Side Channel Attack on OpenSSL ECDSA

Joop van de Pol Nigel Smart Yuval Yarom

Outline

- Background
  - ECDSA
  - wNAF scalar multiplication
  - The Hidden Number Problem
  - The Flush+Reload technique
- Exploiting the side channel
- Recovering the key
- Results

ECDSA

Signer has a private key $1<\alpha<q-1$ and a public key $Q=[\alpha]G$

1. Compute $h=\mathrm{Hash}(m)$
2. Randomly select an ephemeral key $1<k<q$
3. Compute $(x,y)=[k]G$
4. Take $r=x \bmod q$; If $r=0$ repeat from 2
5. Take $s=(h+r\cdot\alpha)\cdot k^{-1} \bmod q$; if $s=0$ repeat from 2
6. (r,s) is the signature

Note that $k=\left|\left(r\cdot s^{-1}\right)\cdot\alpha+\left(h\cdot s^{-1}\right)\right|_q$

wNAF Form

To compute $[k]G$, first write k in wNAF form:

$$k = \sum_{i=0}^{n-1} d_i 2^i \text{ for } d_i \in \left\{0, \pm 1, \pm 3, \ldots, \pm(2^w - 1)\right\}$$

Such that if $d_i \neq 0$ then $d_{i+1} = \ldots = d_{i+w+1} = 0$.

Scalar Multiplication with wNAF form

Precompute $\{\pm G, \pm[3]G, \ldots, \pm[2^w\text{-}1]G\}$

x=0
for i=n-1 **downto** 0
  x = Double(x)
  if ($d_i \neq 0$) **then**
    x = Add(x, $[d_i]G$)
  end
end
return x

The Hidden Number Problem [BV96]

We know enough triples t_i, u_i and z_i such that

$$v_i = \left|\alpha t_i - u_i\right|_q < q / 2^{z_i}$$

for an unknown α.

We can find α by reducing HNP to a lattice problem.

.

HNP and ECDSA [HGS01,NS03]

Recall that $k = \left| \left(r \cdot s^{-1} \right) \cdot \alpha + \left(h \cdot s^{-1} \right) \right|_q$

We want $\left| \alpha t_i - u_i \right|_q < q / 2^{z_i}$

In terms of k:

Or in terms of t_i and u_i:

$$\left| \left(r \cdot s^{-1} \right) \cdot \alpha + \left(h \cdot s^{-1} \right) \right|_q = k$$

HNP and ECDSA [HGS01,NS03]

Recall that $k = \left| \left(r \cdot s^{-1} \right) \cdot \alpha + \left(h \cdot s^{-1} \right) \right|_q$

We want $\left| \alpha t_i - u_i \right|_q < q / 2^{z_i}$

In terms of k:

Or in terms of t_i and u_i:

$$\left| \left(r \cdot s^{-1} \right) \cdot \alpha - \left(-\left(h \cdot s^{-1} \right) \right) \right|_q < q$$

HNP and ECDSA [HGS01,NS03]

Recall that $k = \left| \left(r \cdot s^{-1} \right) \cdot \alpha + \left(h \cdot s^{-1} \right) \right|_q$

We want $\left| \alpha t_i - u_i \right|_q < q / 2^{z_i}$

In terms of k:

Or in terms of t_i and u_i:

$$\left| \left(r \cdot s^{-1} \right) \cdot \alpha - \left(a - \left(h \cdot s^{-1} \right) \right) \right|_q < q$$

HNP and ECDSA [HGS01,NS03]

Recall that $k = \left| \left(r \cdot s^{-1} \right) \cdot \alpha + \left(h \cdot s^{-1} \right) \right|_q$

We want $\left| \alpha t_i - u_i \right|_q < q / 2^{z_i}$

In terms of k:

$$\left(k - a \right) / 2^{z_i}$$

Or in terms of t_i and u_i:

