Heninger-Shacham

RSA

- Secret random primes p and q
- Calculate public $N = pq$

- For all r, $N = pq \bmod r$
  - Specifically for all m, $N = pq \bmod 2^m$

- We know the LSB of p and q
  - Can we guess the next bit?

- What if we know the m LSBs? Can we guess the next bit?

Guessing bits

Let's try

$N=$ 1 0 0 1 0 1 0 1 1 0 1 0 1 1 1 0 1 0 1

$p=$ x x x x x x x x x 1

x x 1

But what if we
know th

- No known bits ⇒ Guess one to determine other
- One known ⇒ determine other
- Two known ⇒ rule out prior wrong guesses

$p=$ x x x x … x x x 1 1

$q=$ x x x x … x x x 1 1

$p=$xxxxxxx001

$q=$xxxxxxx101

$q=$xxxxxxx001

…1

$q=$xxxxxxx111

$p=$xxxxxxx111

$q=$xxxxxxx011

2^n Guesses ☹

RSA

- Random primes p and q
- Calculate $N = pq$
- Select a public exponent e(=65537)
- Compute $d=e^{-1} \bmod \varphi(N)$
- Encrypt: $C=M^e \bmod N$
- Decrypt: $M=C^d \bmod N$

Small e → fast computation

Large d → slow computation

CRT-RSA

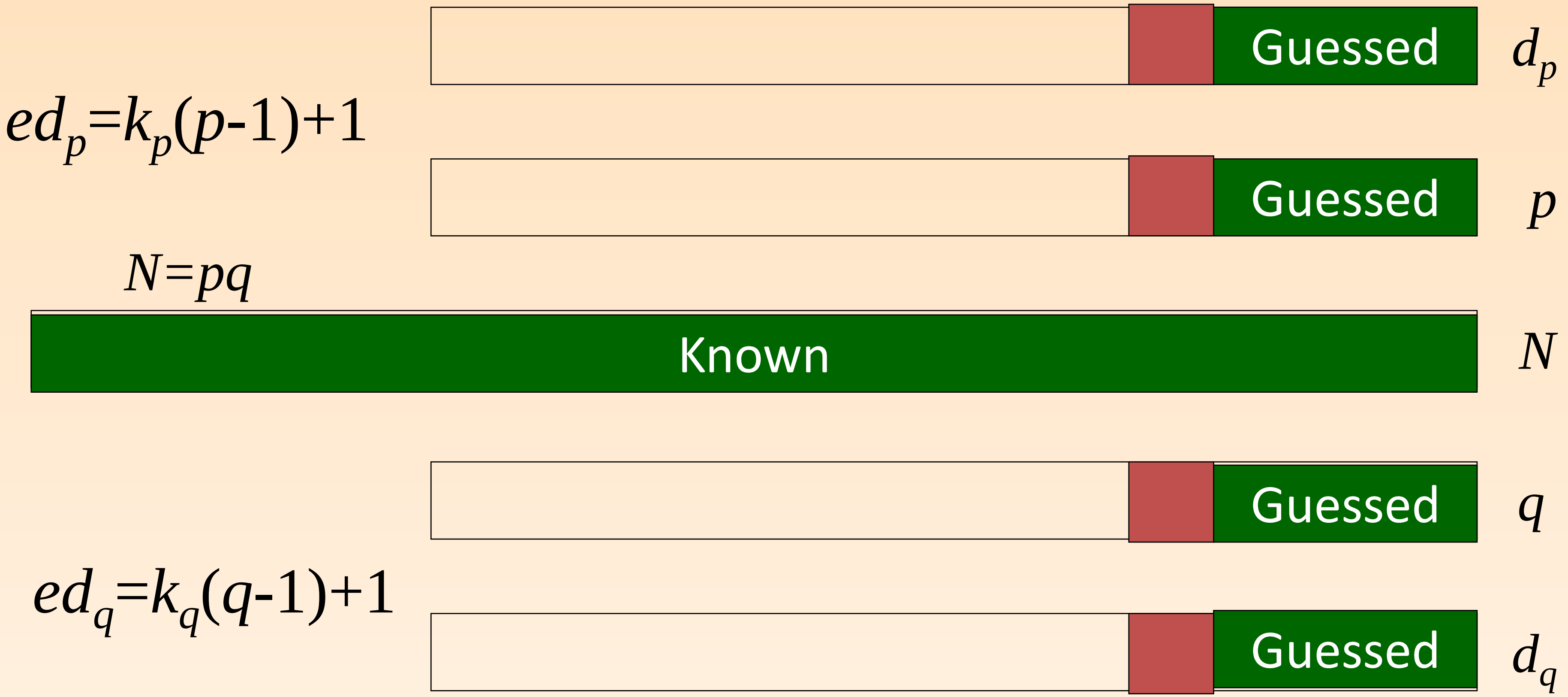
- $d_p = d \bmod (p\text{-}1)$, $d_q = d \bmod (q\text{-}1)$
- $m_p = m^{d_p} \bmod p$, $m_q = m^{d_q} \bmod q$
- $h = q\text{-}1(m_p\text{-}m_q) \bmod p$
- $m = m_q + hq$

- Hence:
  - $N = pq$
  - $ed_p = k_p(p\text{-}1)+1$
  - $ed_q = k_q(q\text{-}1)+1$

- Moreover, $0 < k_p, k_q < e$, and they are all related

Heninger-Shacham (CRYPTO 2009)

- A technique for finding the RSA-CRT key from partial information



Sliding Window Exponentiation

- Represent the exponent d in a convenient form:

  $d = \Sigma d_i 2^i$ where d_i is either 0 or is odd $0<d_i<2^w$
- Precompute odd powers of the base b

  $b[i]=b^i \bmod p$ for odd $0<i<2^w$

$B[1]= b$

$b_{sqr} = b^2 \bmod p$

for $i = 3, 5, \ldots, 2^w\text{-}1$ **do**

  $b[i] = b[i\text{-}2]\cdot b_{sqr} \bmod p$

Sliding Window Exponentiation

- Perform the exponentiation

$r \leftarrow 1$

for $i = |d|-1, \ldots, 0$ **do**

$\quad r \leftarrow r^2 \bmod p$

$\quad$ **if** $d_i \neq 0$ **then**

$\quad\quad r \leftarrow r \cdot b[d_i] \bmod p$

return r

Sliding window representation revisited

$d = \Sigma d_i 2^i$ where d_i is either 0 or is odd $0<d_i<2^w$

Another way of looking at this:

- Divide d into *windows*
  - Windows are at most w bits wide
  - Windows start and end with 1

$d = 4312 = \boxed{1}\ 0\ 0\ 0\ 0\ \boxed{1}\ \boxed{1\ 0\ 1\ 1}\ 0\ 0\ 0$

$d_i = \quad 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 11\ 0\ 0\ 0$

$\Sigma d_i 2^i = 1\cdot 2^{12} + 1\cdot 2^7 + 11\cdot 2^3 = 4096+128+11\cdot 8 = 4312$

Sliding window representation revisited

$d = \Sigma d_i 2^i$ where d_i is either 0 or is odd $0<d_i<2^w$

Not a unique representation

$$d = 4312 = \boxed{1}\ 0\ 0\ 0\ 0\ \boxed{1}\ \boxed{1\ 0\ 1\ 1}\ 0\ 0\ 0$$

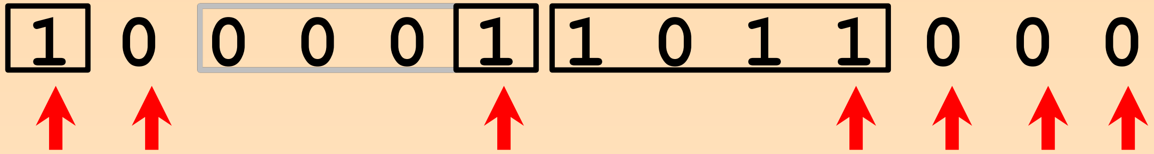
$$\boxed{1}\ 0\ 0\ 0\ 0\ \boxed{1\ 1\ 0\ 1}\ \boxed{1}\ 0\ 0\ 0$$

$$\boxed{1}\ 0\ 0\ 0\ 0\ \boxed{1}\ \boxed{1\ 0\ 1}\ \boxed{1}\ 0\ 0\ 0$$

Minimise the number of windows for best performance

A greedy algorithm

- Scan the bits of d until finding a 1
- *Open* a window of length w bits
- *Close* the window at the last 1
- Repeat



- The greedy algorithm is optimal

Analysis

- Successive open positions are at least w bits apart
- Assuming a random d

  $\boxed{1}$ 0 $\boxed{0\ 0\ 0}$ $\boxed{1}$ $\boxed{1\ 0\ 1\ 1}$ 0 0 0

  - w bits apart with probability $1/2$
  - $w+1$ bits apart with probability $1/4$
  - $w+2$ bits apart with probability $1/8$
  - Etc…
- On average, successive open positions are $w+1$ bits apart
  - Expected number of windows is $|d|/(w+1)$

Left-to-right vs. right-to-left

- Previously, we ran the greedy algorithm from the right to the left.

$\boxed{1}$ 0 0 0 0 $\boxed{1}$ $\boxed{1\ 0\ 1\ 1}$ 0 0 0

- Can also run from the left to the right

$\boxed{1}$ 0 0 0 0 $\boxed{1\ 1\ 0\ 1}$ $\boxed{1}$ 0 0 0

- The analysis still applies
  - We get the same number of windows

- Can combine left-to-right with exponentiation

Recovering the exponent

- We know the positions of the windows

$$\underline{x}\ x\ x\ x\ x\ \underline{x}\ x\ x\ x\ \underline{x}\ x\ x\ x$$

- Windows start with 1

$$\underline{1}\ x\ x\ x\ x\ \underline{1}\ x\ x\ x\ \underline{1}\ x\ x\ x$$

- Everything outside maximum window must be 0

$$\boxed{\quad\quad \underline{1}}\ 0\ \boxed{x\ x\ x\ \underline{1}}\boxed{x\ x\ x\ \underline{1}}\ 0\ 0\ 0$$

- On average we get 2 bits per window or $2|d|/(w+1)$ per exponentiation

Heninger-Shacham and the side-channel results

- A technique for finding the RSA-CRT key from partial information
- On average, needs one bit of p, q, d_p, d_q
- In our case we do not know bits of p or q
- For a successful attack we need to know half the bits of d_p and d_q
- We get 2 bits per window of size $w+1$
  - For $w>3$, we need more information

Left-to-right leaks more

- Right to left (d=8625):

$\boxed{1}$ 0 0 0 0 $\boxed{1}$ $\boxed{1\ 0\ 1\ 1}$ 0 0 0 $\boxed{1}$

$\underline{x}$ x x x x $\underline{x}$ x x x $\underline{x}$ x x x $\underline{x}$

$\underline{1}$ 0 x x x $\underline{1}$ x x x $\underline{1}$ x x x $\underline{1}$

- Left-to-right (d=8625):

$\boxed{1}$ 0 0 0 0 $\boxed{1\ 1\ 0\ 1}$ $\boxed{1}$ 0 0 0 $\boxed{1}$

$\underline{x}$ x x x x x x x $\underline{x}$ $\underline{x}$ x x x $\underline{x}$

