Practical Cryptographic Systems

Side Channel Attacks

Instructor: Matthew Green

Housekeeping:

- A2 is out

News?

N

CPO MAGAZINE

HOME NEWS INSIGHTS RESOURCES

DATA PRIVACY NEWS · 4 MIN READ

WhatsApp Sues Indian Government Over New "Traceability" Rules That Require Circumvention of End-to-End Encryption

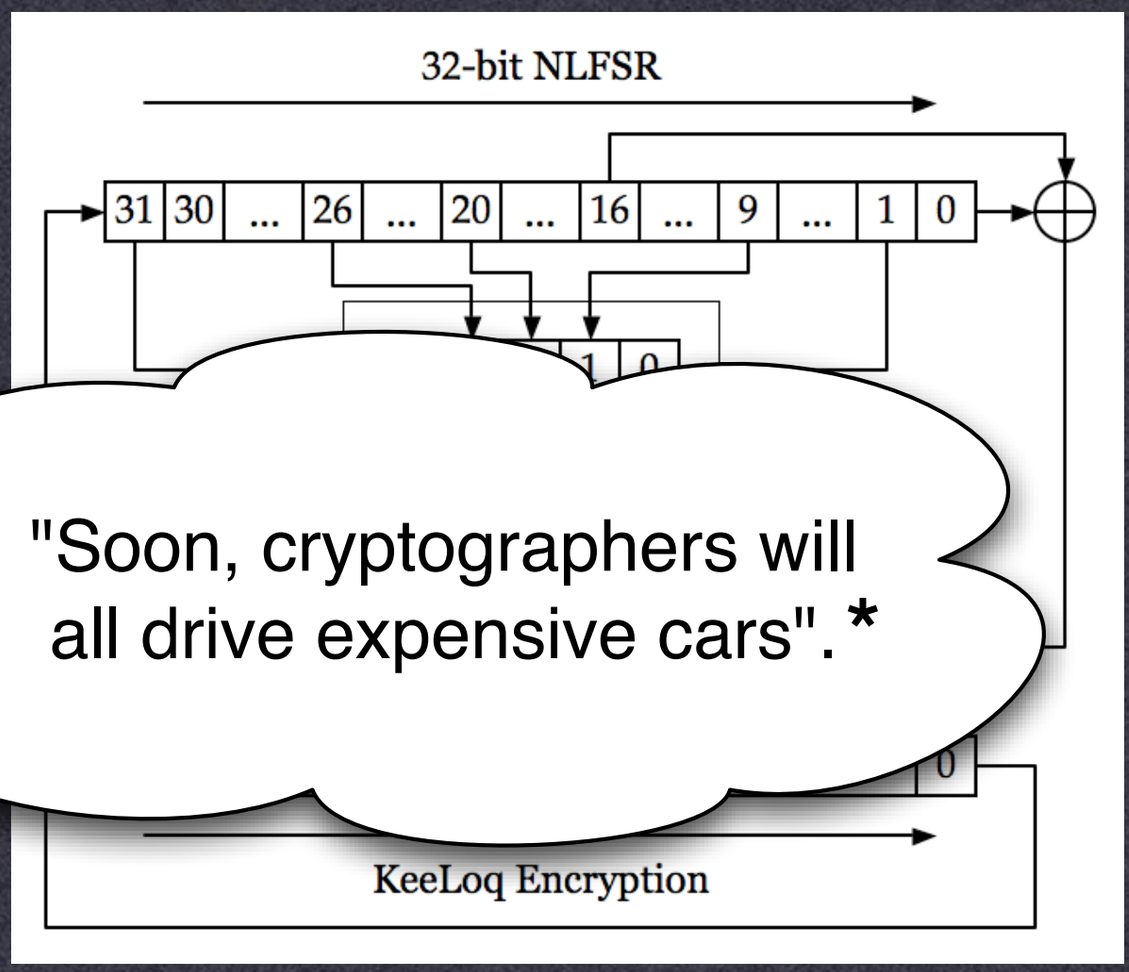
SCOTT IKEDA · JUNE 1, 2021

Recent rules passed in India that threaten end-to-end encryption are being challenged in court by WhatsApp, which stands to lose the central selling point of