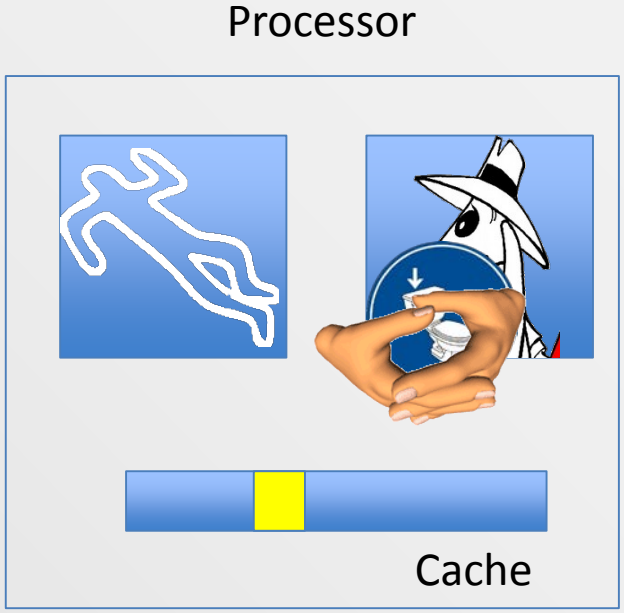
$$\left| \left(\left(r \cdot s^{-1} \right) \cdot \alpha - \left(a - \left(h \cdot s^{-1} \right) \right) \right) / 2^{z_i} \right|_q < q / 2^{z_i}$$

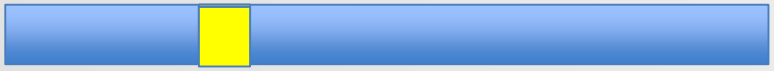
HNP and ECDSA – State of the Art

- Useful information:
  - l bits for l known LSBs
  - Between l-1 and l bits for l known MSBs
  - $l/2$ bits for arbitrary l consecutive bits

FLUSH+RELOAD [YF14]

- A cache-based side-channel attack technique
- **FLUSH** memory line
- Wait a bit
- Measure time to **RELOAD** line
  - fast-> access
  - slow-> no access
- Repeat





Outline

- Background
  - ECDSA
  - wNAF scalar multiplication
  - The Hidden Number Problem
  - The Flush+Reload technique
- Exploiting the side channel
- Recovering the key
- Results

Attacking OpenSSL wNAF

- Use FLUSH+RELOAD to recover the double and add chain of the wNAF calculation
  - Divide time into slots of 1200 cycles (about 0.4μs)
  - In each slot, probe a memory line in the code of the Double and Add functions.

x=0
for i=n-1 **downto** 0
  x = Double(x)
  if ($d_i \neq 0$) **then**
    x = Add(x, $[d_i]G$)
  end
end
return x

Sample Trace

Raw:

D|||D|D
D			D	
		D	D	
D			D	
	D			D
		D		…

||D||D|||A|A|||D|||D|||
D|D||D|||D|||D||A|A|||D
||D||D|||D|||A|A|||D||
D|||D|||A|||D|||D|D|
D|D||D|||D|||D|||A|||D|D

Errors occur in approx. 1 symbol in 1000.

Processed:

DDDADDDDADDDDDDADDDDDADDDDADDDDDADDDDADDDDADDDDDADDD
DDDDADDDDADDDDADDDDDADDDDADDDDDDDADDDDDDDADDDDADDDDAD
DDDADDDDADDDDADDDDDADDDDDADDDDADDDDADDDDADDDDADDDDAD
DDDADDDDDDDADDDDDADDDDADDDDDDADDDDADDDDDDDADDDDDADDDD
ADDDDDADDDDDADDDDDADDDDDADDDDXDDDADDDDADDDDADDDDADDD
DDADDDDADDDDDDADDDDDADDDDADDDDDDDADDDDDDDADDDDADD

Sample Trace

We know how to use the revealed LSBs

Processed:

DDDADDDDADDDDDDADDDDDADDDDADDDDDADDDDADDDDADDDDDADDD
DDDDADDDDADDDDADDDDDADDDDADDDDDDDADDDDDDADDDDADDDDAD
DDDADDDDADDDDADDDDDADDDDDADDDDADDDDADDDDADDDDADDDDAD
DDDADDDDDDDADDDDDADDDDADDDDDDADDDDADDDDDDADDDDDADDDD
ADDDDDADDDDDADDDDDADDDDDADDDXDDDADDDDADDDDADDDDADDD
DDADDDDADDDDDDADDDDDADDDDADDDDDDADDDDDDADDDDADD

Sample Trace

We know how to use the revealed LSBs

But these give an average of 2 bits per observed signature.

Can we use the information about the MSBs?