$\underline{1}$ 0 0 0 0 $\boxed{x\ x\ x\ \underline{1}}$ $\underline{1}$ x x x $\underline{1}$

$\underline{1}$ 0 0 0 0 $\boxed{1\ x\ x\ \underline{1}}$ $\boxed{\underline{1}\ x\ x\ x}$ $\underline{1}$

$\underline{1}$ 0 0 0 0 1 x x $\underline{1}$ $\underline{1}$ 0 0 0 $\underline{1}$

Results

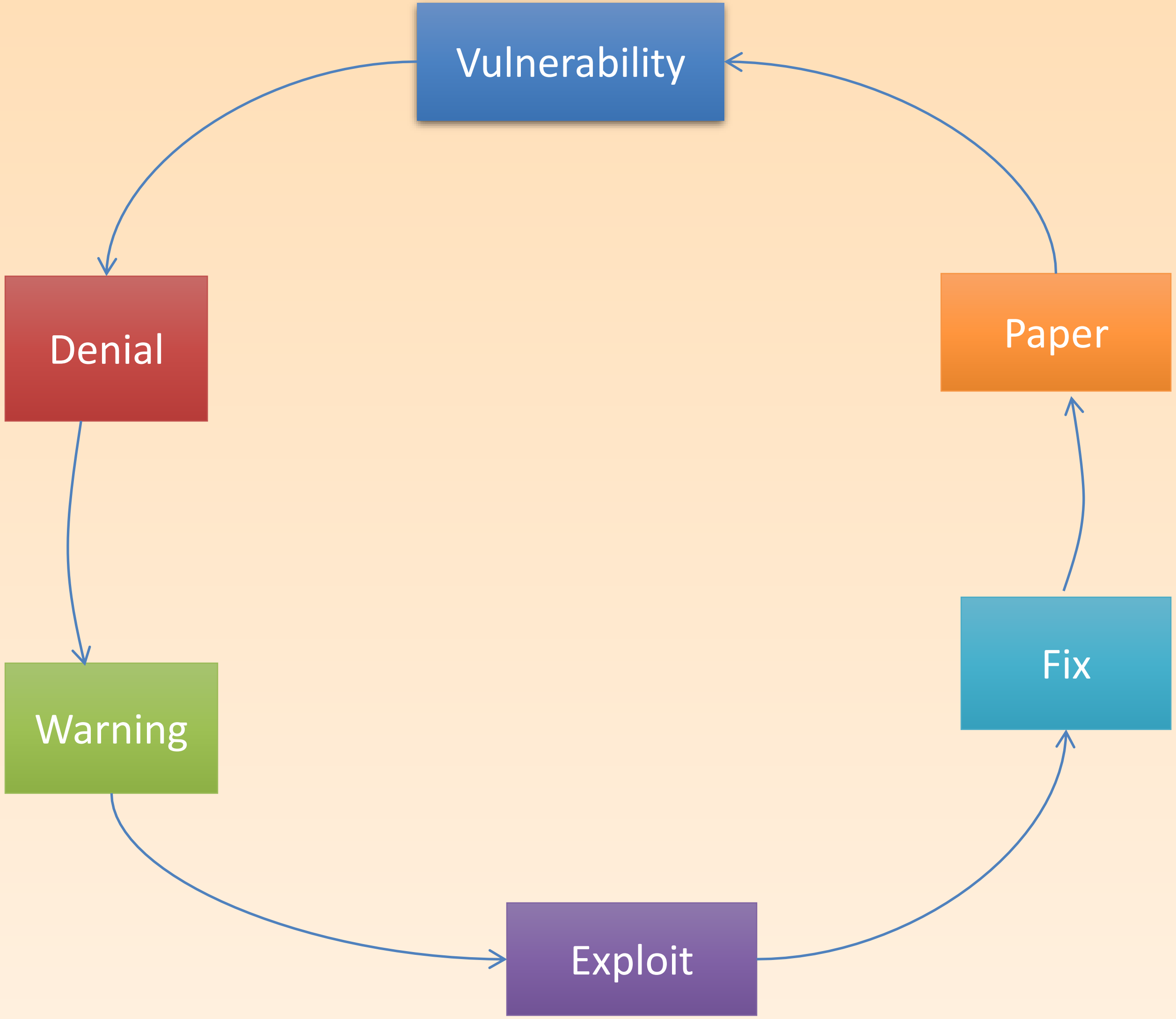
- For $w=4$ can get 2.725 bits per window
  - Break all 1024-bit RSA keys in Libgcrypt

- For $w=5$ can get 2.766 bits per window
  - Break 13% of the 2048-bit RSA keys

- For more information see

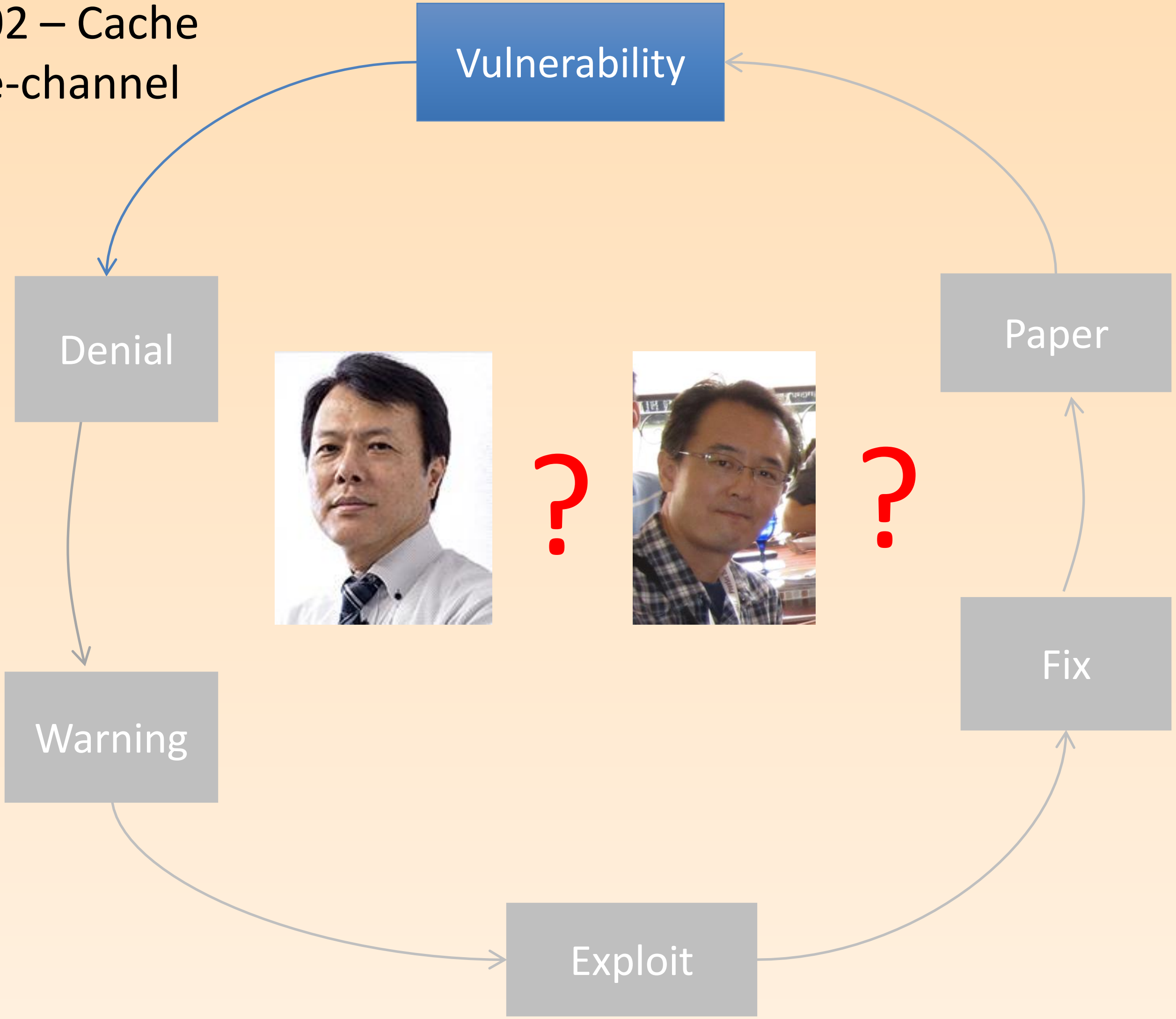
  Bernstein et al. “Sliding Right Into Disaster: Left-to-Right Sliding Windows Leak”, CHES 2017

Attack Life Cycle



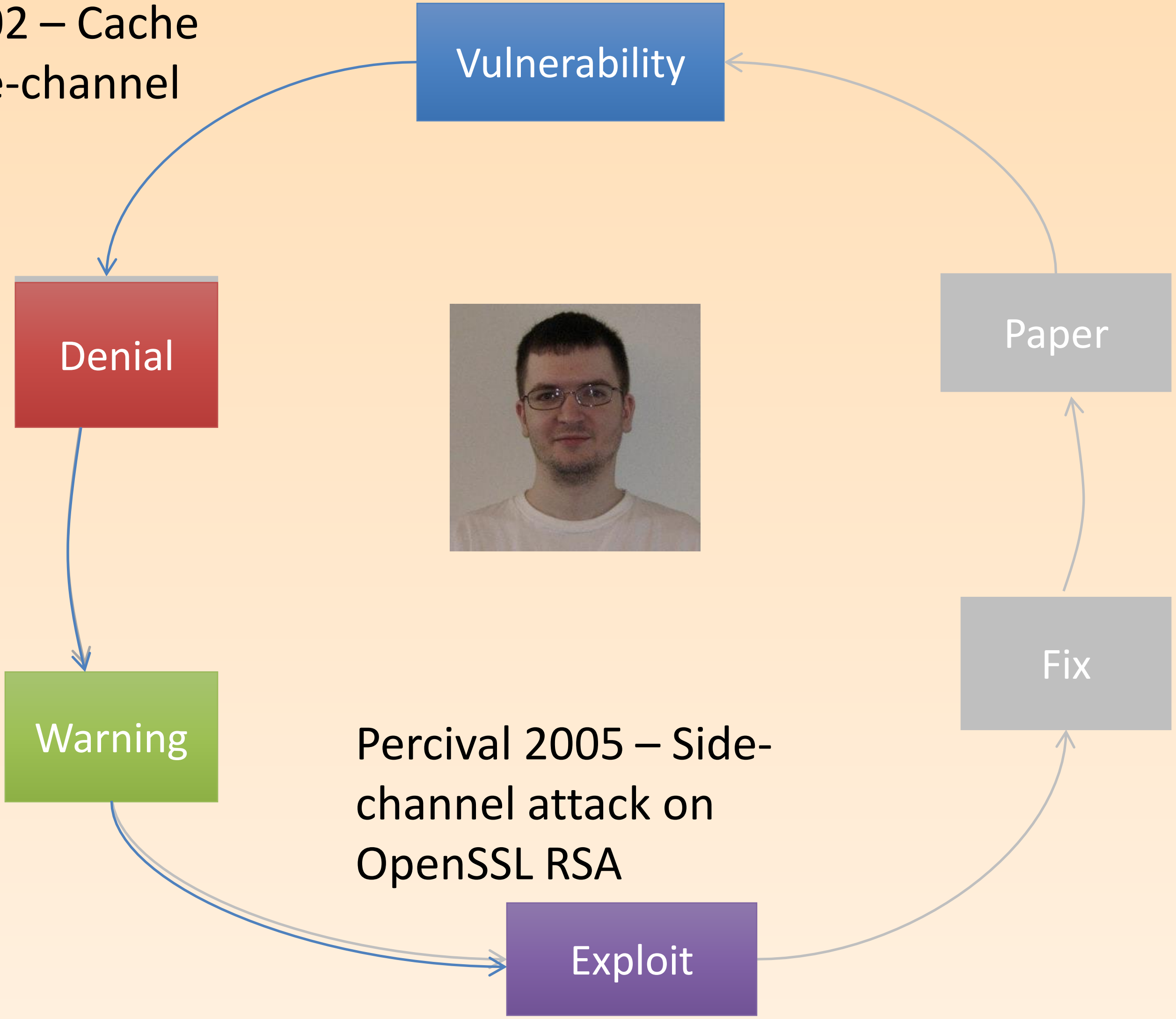
Round 1

Tsunoo, Tsujihara, Minematsu and Miyauchi 2002 – Cache may leak side-channel information