KeeLoq

- Designed by Kuhn & Smit
  - Block cipher: 64-bit key, 32-bit block
  - 528 rounds
  - Mostly used for door opening
  - Direct attacks*:

A. 2^{16} data & 2^{51} operation
B. 2^{32} data & 2^{27} operations
C. Indesteege et al.:
(65 min to get data,
218 CPU days to process)



"Soon, cryptographers will all drive expensive cars".*

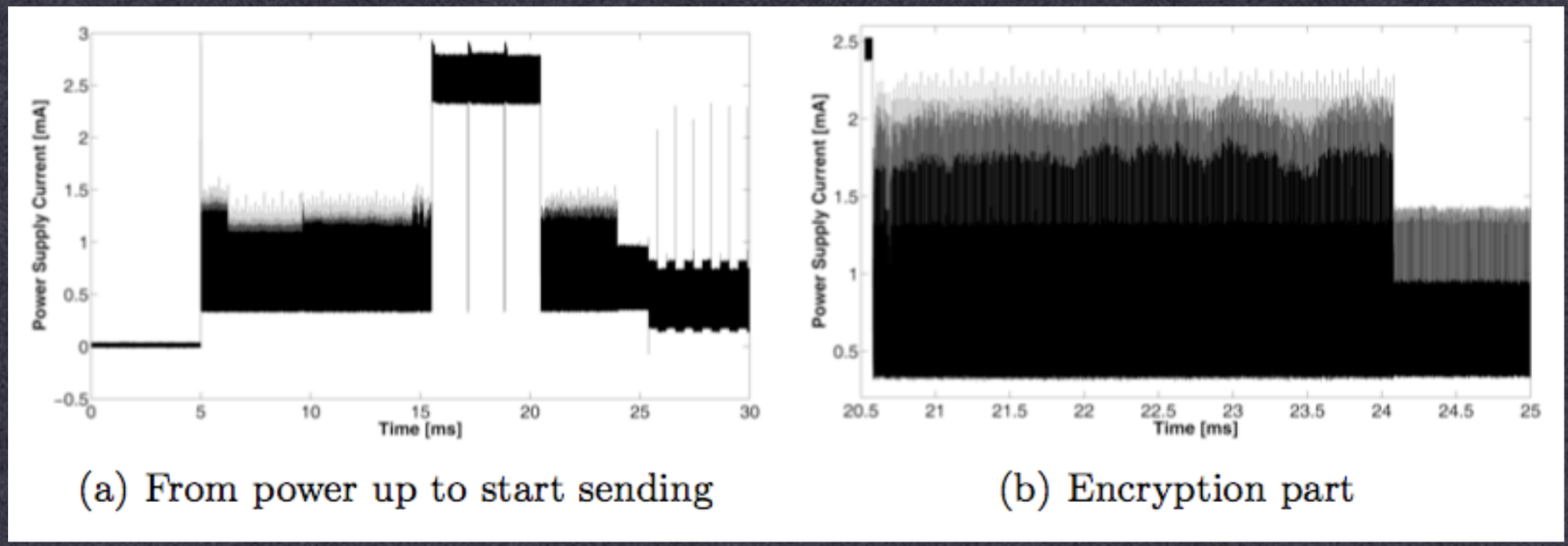
Public domain image from Wikipedia
* Source: Indesteege et al.: http://www.cosic.esat.kuleuven.be/keeloq/keeloq-slides.pdf

KeeLoq

- Attack CRYPTO 2008
  - Requires physical access to device
  - Recovers secret key after 10-60 reads
  - How?