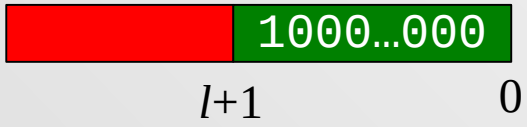
Processed:

DDDADDDDADDDDDDADDDDDADDDDADDDDDADDDDADDDDADDDDDADDD
DDDDADDDDADDDDADDDDDADDDDADDDDDDDADDDDDDADDDDADDDDAD
DDDADDDDADDDDADDDDDADDDDDADDDDADDDDADDDDADDDDADDDDAD
DDDADDDDDDDADDDDDADDDDADDDDDDADDDDADDDDDDADDDDDADDDD
ADDDDDADDDDDADDDDDADDDDDADDDDXDDDADDDDADDDDADDDDADDD
DDADDDDADDDDDDADDDDDADDDDADDDDDDADDDDDDADDDDADD

Using the MSBs

Assume $d_{m+l}, d_m \neq 0$

Before adding $[d_m]G$, x is:

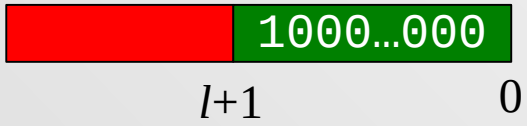


$x=0$
for $i=n$-1 **downto** 0
  x = Double(x)
  if ($d_i \neq 0$) **then**
    x = Add(x, $[d_i]G$)
  end
end
return x

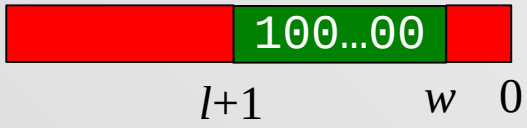
Using the MSBs

Assume $d_{m+l}, d_m \neq 0$

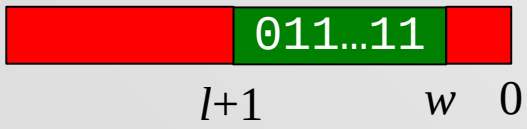
Before adding $[d_m]G$, x is:



After adding $[d_m]G$, for $d_m > 0$ it is



And for $d_m < 0$

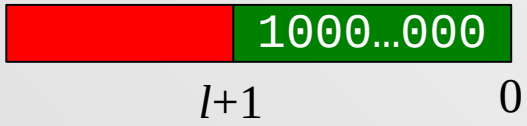



```
x=0
for i=n-1 downto 0
  x = Double(x)
  if (d_i≠0) then
    x = Add(x, [d_i]G)
  end
end
return x
```

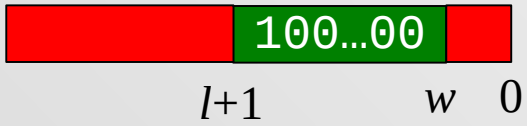
Using the MSBs

Assume $d_{m+l}, d_m \neq 0$

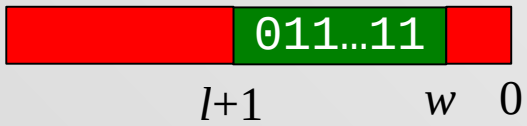
Before adding $[d_m]G$, x is:



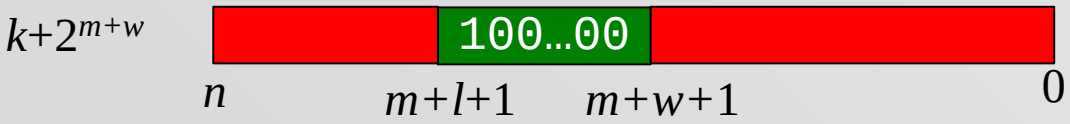
After adding $[d_m]G$, for $d_m > 0$ it is



And for $d_m < 0$



Either way,




```
x=0
for i=n-1 downto 0
  x = Double(x)
  if (d_i≠0) then
    x = Add(x, [d_i]G)
  end
end
return x
```

Observation

For many “standard” curves, q, the group order is close to a power of two. That is, $q=2^n-\varepsilon$ such that $|\varepsilon|<2^p$ for $p \ll n$.