Round 1

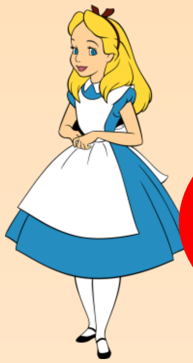
Tsunoo, Tsujihara, Minematsu and Miyauchi 2002 – Cache may leak side-channel information



Percival 2005 – Side-channel attack on OpenSSL RSA

The RSA Encryption System

- The RSA encryption is a public key cryptographic scheme



C

$M = C^d \bmod N$

M

$C = M^e \bmod N$

Key Generation:

- Select random primes p and q
- Calculate $N = pq$
- Select a public exponent $e(=65537)$
- Compute $d=e^{-1} \bmod \varphi(N)$
- (N, e) is the public key
- (p, q, d) is the private key

Fixed Window Exponentiation

- Divide exponent into *windows* of size w
- Precompute $b_i = b^i \bmod m$:

1 0	0 1 1 0	1 0 0 1	0 1 0 0
$d_3=2$	$d_2=6$	$d_1=9$	$d_0=4$

$b_0 \leftarrow 1$

$b_1 \leftarrow \mathrm{b}$

for $i = 2, 3, \ldots, 2^w\text{-}1$ **do**

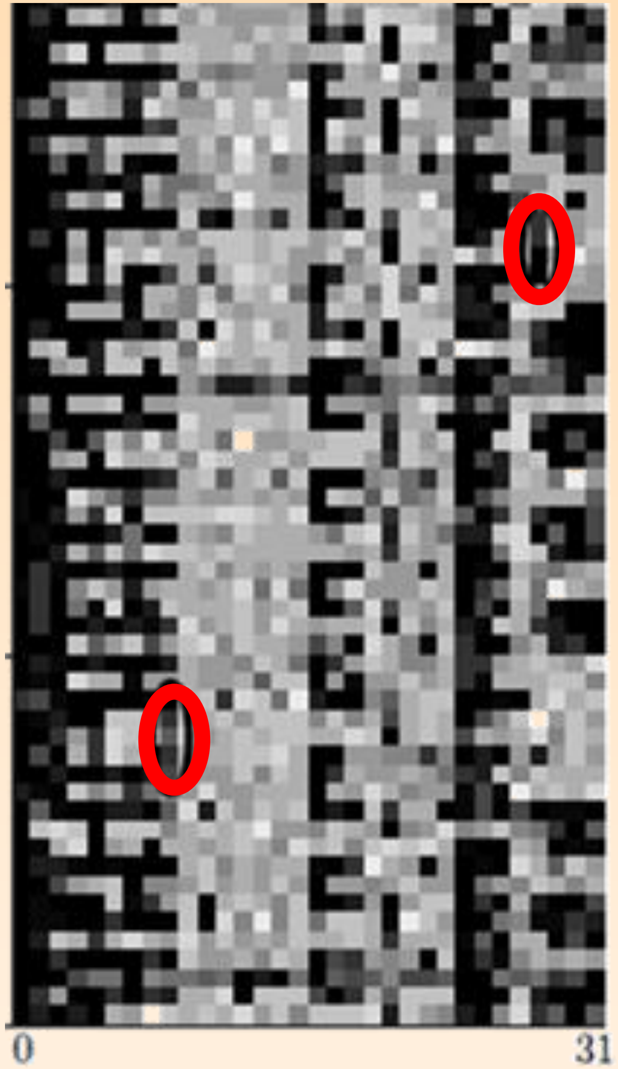
$\qquad b_i \leftarrow b_{i\text{-}1} \cdot b \bmod m$

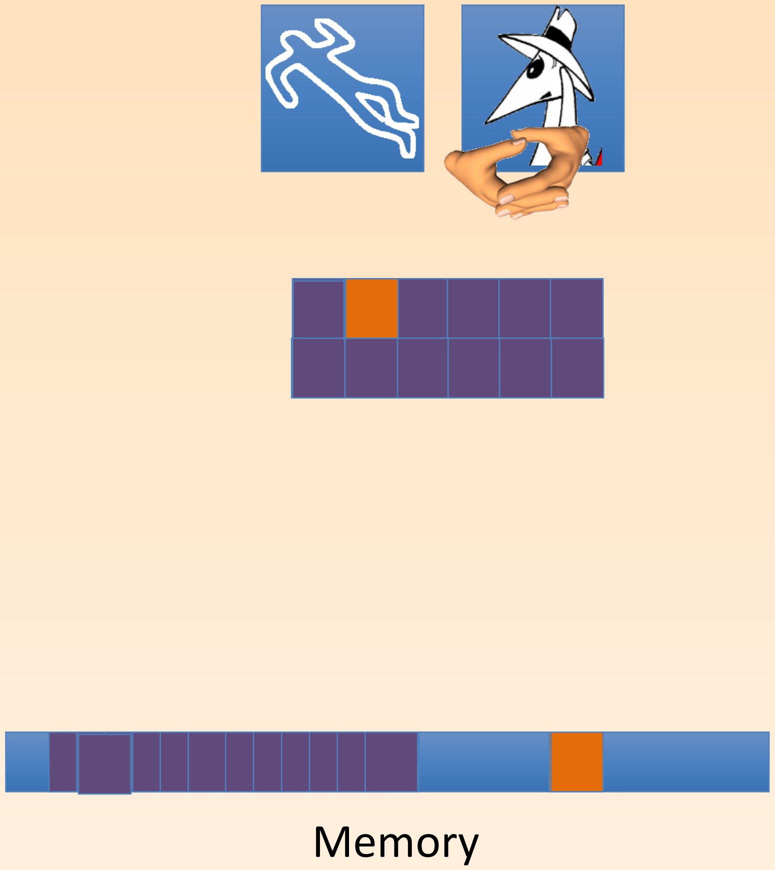
Calculating the exponent

$r \leftarrow 1$
for $i = \lceil n/w \rceil$-1, …, 0 **do**
  for j = 1, ..., w **do**
    $r \leftarrow r \cdot r \bmod m$
  $r \leftarrow r \cdot b_{d_i} \bmod m$
return r

1 0	0 1 1 0	1 0 0 1	0 1 0 0
d_3=2	d_2=6	d_1=9	d_0=4

Prime+Probe against RSA (Percival 2005)





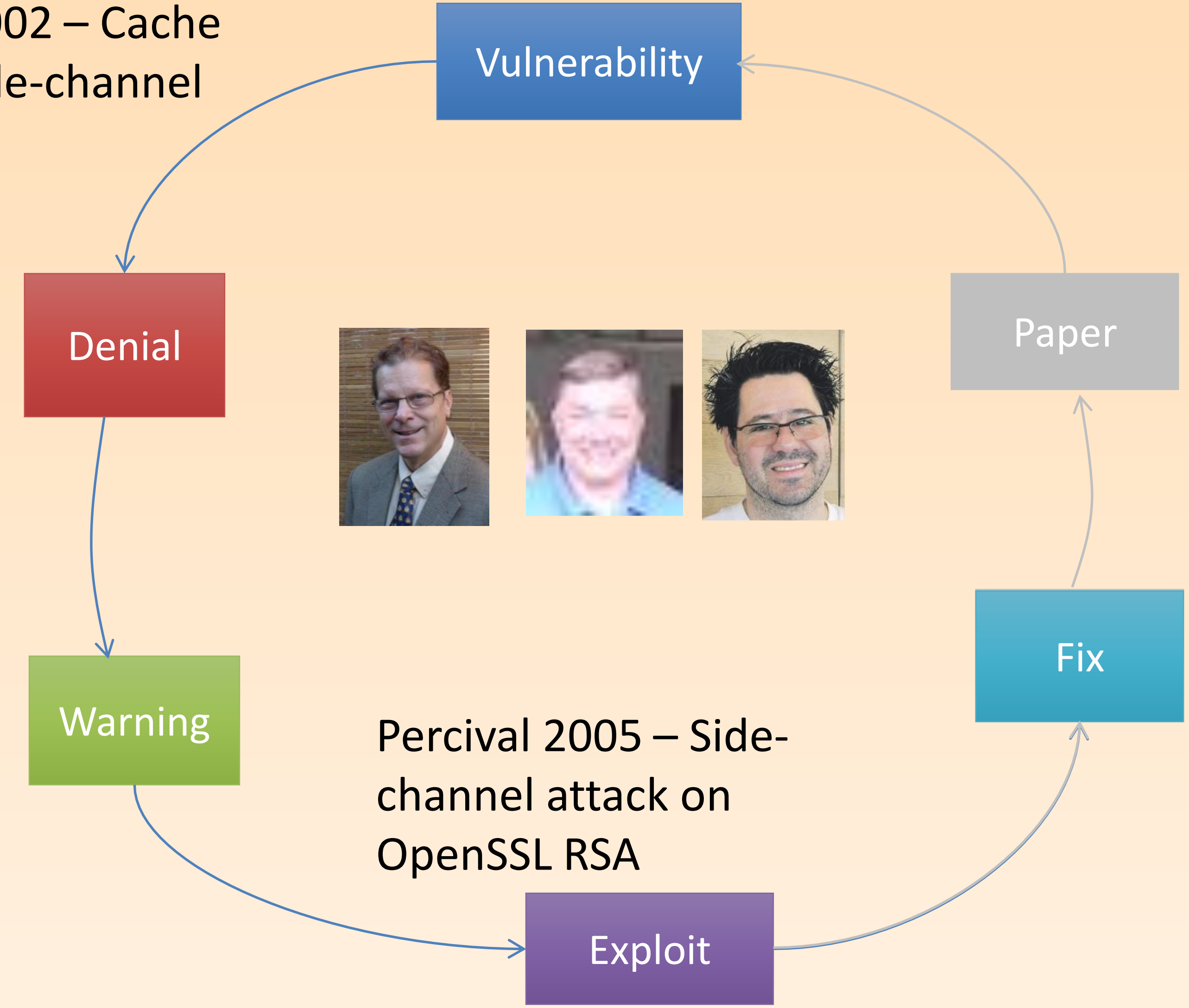
Why we can identify multipliers

- Each multiplier occupies consecutive cache lines
- Accessed throughout the multiplication

offset	0	1	2		63
Line 0	$M_0[0]$	$M_0[1]$	$M_0[2]$	•••	$M_0[63]$
Line 1	$M_0[64]$	$M_0[65]$	$M_0[66]$	•••	$M_0[127]$
Line 2	$M_0[128]$	$M_0[129]$	$M_0[130]$	•••	$M_0[191]$
Line 3	$M_1[0]$	$M_1[1]$	$M_1[2]$	•••	$M_1[63]$
Line 4	$M_1[64]$	$M_1[65]$	$M_1[66]$	•••	$M_1[127]$
Line 5	$M_1[128]$	$M_1[129]$	$M_1[130]$	•••	$M_1[191]$
Line 6	$M_2[0]$	$M_2[1]$	$M_2[2]$	•••	$M_2[63]$
Line 7	$M_2[64]$	$M_2[65]$	$M_2[66]$	•••	$M_2[127]$
Line 8	$M_2[128]$	$M_2[129]$	$M_2[130]$	•••	$M_2[191]$
	⋮	⋮	⋮		⋮
Line 191	$M_{63}[128]$	$M_{63}[129]$	$M_{63}[130]$	•••	$M_{63}[191]$