KeyLoq

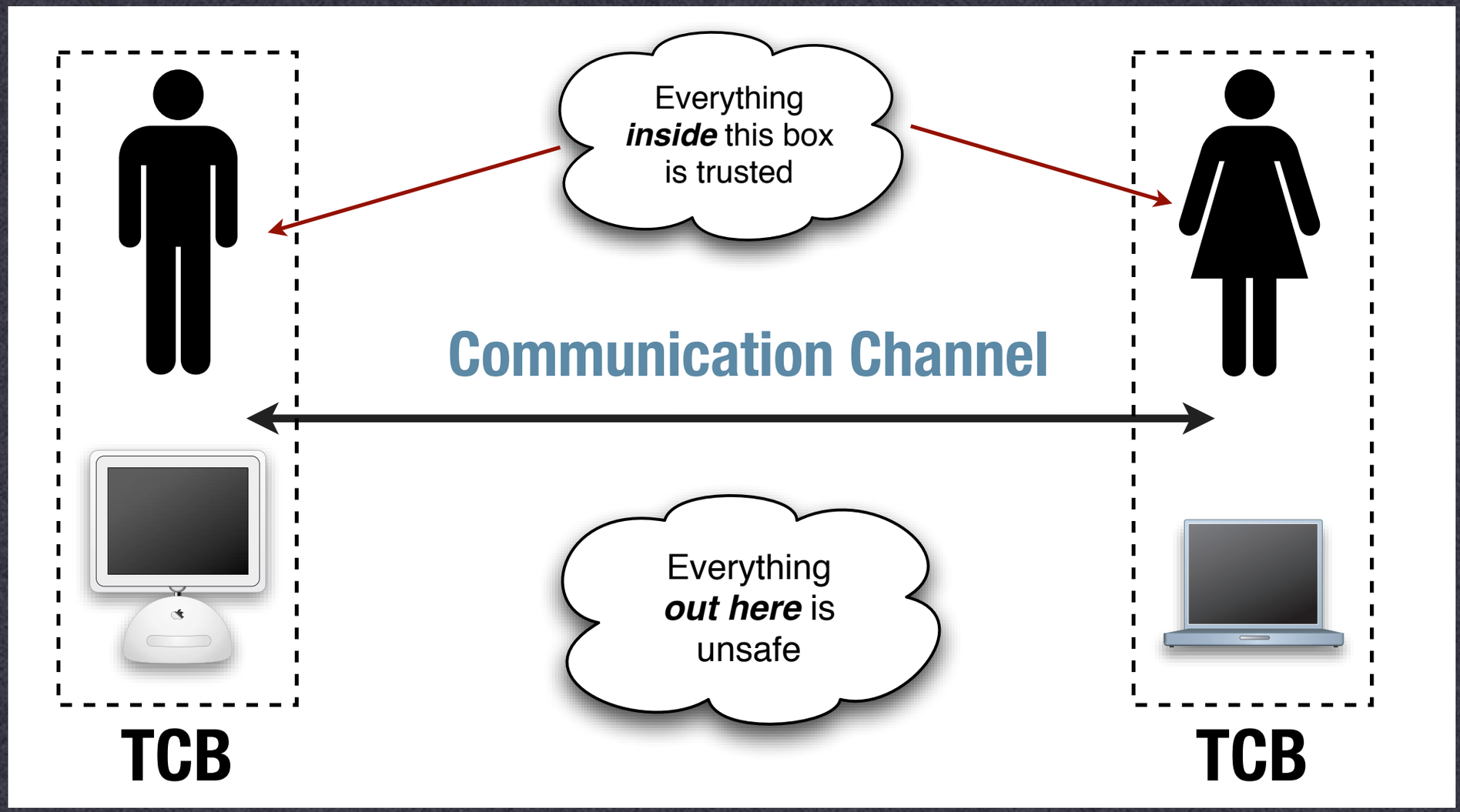
- Side channel attack
  - Doesn't directly attack the cipher
  - Instead, measures how the cipher works in operation.
  - In this case, use power consumption