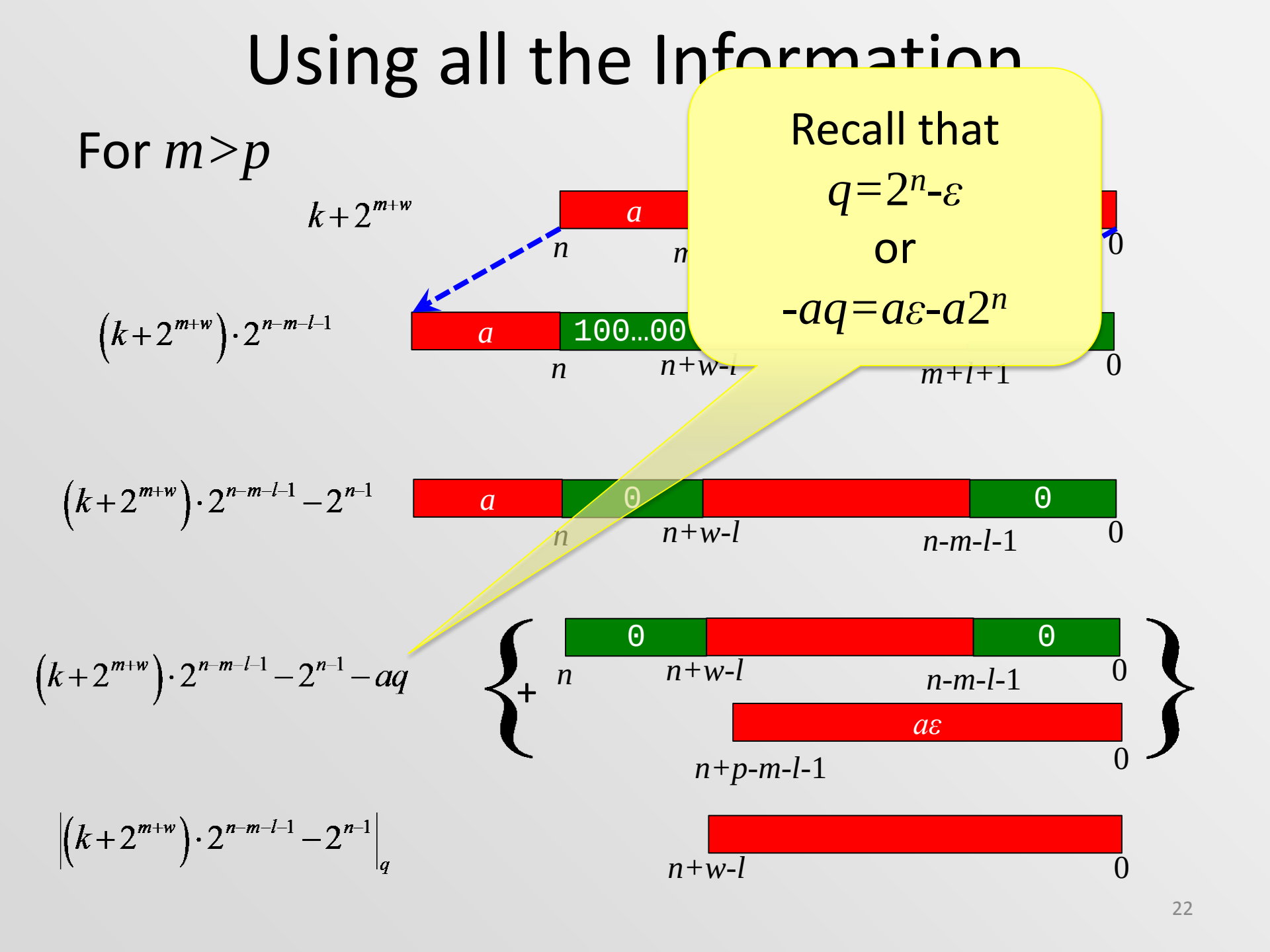
For example for **secp256k1**

q=FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE
EBAAEDCE6AF48A03BBFD25E8CD0364141

Adding or subtracting q to an n bit number is unlikely to change the MSBs

Using all the Information

For $m>p$



Summary

- FLUSH+RELOAD provides a nearly perfect side channel
  - Probability of error 1/1000 symbols
  - 99% of errors are noticed
  - 2 out of every 3 observed traces are perfect
- Curve choice allows using almost half of the information in each perfect trace

Results

Previous results:

- [LN13]: 160-bit key, 100 signatures with 2 known bits
- [BPSY14]: 256-bit key, expected 200 signatures

Our results:

# perfect traces	Time (s)	Success probability
10	2.25	0.07
11	4.66	0.25
12	7.68	0.38
13	11.3	0.54

With a very high probability, observing 25 signatures yields more than 13 perfect traces.