Round 1

Tsunoo, Tsujihara, Minematsu and Miyauchi 2002 – Cache may leak side-channel information



Percival 2005 – Side-channel attack on OpenSSL RSA

Brickell, Graunke & Seifert 2006 – Use Scatter Gather

Scatter-Gather

- Mitigate Prime+Probe
  - Sequence of accesses to cache lines does not depend on secret data

offset	0	1	2		63
Line 0	$M_0[0]$	$M_0[1]$	$M_0[2]$	• • •	$M_0[63]$
Line 1	$M_0[64]$	$M_0[65]$	$M_0[66]$	• • •	$M_0[127]$
Line 2	$M_0[128]$	$M_0[129]$	$M_0[130]$	• • •	$M_0[191]$
Line 3	$M_1[0]$	$M_1[1]$	$M_1[2]$	• • •	$M_1[63]$
Line 4	$M_1[64]$	$M_1[65]$	$M_1[66]$	• • •	$M_1[127]$
Line 5	$M_1[128]$	$M_1[129]$	$M_1[130]$	• • •	$M_1[191]$
Line 6	$M_2[0]$	$M_2[1]$	$M_2[2]$	• • •	$M_2[63]$
Line 7	$M_2[64]$	$M_2[65]$	$M_2[66]$	• • •	$M_2[127]$
Line 8	$M_2[128]$	$M_2[129]$	$M_2[130]$	• • •	$M_2[191]$
	⋮	⋮	⋮		⋮
Line 191	$M_{63}[128]$	$M_{63}[129]$	$M_{63}[130]$	• • •	$M_{63}[191]$

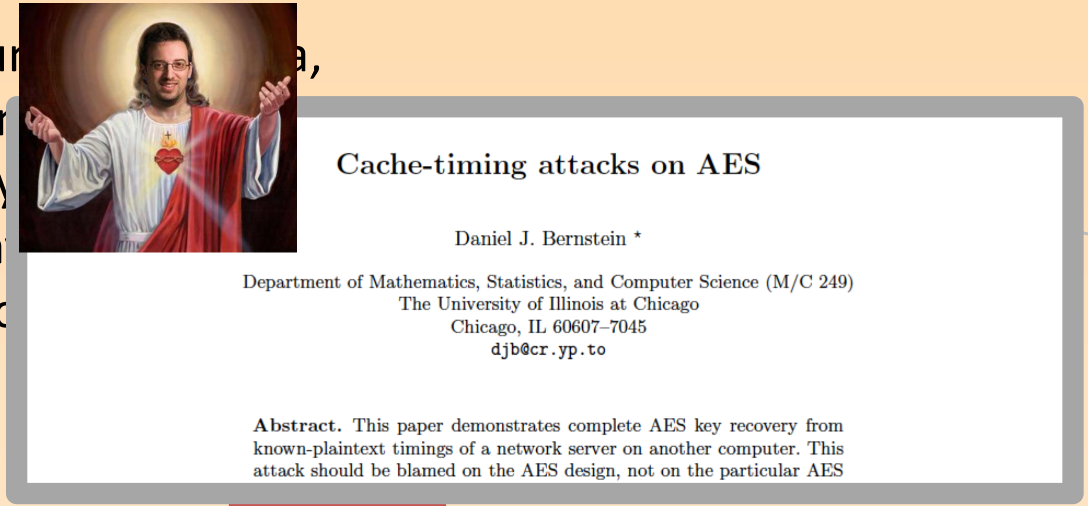
offset	0	1	2		63
Line 0	$M_0[0]$	$M_1[0]$	$M_2[0]$	• • •	$M_{63}[0]$
Line 1	$M_0[1]$	$M_1[1]$	$M_2[1]$	• • •	$M_{63}[1]$
Line 2	$M_0[2]$	$M_1[2]$	$M_2[2]$	• • •	$M_{63}[2]$
Line 3	$M_0[3]$	$M_1[3]$	$M_2[3]$	• • •	$M_{63}[3]$
Line 4	$M_0[4]$	$M_1[4]$	$M_2[4]$	• • •	$M_{63}[4]$
Line 5	$M_0[5]$	$M_1[5]$	$M_2[5]$	• • •	$M_{63}[5]$
Line 6	$M_0[6]$	$M_1[6]$	$M_2[6]$	• • •	$M_{63}[6]$
Line 7	$M_0[7]$	$M_1[7]$	$M_2[7]$	• • •	$M_{63}[7]$
Line 8	$M_0[8]$	$M_1[8]$	$M_2[8]$	• • •	$M_{63}[8]$
	⋮	⋮	⋮		⋮
Line 191	$M_0[191]$	$M_1[191]$	$M_2[191]$	• • •	$M_{63}[191]$

OpenSSL’s Layout

offset	0 Bin 0 7	8 Bin 1 15		56 Bin 7 63
Line 0	M_0[0-7]	M_1[0-7]	• • •	M_7[0-7]
Line 1	M_8[0-7]	M_9[0-7]	• • •	M_{15}[0-7]
Line 2	M_{16}[0-7]	M_{17}[0-7]	• • •	M_{23}[0-7]
Line 3	M_{24}[0-7]	M_{25}[0-7]	• • •	M_{31}[0-7]
Line 4	M_0[8-15]	M_1[8-15]	• • •	M_7[8-15]
Line 5	M_8[8-15]	M_9[8-15]	• • •	M_{15}[8-15]
Line 6	M_{16}[8-15]	M_{17}[8-15]	• • •	M_{23}[8-15]
Line 7	M_{24}[8-15]	M_{25}[8-15]	• • •	M_{31}[8-15]
Line 8	M_0[16-23]	M_1[16-23]	• • •	M_7[16-23]
	⋮	⋮		⋮
Line 95	M_{24}[184-191]	M_{25}[184-191]	• • •	M_{31}[184-191]

Round 1

Tsur a,
Mi
Mi
ma
inf



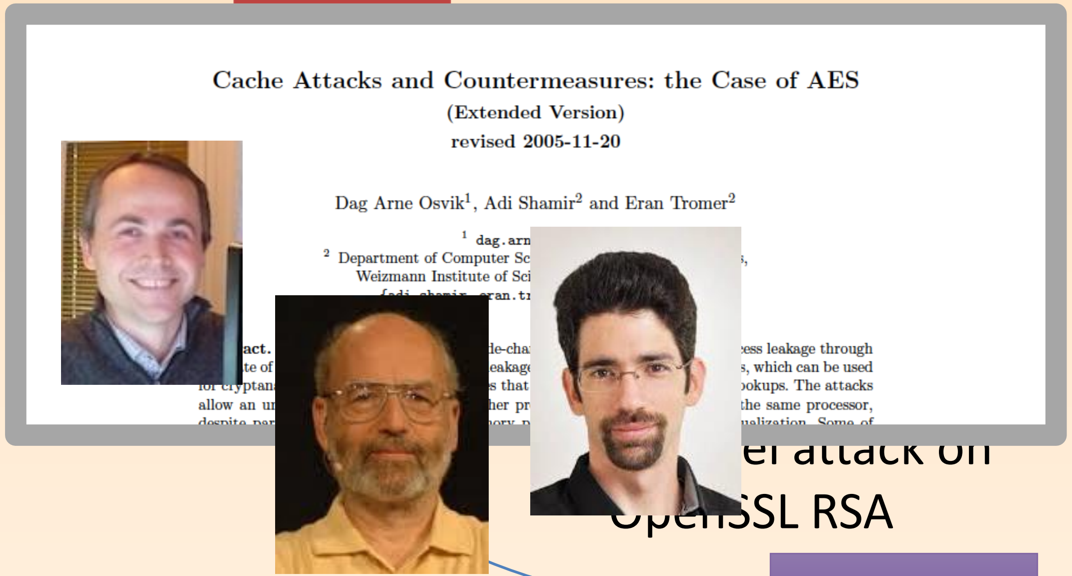
Cache-timing attacks on AES

Daniel J. Bernstein *

Department of Mathematics, Statistics, and Computer Science (M/C 249)
The University of Illinois at Chicago
Chicago, IL 60607–7045
djb@cr.yp.to

Abstract. This paper demonstrates complete AES key recovery from known-plaintext timings of a network server on another computer. This attack should be blamed on the AES design, not on the particular AES

Denial



Cache Attacks and Countermeasures: the Case of AES
(Extended Version)
revised 2005-11-20

Dag Arne Osvik[1], Adi Shamir[2] and Eran Tromer[2]

[1] dag.arn
[2] Department of Computer Sc
Weizmann Institute of Sci

er attack on
SSL RSA

Bernstein 2005, Osvik Shamir & Tromer 2006 – Scatter Gather may leak information

Paper

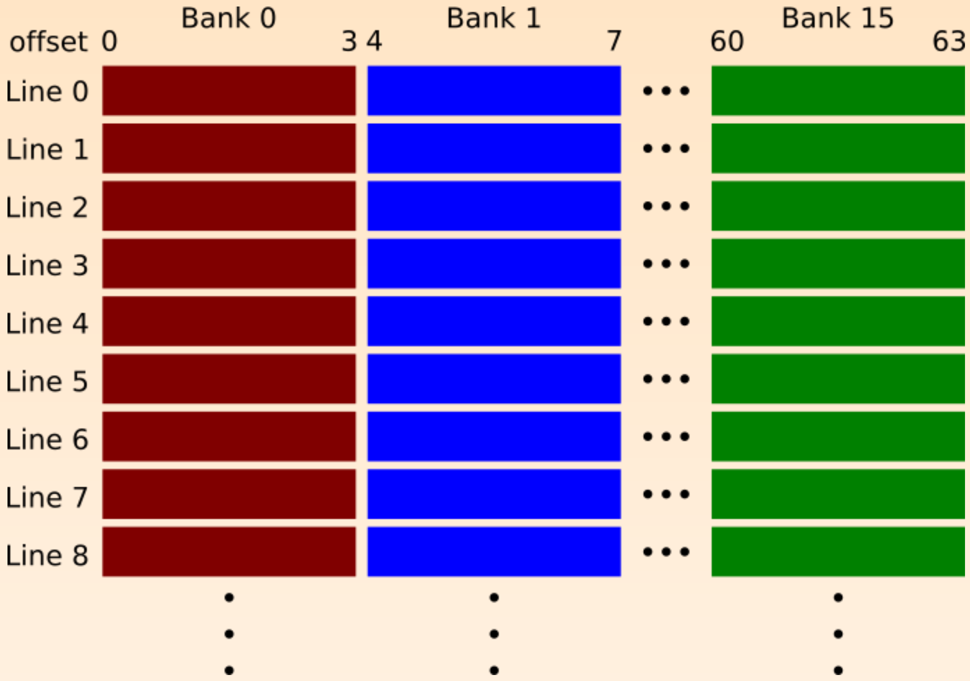
Brickell, Graunke & Seifert 2006 – Use Scatter Gather