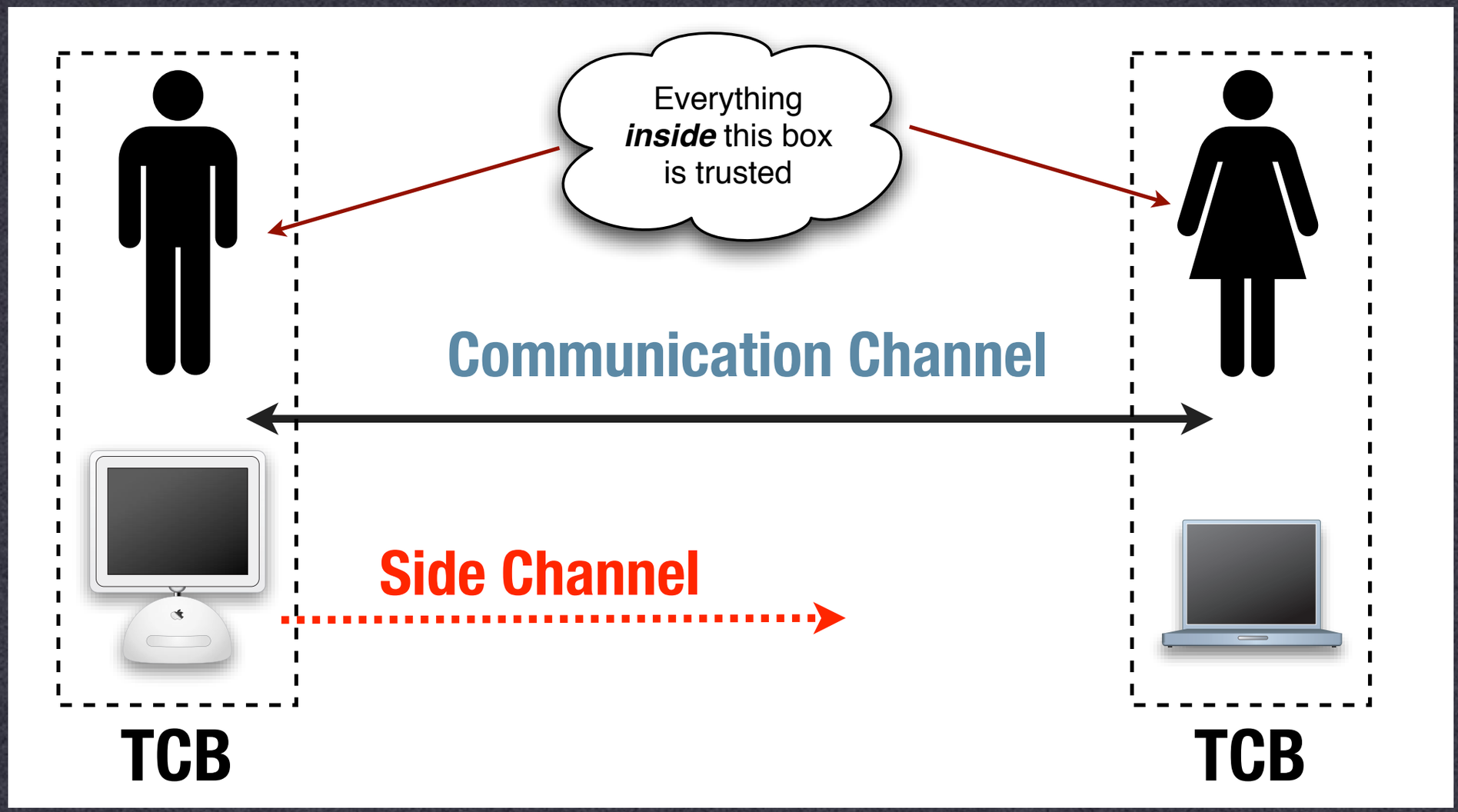
(a) From power up to start sending (b) Encryption part