Cache storage attacks

CT-RSA 2015

B. B. Brumley

Department of Pervasive Computing
Tampere University of Technology, Finland
billy.brumley AT tut.fi

21 Apr 2015

TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

Covert channels

Rainbow Series: Light Pink Book

A communication channel is covert if it is neither designed nor intended to transfer information at all.

Covert Channel Analysis of Trusted Systems (NCSC-TG-030)

Covert timing channels

A potential covert channel is a timing channel if its scenario of use involves a process that "signals information to another by modulating its own use of system resources (e.g., CPU time) in such a way that this manipulation affects the real response time observed by the second process."

Covert storage channels

A potential covert channel is a storage channel if its scenario of use "involves the direct or indirect writing of a storage location by one process and the direct or indirect reading of the storage location by another process."

Crypto and side-channel attacks

High level: one party is legit, the other isn't

Timing attacks

- D. Page (DES, 2002)
- D. J. Bernstein (AES, 2005)
- C. Percival (RSA, 2005)
- Osvik et al. (AES, 2005)
- Neve and Seifert (AES, 2006)
- B. B. Brumley and R. Hakala (ECDSA, 2009)
- Aciiçmez et al. (DSA, 2010)
- B. B. Brumley and N. Tuveri (ECDSA, 2011)
- Yarom et al. (ECDSA, 2014)
- ...

Storage attacks

Data caching

```
idx   tag  data
--- ------ ----
  0 de30ec ????
  1 096324 ????
  .      .    .
  .      .    .
  F 61eff8 ????
```

Cache-timing attacks

Time access to own data and infers cache hits / misses to determine victim state that caused the eviction.

Cache debugging and HW privilege separation

Invasive cache debug

- Direct reads to cache lines and metadata
- Cache footprint profiling
- HW errata or coherency

Access control

- Cannot allow unchecked direct access to cache lines (exception)
- x86 protection mode: Ring 0/3
- ARM ft. TrustZone: NS-bit

Data caching with HW privilege separation

Note priv 0 can evict priv 1 and vice versa.

idx	priv	tag	data
0	1	de30ec	????
1	0	096324	????
.	.	.	.
.	.	.	.
F	1	61eff8	????

The covert storage channel (1)

Alice (priv 0) wants to send one nibble (`0x2`) to Bob (priv 1).

Bob pollutes:

```
idx priv   tag  data
--- ---- ------ ----
  0    1 de30ec ????
  1    1 096324 ????
  2    1 ce5f84 ????
  .    .      .    .
  .    .      .    .
  F    1 61eff8 ????
```

The covert storage channel (2)

Alice does one read to evict one line (`0x2`):

```
idx priv   tag  data
--- ---- ------ ----
  0    1 de30ec ????
  1    1 096324 ????
  2    0 b7d710 ????
  .    .      .    .
  .    .      .    .
  F    1 61eff8 ????
```

The covert storage channel (3)

Bob then:

1. Reads from Line 0. No exception.
2. Reads from Line 1. No exception.
3. Reads from Line 2. Exception – receives `0x2`.

```
idx priv   tag  data
--- ---- ------ ----
  0    1 de30ec ????
  1    1 096324 ????
  2    0 b7d710 ????
  .    .      .    .
  .    .      .    .
  F    1 61eff8 ????
```

From cache storage to clean cache-timing traces

1. Read directly from a cache line. A processor exception indicates `M`, otherwise `H`.
2. If `M` go back to the first step. This requires another query because the processor exception most like wipes the cache state and/or triggers a reset.
3. If `H` continue with the next line.