Fix

Exploit

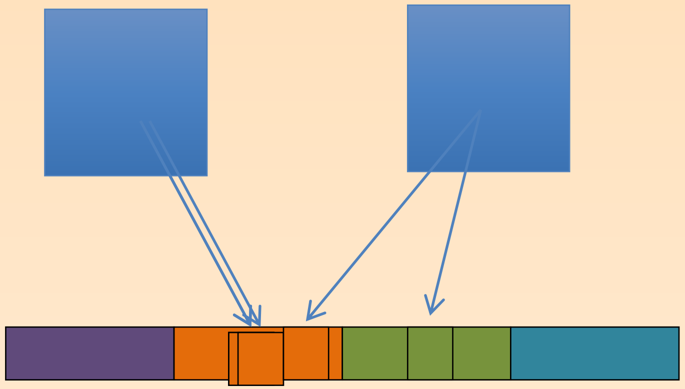
Cache banks

- To support superscalar processing the cache is divided into cache banks
  - Bits 2-5 of the address determine the bank



Cache Banks

- In Sandy Bridge, each bank can serve only one request per cycle.
  - Concurrent access to different banks is always possible
  - Concurrent access to the same bank causes delays



Round 2

Bernstein 2005, Osvik Shamir & Tromer 2006 – Scatter Gather may leak information

Vulnerability

Brickell 2011 – OpenSSL mitigates side channels

Denial

Paper

Schwabe 2015 - “TODO: Real attack against e.g. OpenSSL”

Bernstein & Schwabe 2013 – There is a side-channel

Warning

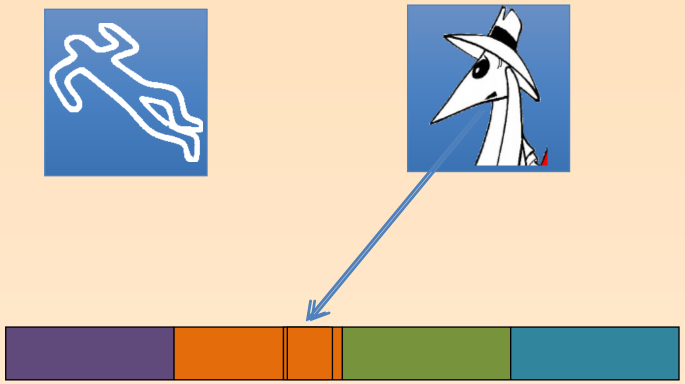
Fix

Yarom, Genkin & Heninger 2016 – CacheBleed

Exploit

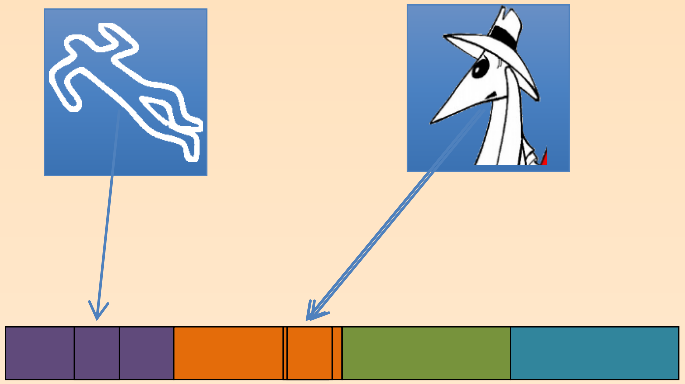
CacheBleed Operation

- Spy generate a long sequence of accesses to the same cache bank



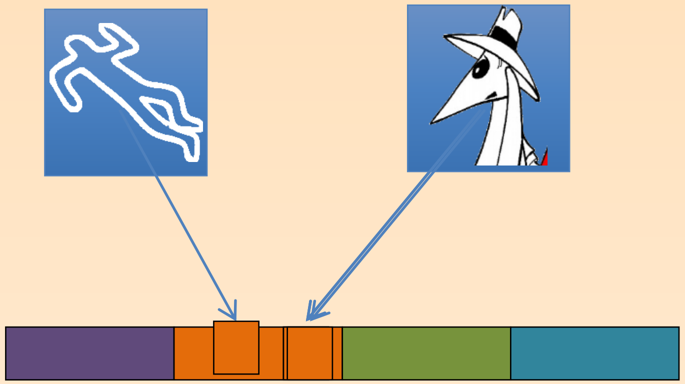
CacheBleed Operation

- Spy generate a long sequence of accesses to the same cache bank
- Victim accesses to a different cache bank do not affect speed



CacheBleed Operation

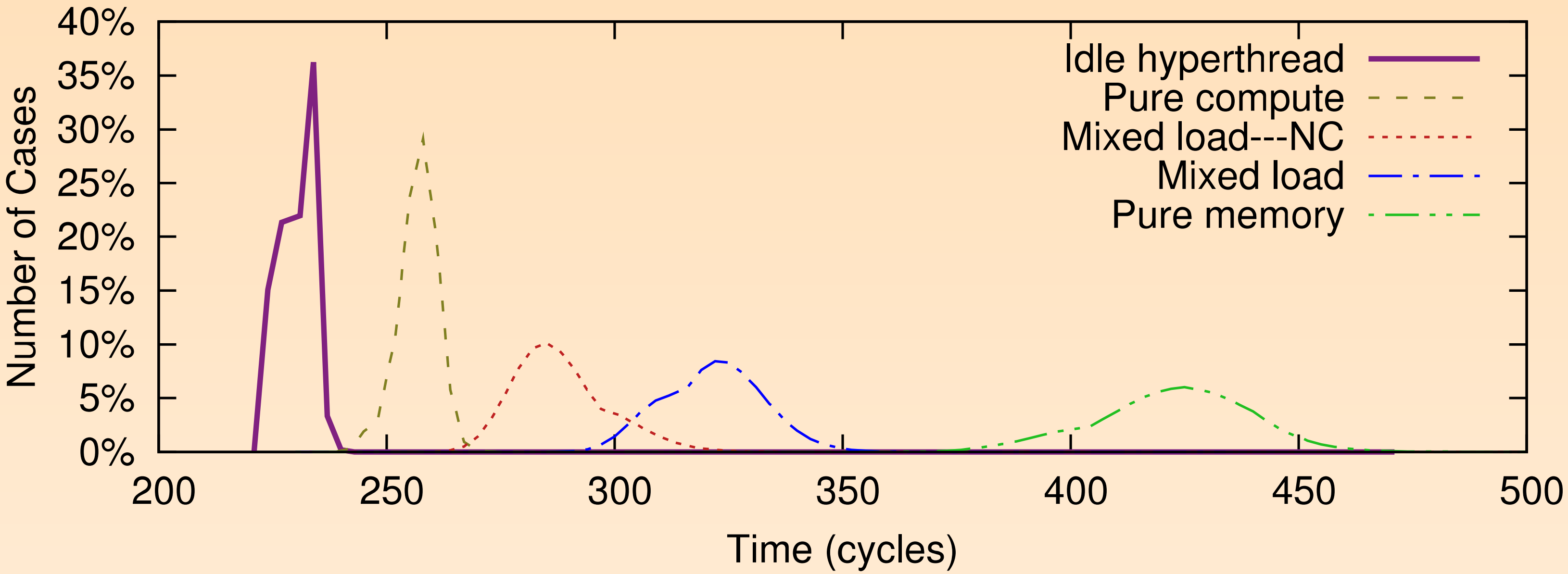
- Spy generate a long sequence of accesses to the same cache bank
- Victim accesses to a different cache bank do not affect speed
- Victim accesses to same cache bank cause delays



Implementation

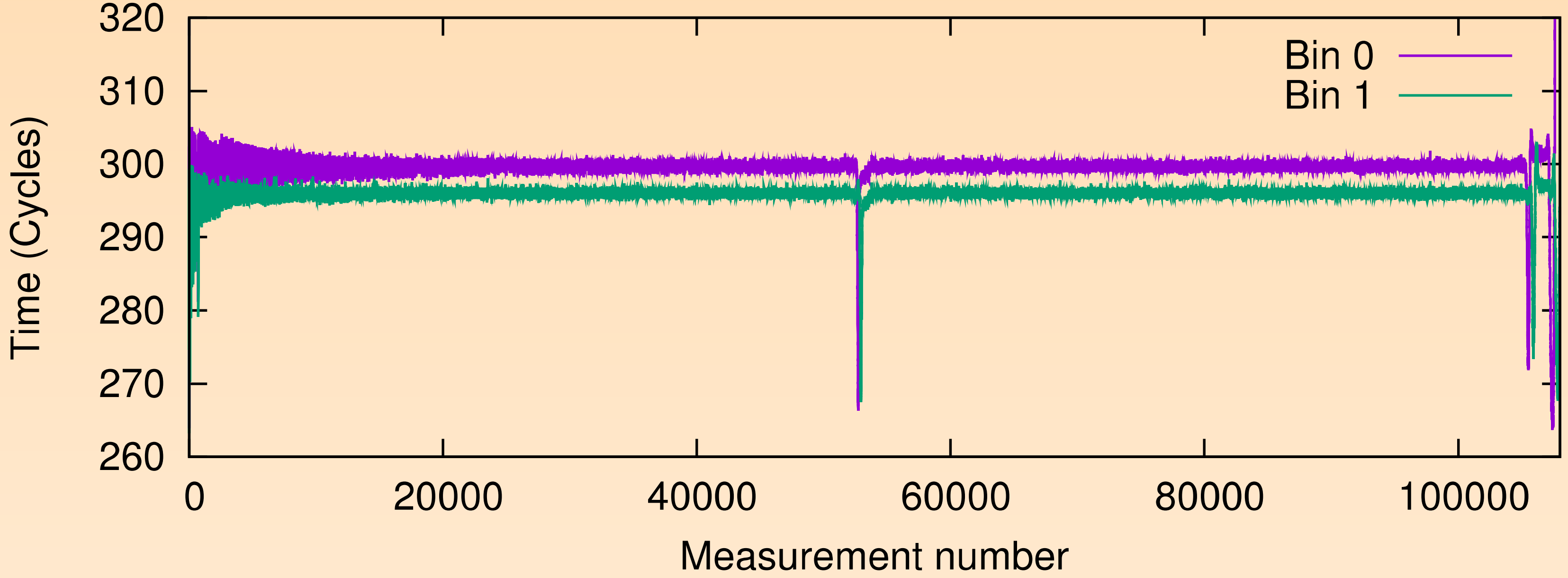
```
rdtscp
movq    %rax, %r10

        addl    0x000(%r9), %eax
        addl    0x040(%r9), %ecx
        addl    0x080(%r9), %edx
        addl    0x0c0(%r9), %edi
        addl    0x100(%r9), %eax
        addl    0x140(%r9), %ecx
        addl    0x180(%r9), %edx
        addl    0x1c0(%r9), %edi
.
.
.
        addl    0xf00(%r9), %eax
        addl    0xf40(%r9), %ecx
        addl    0xf80(%r9), %edx
        addl    0xfc0(%r9), %edi

        rdtscp
        subq    %r10, %rax
```

