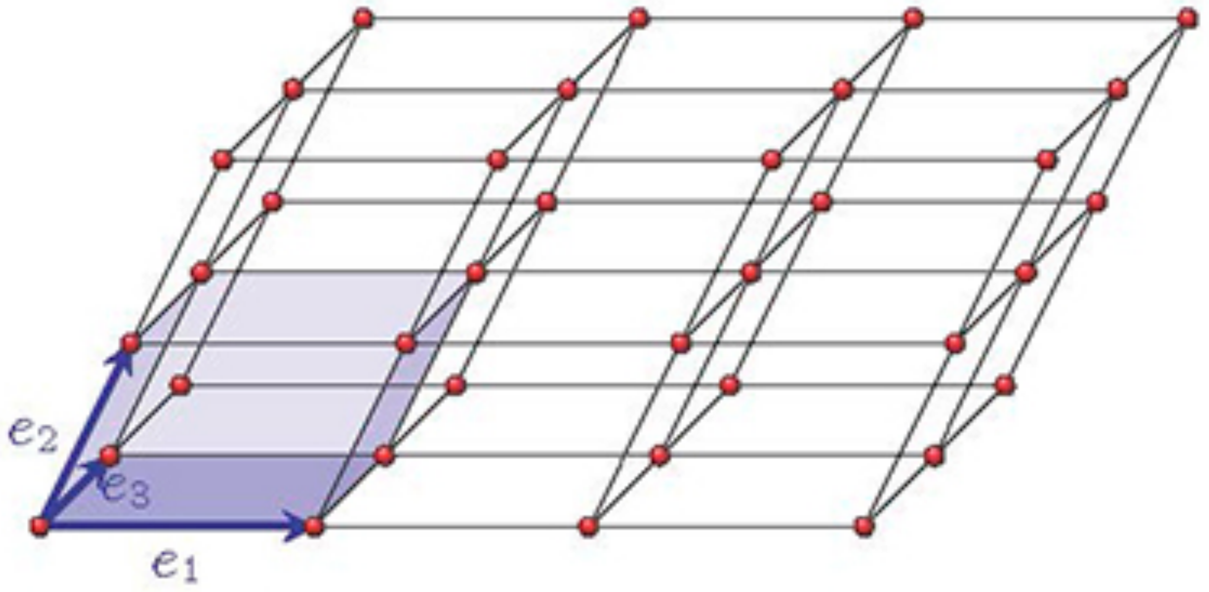
- In our attack, we get many samples $\{(s_f, H)\}_i$ which satisfy:

$$s_f \equiv r_f + Hf \bmod p$$

- And information about the most significant zero bits ℓ in r_f.

$$\Rightarrow \quad |s_i - H_i f| = |r_i| \leq \frac{p}{2^\ell}$$

Lattices



- A lattice L is a discrete additive subgroup of R^m.

- Any m-dimensional lattice can be specified by a basis of at most m linearly independent vectors.

- A basis of a lattice is represented as a matrix whose rows are the basis vectors.

Closest Vector Problem (CVP)

- Given a lattice L and a target point x, CVP asks to find the lattice point closest to the target.

- Many applications of the CVP only require finding a lattice vector that is not too far from the target, even if not necessarily the closest.

- Solving CVP in a special lattice will give a solution to the hidden number problem.

CVP embedding

$$\begin{bmatrix}
2^{\ell_1+1}p & \dots & & & & 0 & 0 \\
0 & 2^{\ell_2+1}p & & & & 0 & \vdots \\
 & & \ddots & & & & \\
 & & & 2^{\ell_d+1}p & & 0 & \\
2^{\ell_1+1}H_1 & 2^{\ell_2+1}H_2 & 2^{\ell_3+1}H_3 & \dots & \ddots & \frac{X}{p} & 0 \\
2^{\ell_1+1}s_1 & 2^{\ell_2+1}s_2 & 2^{\ell_3+1}s_3 & \dots & & 0 & X
\end{bmatrix}$$

Target vector for CVP

Shortest vector: $(2^{\ell_1+1}z_1, \ldots, 2^{\ell_d+1}z_d, f, -p)$

Recentering the nonces

- l_i: positive value $\Rightarrow$ lattice construction allows negative values too.

- Recenter the r_i around zero.