Example

`HMHHHHMHHHHHHMHHH`

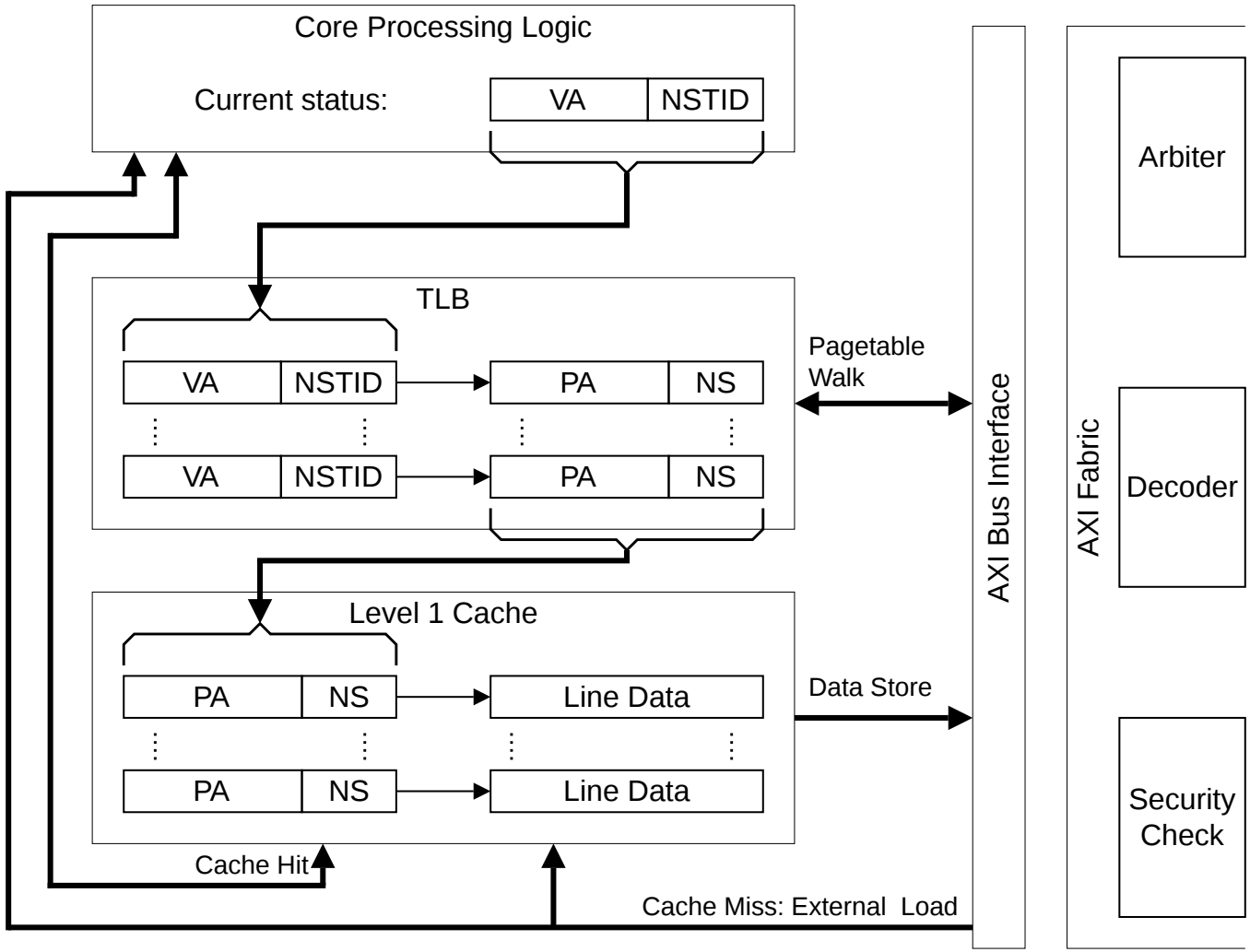
A practical architecture (1)

SRC: ARM Security Technology documentation

The content of the caches, with regard to the security state, is dynamic. Any non-locked down cache line can be evicted to make space for new data, regardless of its security state. It is possible for a Secure line load to evict a Non-secure line, and for a Non-secure line load to evict a Secure line.

A practical architecture (2)

SRC: ARM Security Technology documentation



A practical architecture (3)

SRC: ARM1156T2-S Technical Reference Manual

The purpose of the data cache Tag RAM operation is to:

- read the data cache Tag RAM contents and write into the Data Cache Debug Register.
- write into the Data Cache Debug Register and write into the Data Tag RAM.