CacheBleed timing



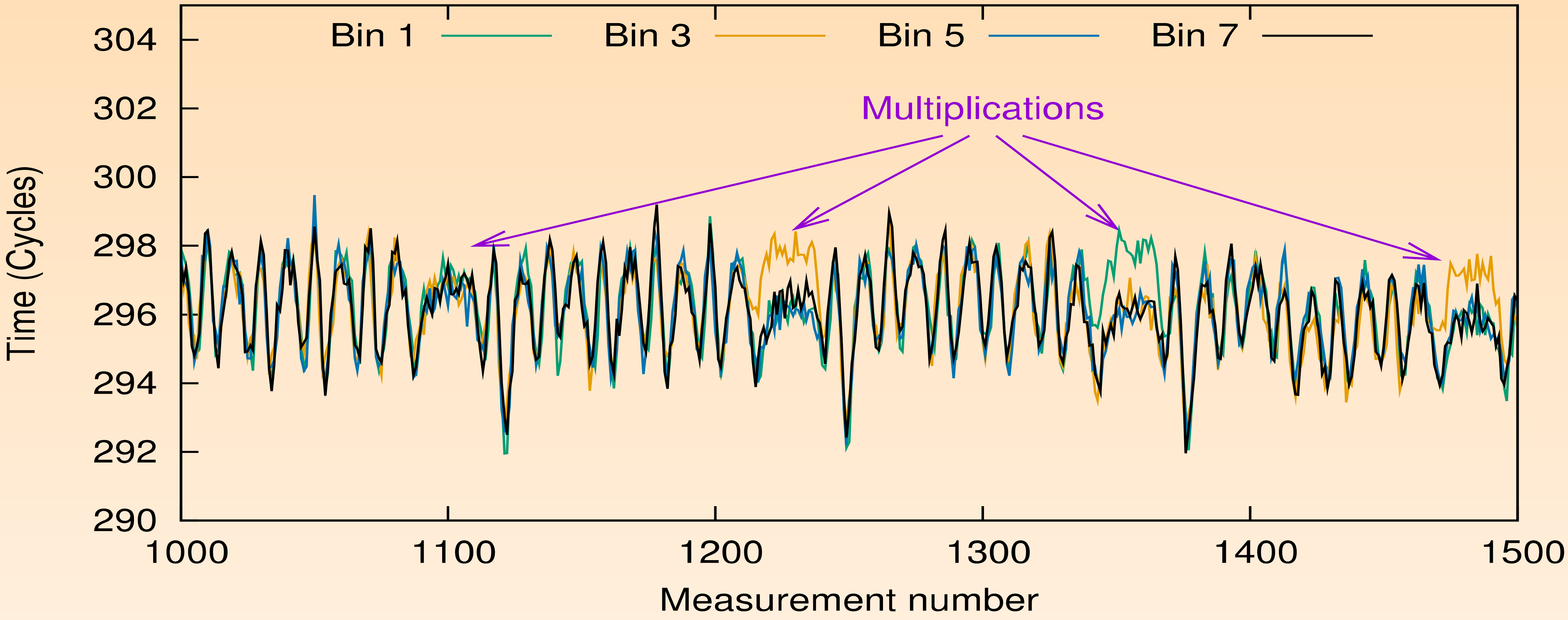
- Need multiple samples to determine cache-bank conflicts

CacheBleed on OpenSSL

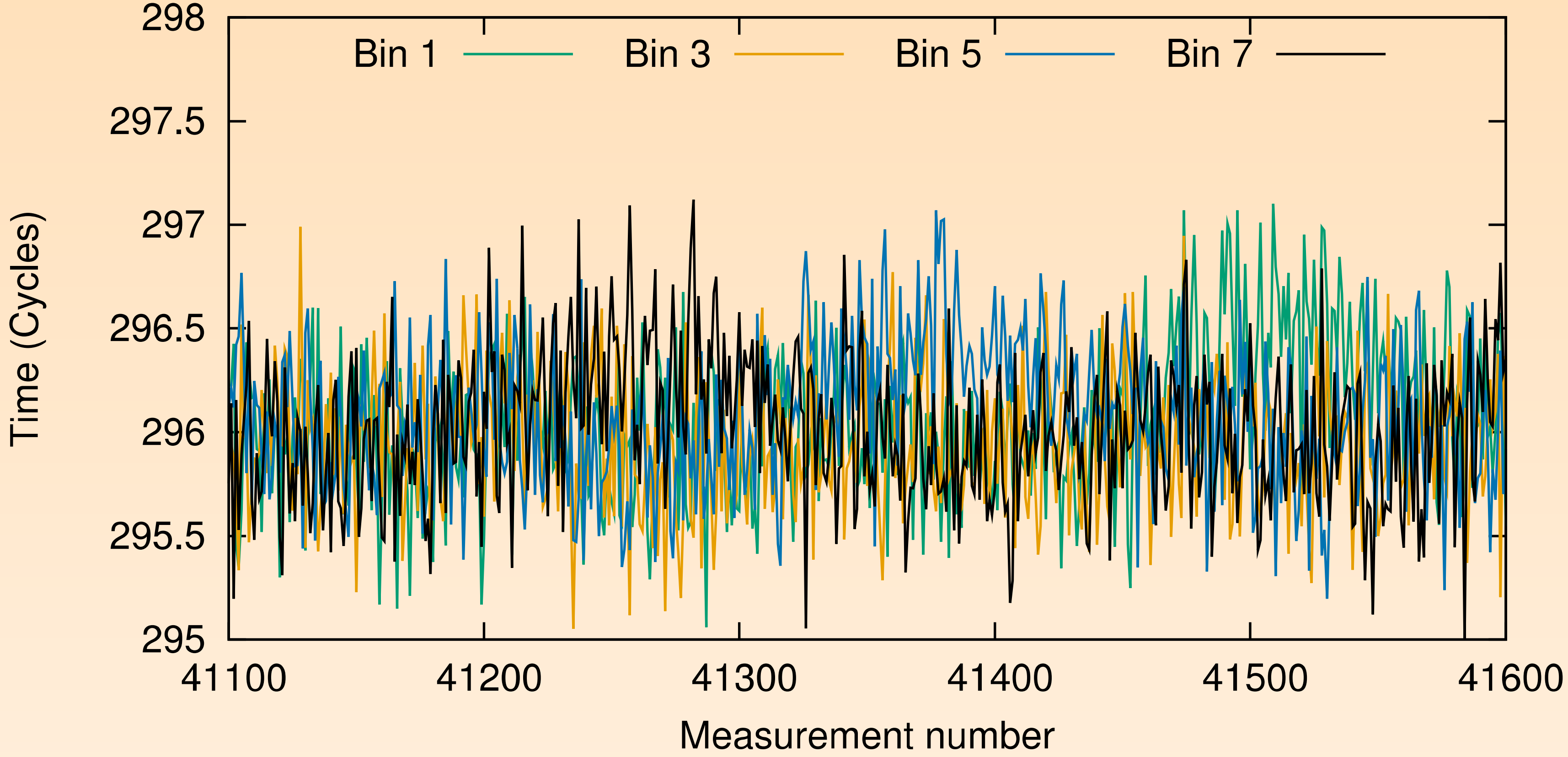


- Average of 1,000 sequences on each bin
- Odd and even bins have different timing characteristics