Source: Eisenbarth et al., On the Power of Power Analysis in the Real World: A Complete Break of the KeeLoq Code Hopping Scheme

Review

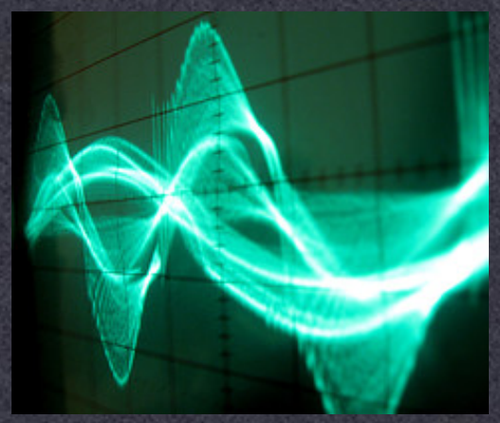

Everything
inside this box
is trusted
Communication Channel
Everything
out here is
unsafe
TCB
TCB

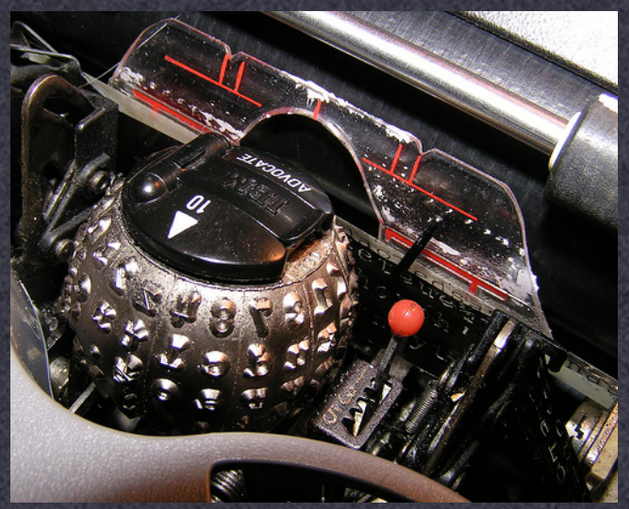
Side Channels



Side Channels

- Some history:
  - 1943: Bell engineer detects power spikes from encrypting teletype
  - 1960s: US monitors EM emissions from foreign cipher equipment
  - 1980s: USSR places eavesdropping device inside IBM Selectric typewriters



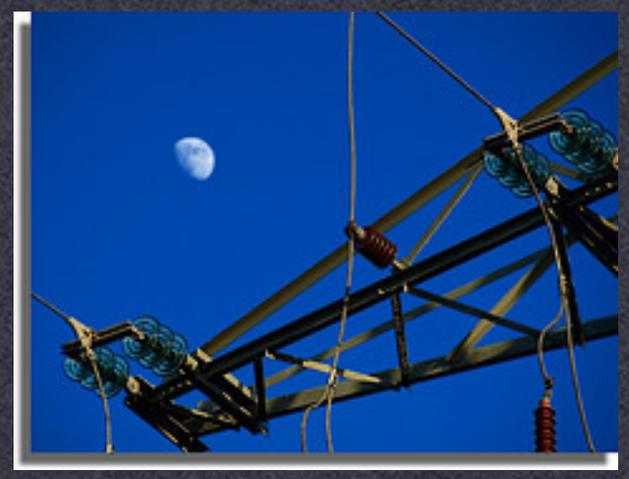


Side Channels

- Some history:
  - 1990s: Paul Kocher demonstrates timing attacks, power analysis attacks against RSA, Elgamal