- Length of $r_i \leq 2^{n_i}$, $\quad n_i = 256 - l_i$
- Rewrite:

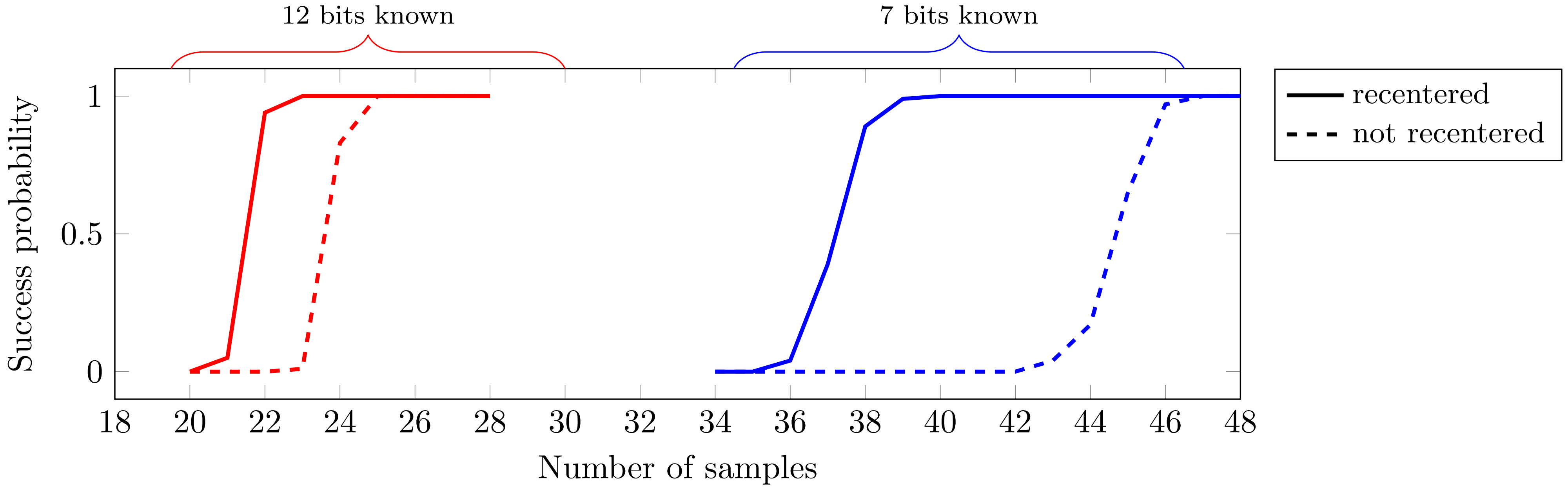
$$s'_i - r'_i \equiv H_i f \bmod p$$

With: $s'_i = s_i - 2^{n_i}$,

$$r'_i = r_i - 2^{n_i},$$

- New problem: $-\frac{p}{2^{l_i+1}} \leq r'_i \leq \frac{p}{2^{l_i+1}}$

Effect of recentering the nonces



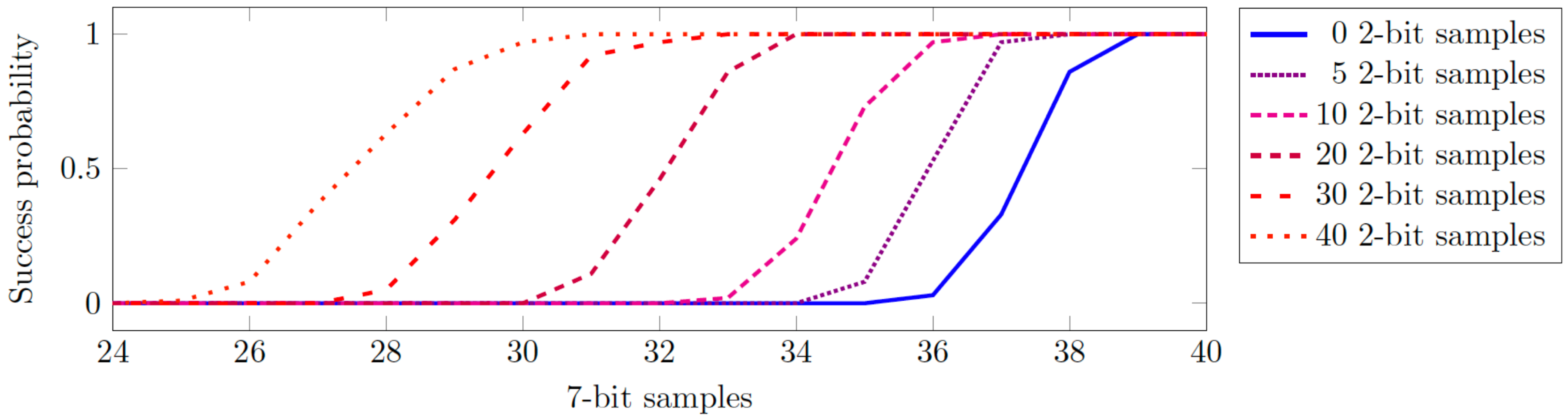
Recentering the nonces has a noticeable impact on the number of samples required for key recovery.