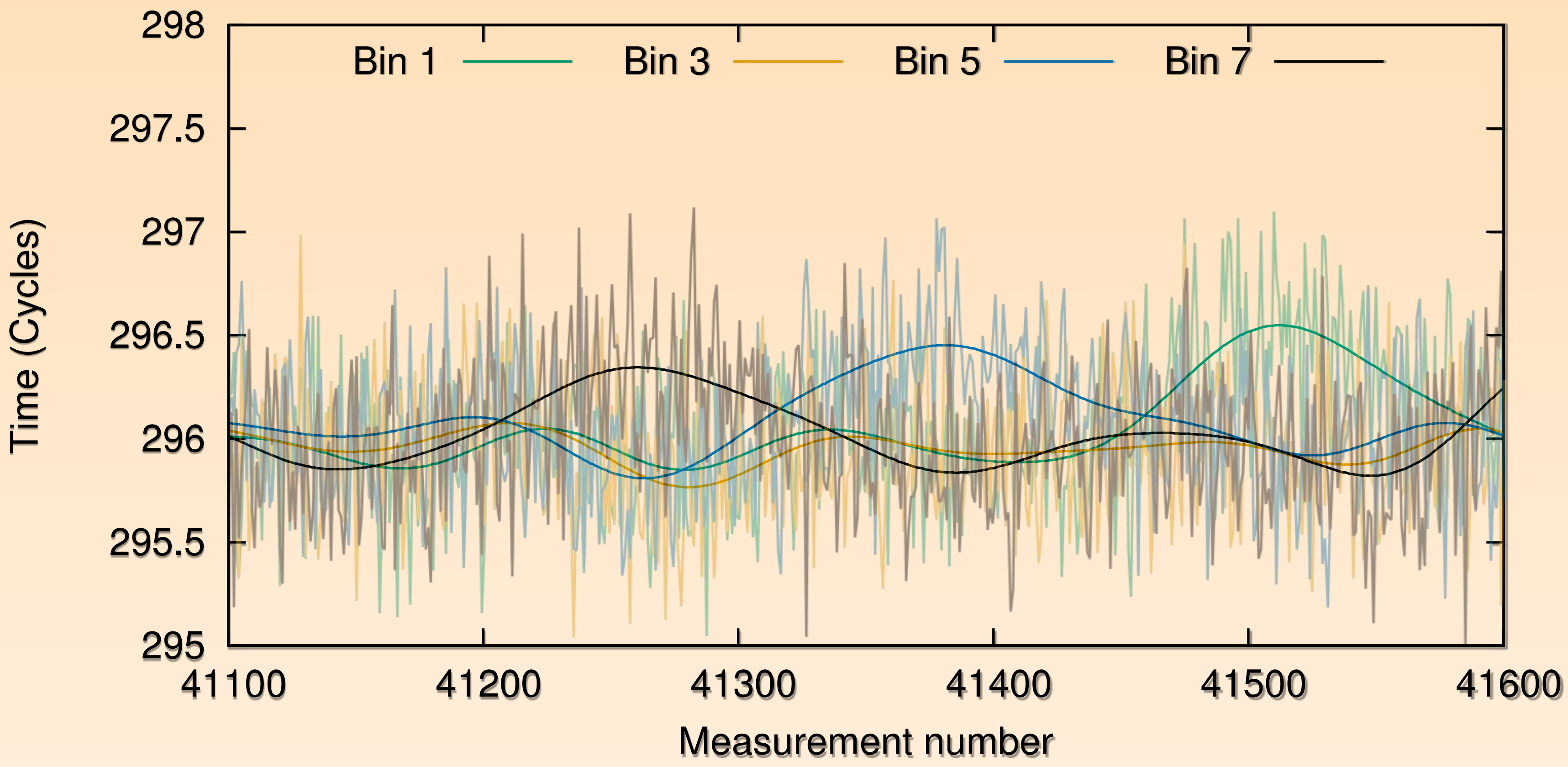
CacheBleed on OpenSSL - Details



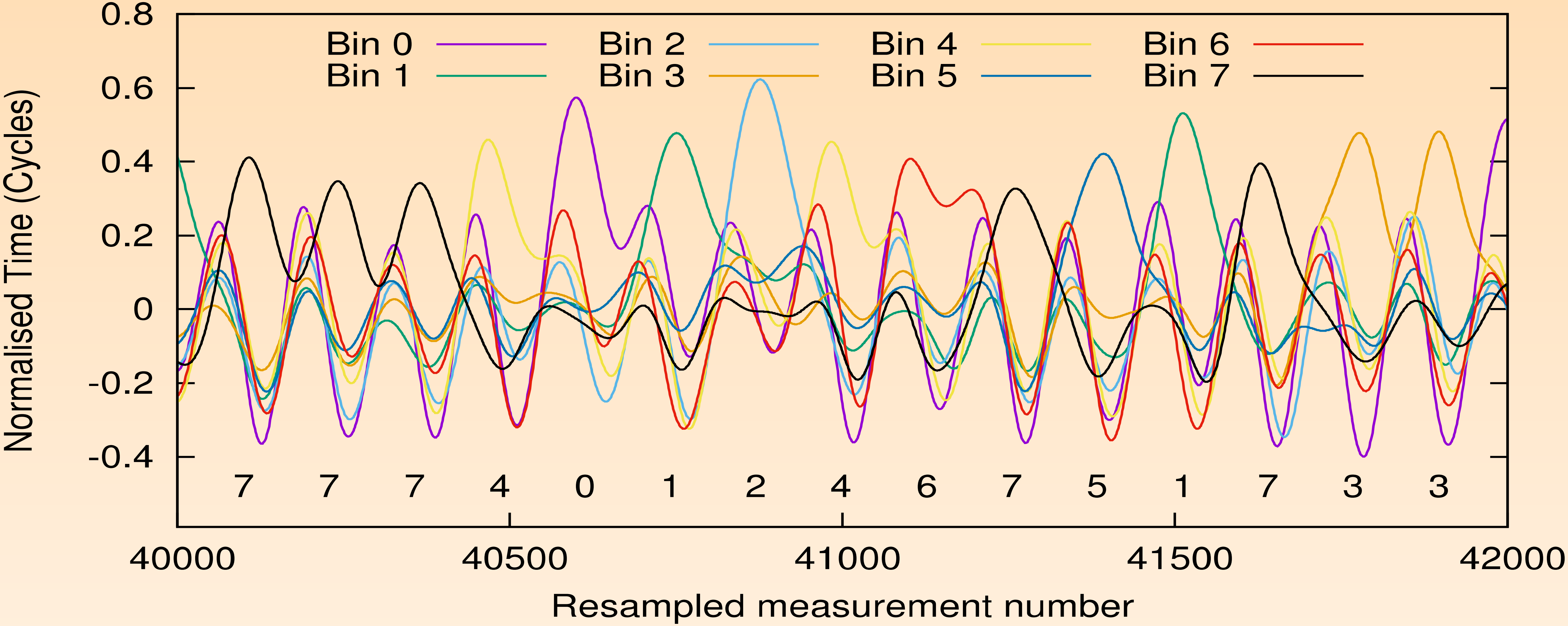
Clock Drift


Bin 1
Bin 3
Bin 5
Bin 7
298
297.5
297
296.5
296
295.5
295
Time (Cycles)
41100
41200
41300
41400
41500
41600
Measurement number

Low-pass filter



Normalised + resampled



Results

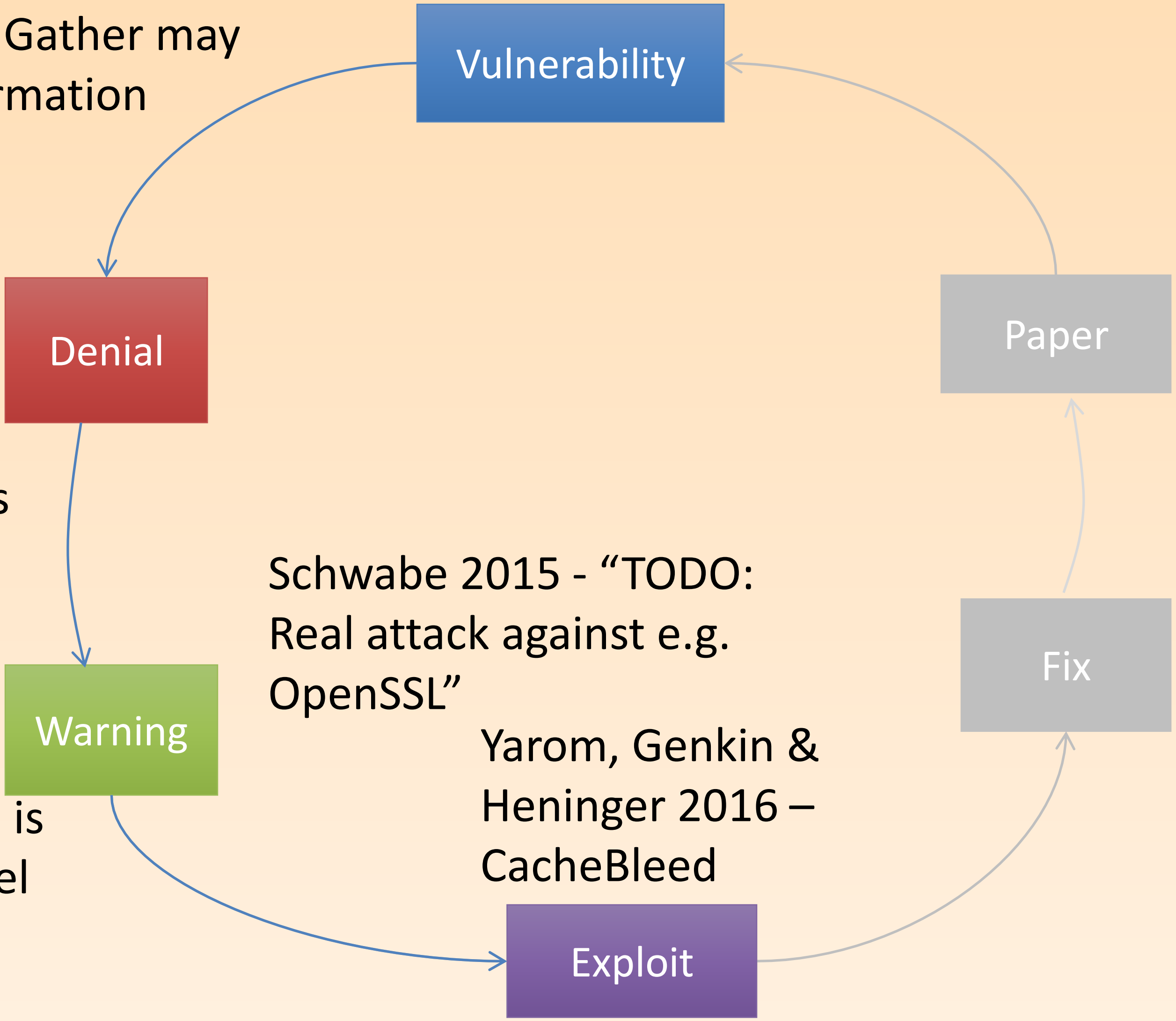
- 16,000 decryptions (1,000 sequences per bin per exponentiation)
  - Less than 5 minutes online attack
- Recover three bits of each multiplier
  - Miss the first and last one or two multipliers

Recovering missing bits

- We know 60% of the bits (3 bits in each 5) in both d_p and d_q
  - Heninger-Shacham requires 50% of the bits
  - Complete key recovery requires two CPU hours – less than 3 minutes on a high-end server

Round 2

Bernstein 2005, Osvik Shamir & Tromer 2006 – Scatter Gather may leak information



Brickell 2011 – OpenSSL mitigates side channels

Bernstein & Schwabe 2013 – There is a side-channel

Schwabe 2015 - “TODO: Real attack against e.g. OpenSSL”

Yarom, Genkin & Heninger 2016 – CacheBleed

Round 2

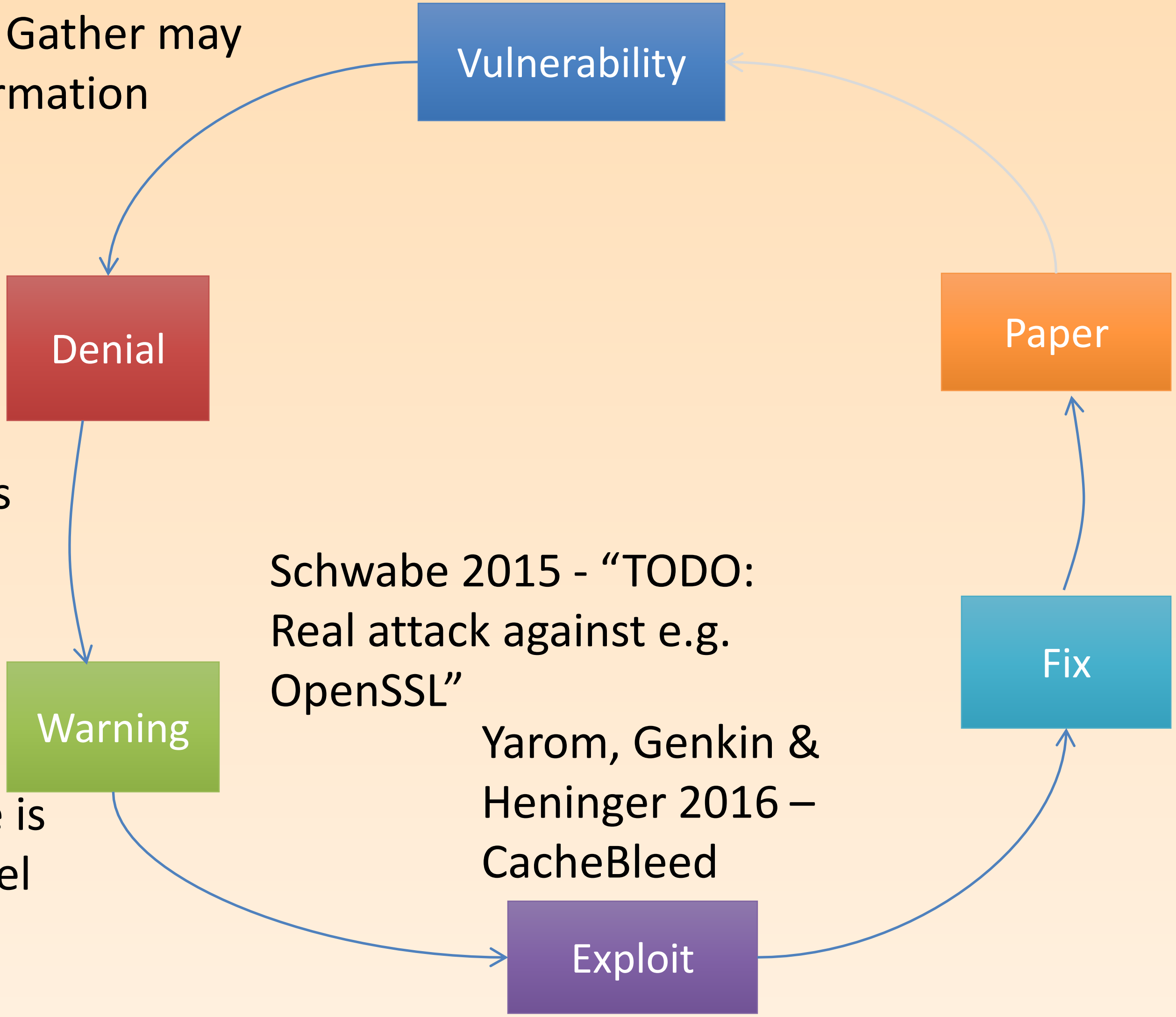
Bernstein 2005, Osvik Shamir & Tromer 2006 – Scatter Gather may leak information