Common Examples

- Timing
- Power Consumption
- RF Emissions
- Audio



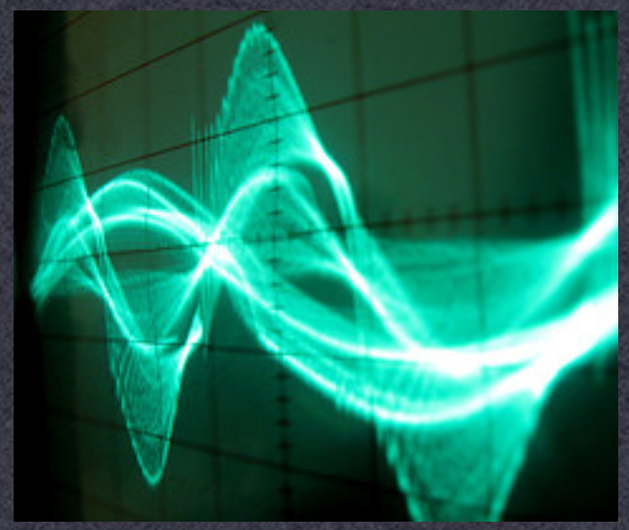


Photo Credits: Clock - Flickr user laffy4k, Waveform - Flickr user altemark, Power Line - Flickr user Kliefi. All used under a Creative Commons license.

MAC verification

- How does one verify a MAC?

RSA Cryptosystem

$N = p \cdot q$

$e \cdot d \equiv 1 \; mod \; (p-1)(q-1)$

Encryption

$$c = m^e \; mod \; N$$

Decryption

$$m = c^d \; mod \; N$$

RSA Cryptosystem

$$N = p \cdot q$$

$$e \cdot d \equiv 1 \; mod \; (p-1)(q-1)$$

Encryption

$$c = m^e \; mod \; N$$

Decryption

$$m = c^d \; mod \; N$$

$$m = \underbrace{c * c * c * \cdots * c}_{\text{d times}} \; mod \; N$$

Modular Exponentiation

$$m = c^d \ mod \ N$$

$$d = 10101010011101010110010010001001$$

```
modpow(c, d, N) {

    result = 1;

    for (i = 1 to |d|) {
        if (the ith bit of d is 1) {
            result = (result * c) mod N;
        }

        c = c² mod N;
    }

    return result;
}
```

Modular Exponentiation

$$m = c^d \bmod N$$

$$d = 10101010011101010110010010001001$$

```
modpow(c, d, N) {

    result = 1;

    for (i = 1 to |d|) {
        if (the ith bit of d is 1) {
            result = (result * c) mod N;
        }

        c = c² mod N;
    }

    return result;
}
```

Modular Exponentiation

$$m = c^d \bmod N$$

$$d = 10101010011101010110010010001001$$

```
modpow(c, d, N) {

    result = 1;

    for (i = 1 to |d|) {
        if (the ith bit of d is 1) {
            result = (result * c) mod N;
        }

        c = c² mod N;
    }

    return result;
}
```