Samples of different lengths

Performance tradeoff: decrease the signature sampling time vs increasing time spent running lattice basis reduction.

- Probability of having a signature such that r_f has ℓ MSB equal to 0 is $\frac{1}{2^\ell}$
- The higher number of bits required, the more observations we need.
- Use loops with less bits (e.g. 2-bit samples) to reduce the sampling time.
- Less bits means more samples to recover the key, so higher dimension lattice.

Example



- Using only 49-loop samples in the lattice, i.e. learning 7 most significant bits of the nonce, we need 38 samples to achieve above a 50% success rate in the lattice construction (blue line).

We can reduce the total number of samples we need to collect by including samples that had revealed 2 bits of the nonce.

Error correction

- Quite common in a side-channel attack to have errors during the collection process.

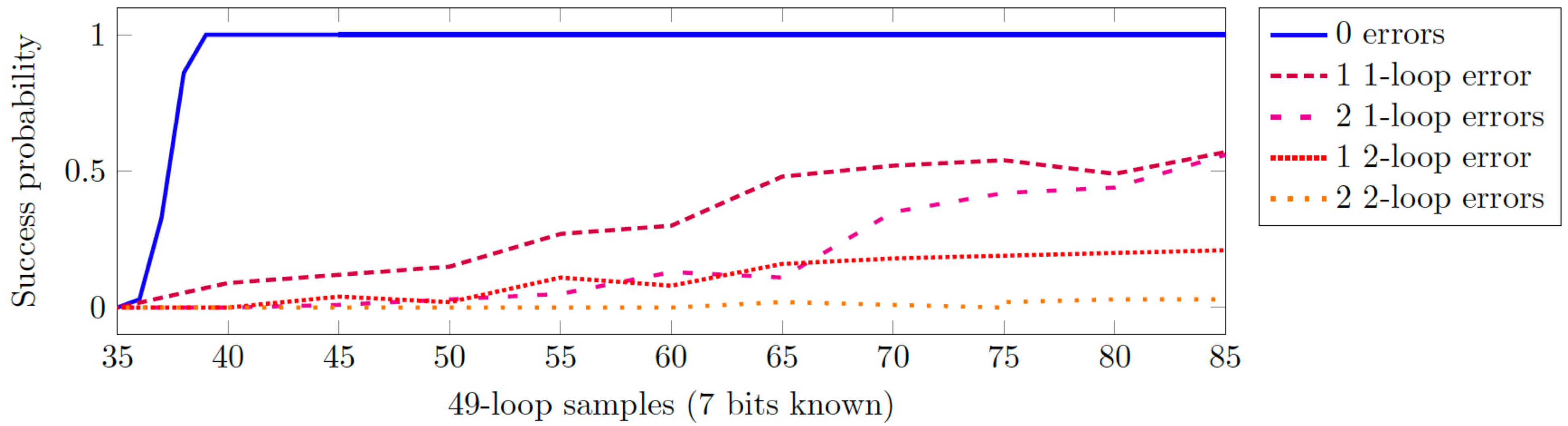
- Error => incorrect bound on the size of the r_i

- Problem when undercounting the number of loops => lattice construction fails in this case

Prior work

- Ignore the issue entirely
- Use signal processing
- Subsample different subsets of samples until we get an error-free sample

Error analysis



When the lattice includes more samples than necessary, key recovery may still be possible in the presence of errors.

In our measurements, an error corresponds to an incorrect loop count.

Recovering *f*

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

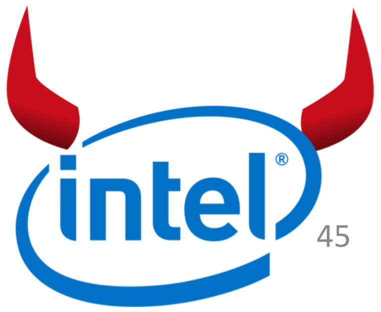
Signatures	48-loop	49-loop	50-loop	BKZ block size	BKZ time
10300	2	35	0	2	0.1s
10000	2	31	10	20	0.2s
9000	2	29	21	30	1.4s
8000	2	25	35	30	4.5s

- 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 !

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