To read the Data Tag RAM, write CP15 with:

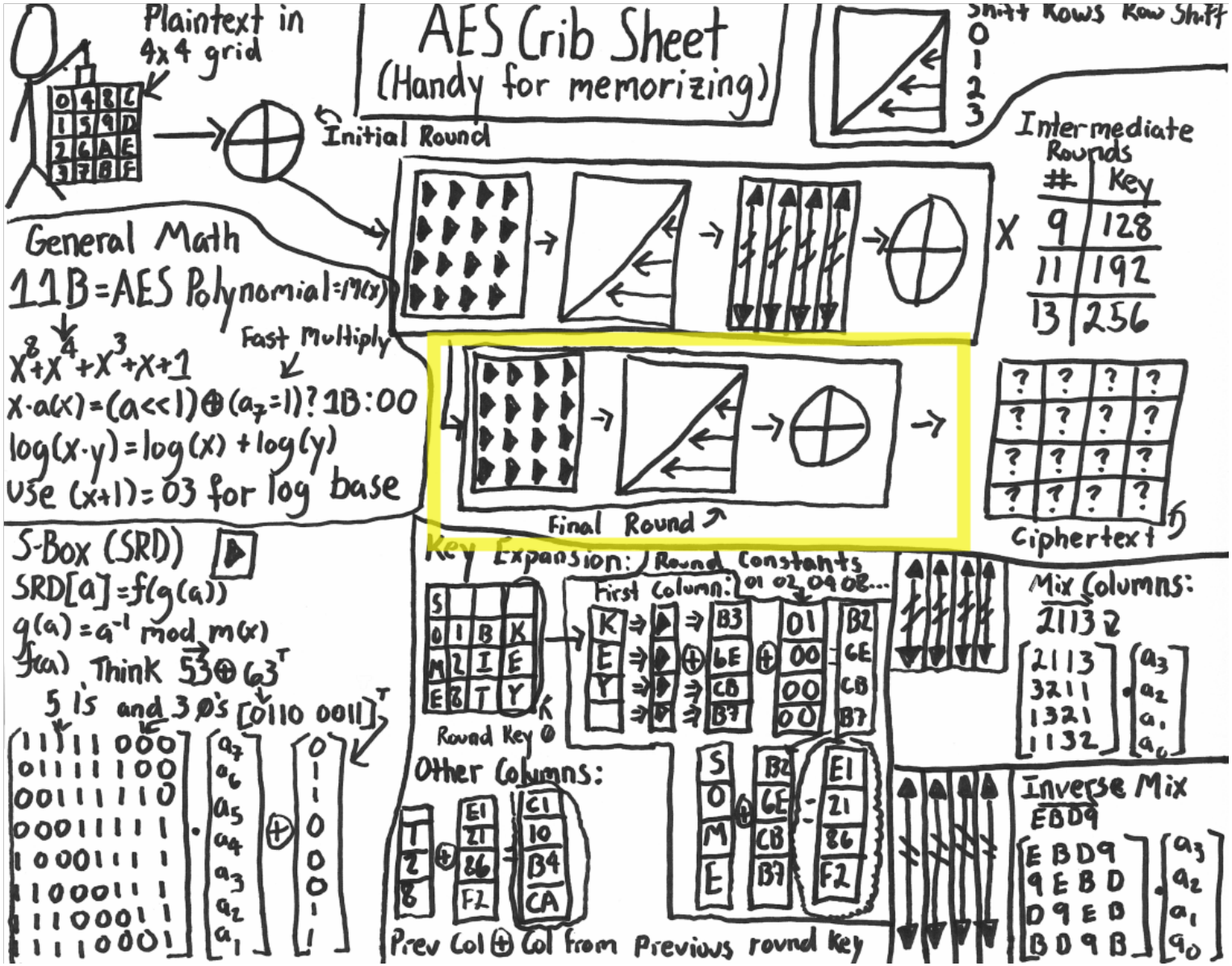
```
; Data Tag RAM read operation
MCR p15, 3, <Rd>, c15, c2, 0
```

Transfer data to the Data Cache Debug Register to the core:

```
; Read Data Cache Debug Register
MRC p15, 3, <Rd>, c15, c0, 0
```