Modular Exponentiation

$$m = c^d \bmod N$$

$$d = 10101010011101010110010010001001$$

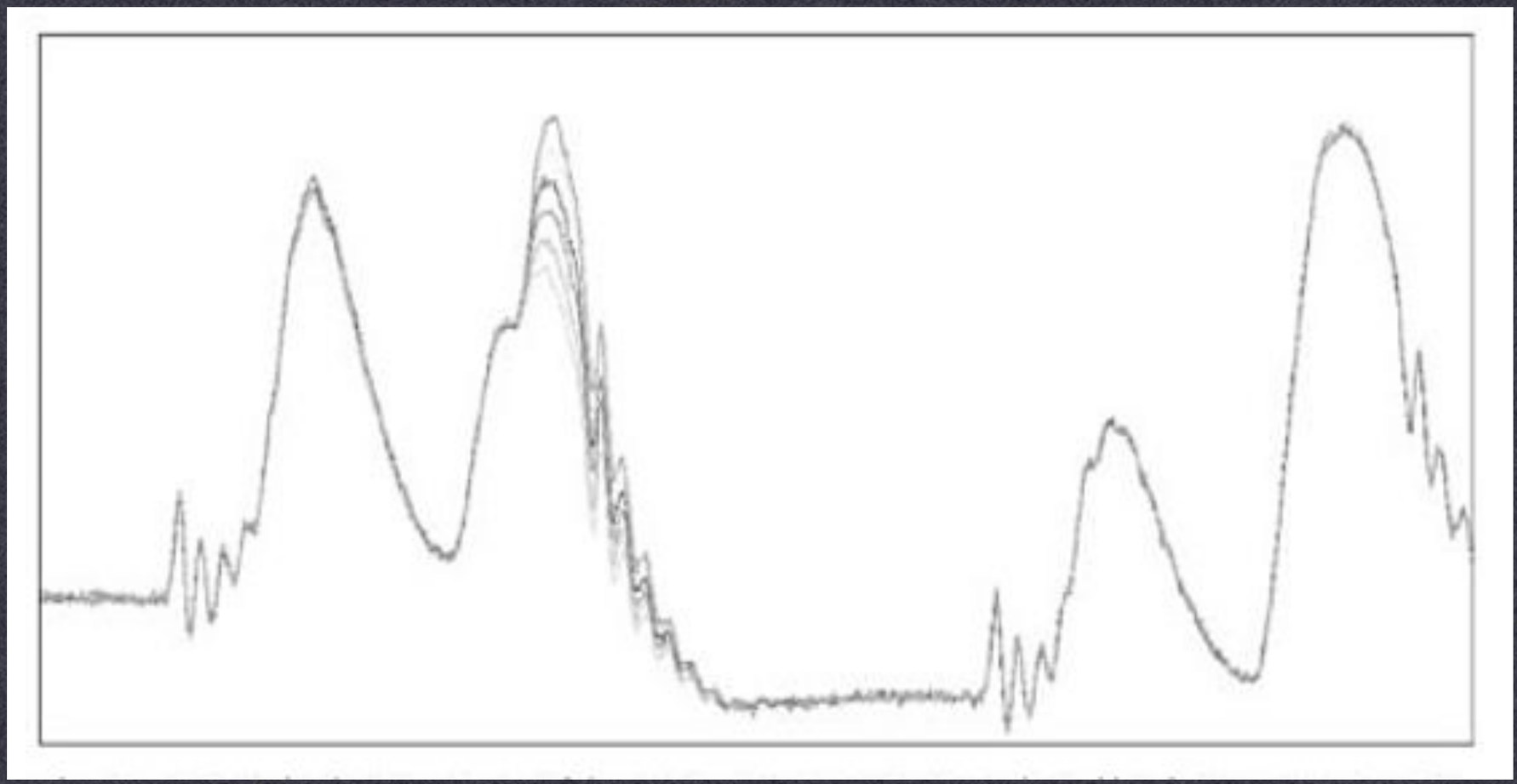
```
modpow(c, d, N) {

    result = 1;

    for (i = 1 to |d|) {
        if (the ith bit of d is 1) {
            result = (result * c) mod N;
        }

        c = c² mod N;
    }

    return result;
}
```