Brickell 2011 – OpenSSL mitigates side channels

Bernstein & Schwabe 2013 – There is a side-channel

Schwabe 2015 - "TODO: Real attack against e.g. OpenSSL"

Yarom, Genkin & Heninger 2016 – CacheBleed



OpenSSL Proposed “Fix”

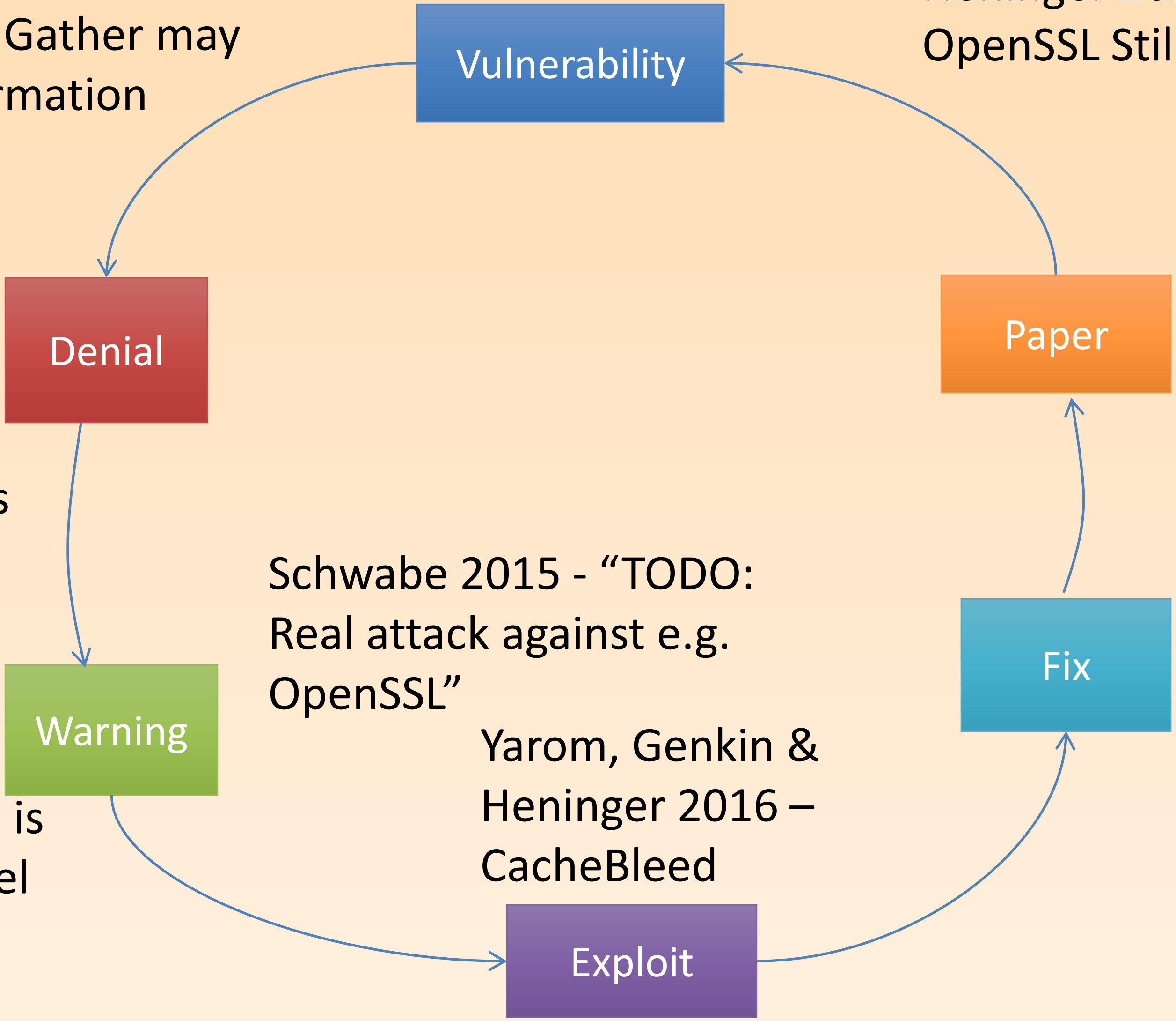
- Use 128-bit reads with masking
  - Only leaks 2 bits per multiplier – not enough for Heninger-Shacham
- Read at a different offset in each of the four cache lines
  - Order depends on the multiplier
  - Too fast for our attack

Round 2

Bernstein 2005, Osvik Shamir & Tromer 2006 – Scatter Gather may leak information

Yarom, Genkin & Heninger 2016 – OpenSSL Still leaks

Brickell 2011 – OpenSSL mitigates side channels



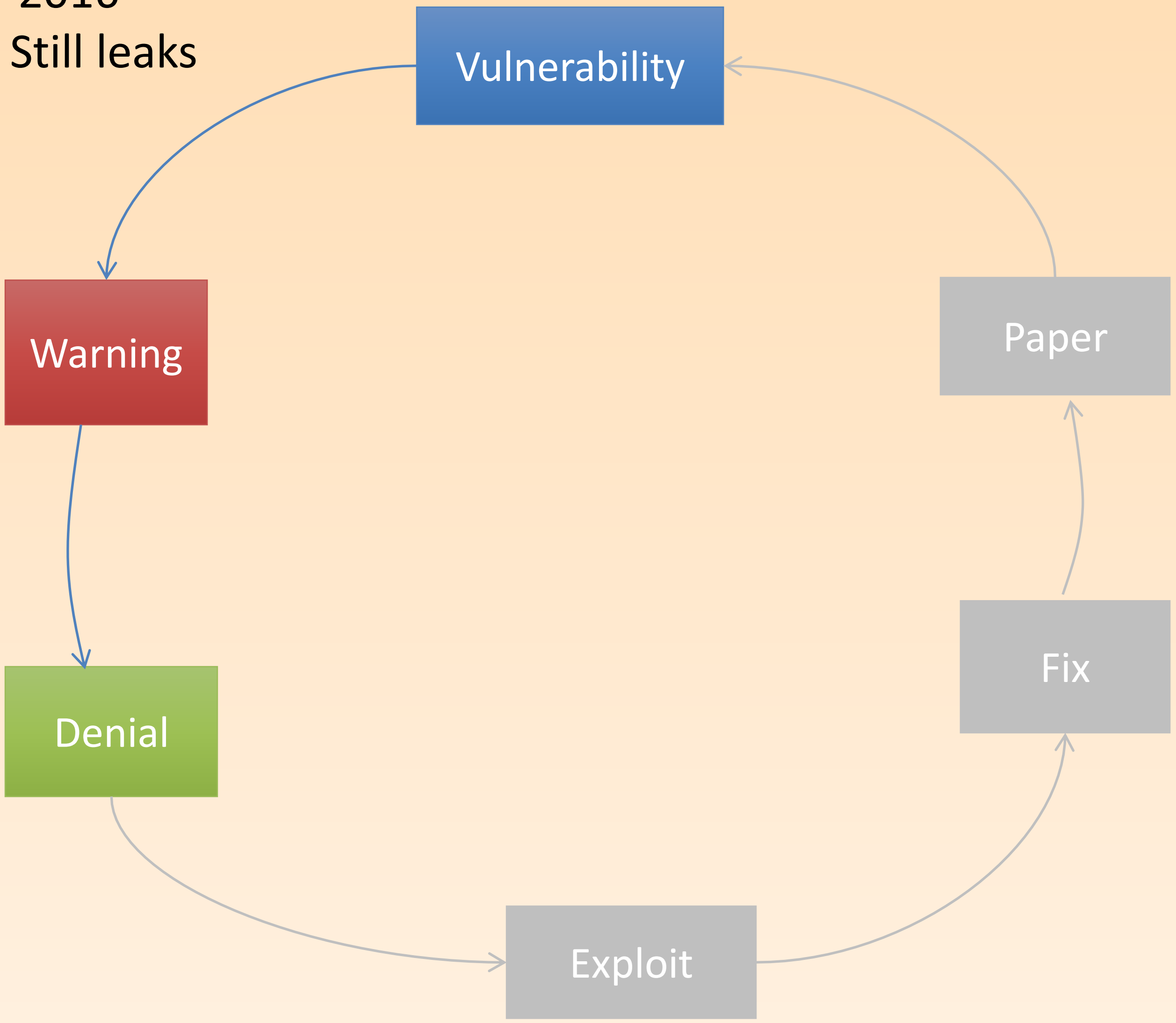
Schwabe 2015 - “TODO: Real attack against e.g. OpenSSL”

Bernstein & Schwabe 2013 – There is a side-channel

Yarom, Genkin & Heninger 2016 – CacheBleed

Round 3

Yarom, Genkin & Heninger 2016 – OpenSSL Still leaks



CacheBleed

- Fixed in the SkyLake microarchitecture
  - Multiple cache ports
- MemJam – false dependencies
  - Moghimi et al. “MemJam: A False Dependency Attack against Constant-Time Crypto Implementations”, CT-RSA 2018
- Port contention
  - Aldaya et al. “Port Contention for Fun and Profit”, IEEE SP 2019

Summary

- Microarchitectural attacks often return partial information
- Can use redundancy to reconstruct key
- Heninger-Shacham algorithm

- Next: lower-level caches and eviction sets
  - Read: Vila et al. “Theory and Practice of Finding Eviction Sets”, IEEE S&P 2019