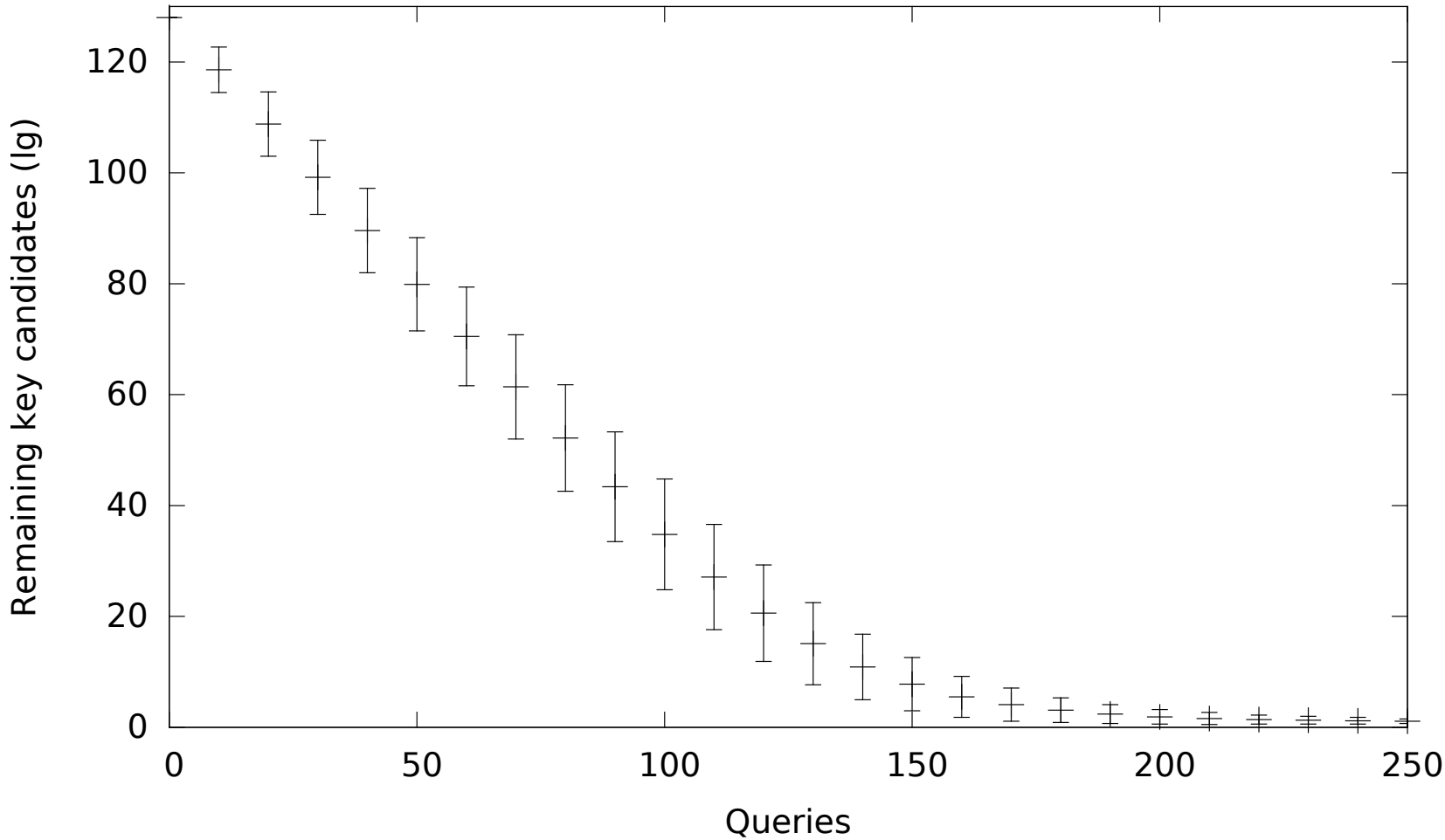
AES and side-channels

SRC: Jeff Moser



AES cache storage attack simulation

Borrowed cache-timing attack by Neve and Seifert (2006)



Conclusion

- Cache debugging mechanisms and hardware-enforced privilege separation can create covert channels
- Can potentially put crypto keys at risk
- Compared to cache-timing attacks:
  - Each "cache miss" costs an additional query
  - But the traces themselves are much more accurate
- For ARM ft. TrustZone, attack would be from Normal World kernel space to Secure World
- Concrete mitigation: implementation defined instructions
- Thanks! Questions?