Expensive Operation

Simple Power Analysis

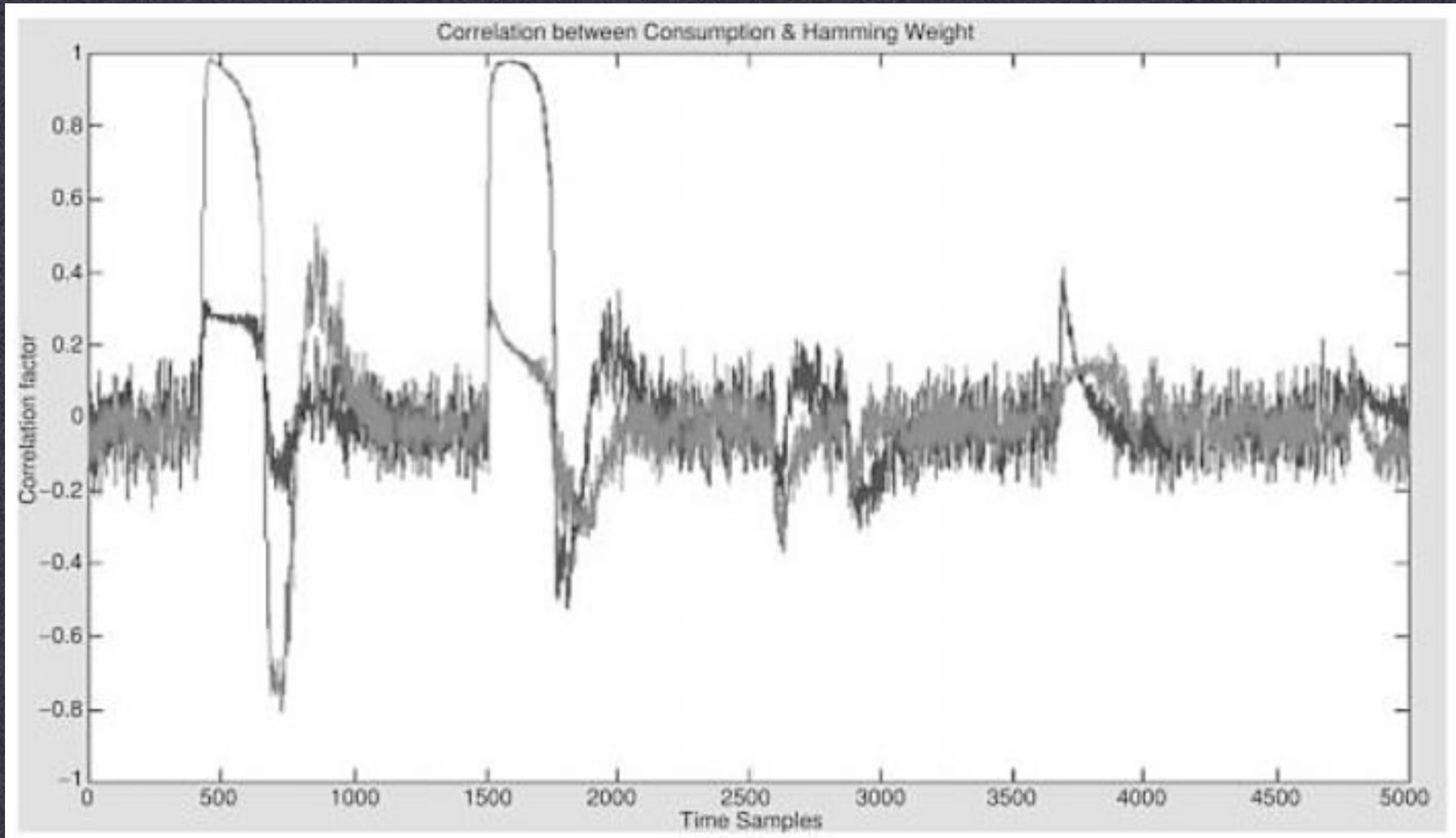
Source: Bidgoli, Handbook of Information Security

Differential Power Analysis



Source: Bidgoli, Handbook of Information Security

Differential Power Analysis



Source: Bidgoli, Handbook of Information Security

Modular Exponentiation

$$m = c^d \ mod \ N$$

$$d = 10101010011101010110010010001001$$

```
modpow(c, d, N) {

    result = 1;

    for (i = 1 to |d|) {
        c = c² mod N;

        if (the iᵗʰ bit of d is 1) {
            result = (result * c) mod N;
        }
    }

    return result;
}
```

Expensive Operation

Modular Exponentiation

$$m = c^d \ mod \ N$$

$$d = 10101010011101010110010010001001$$

```
modpow(c, d, N) {

    result = 1;

    for (i = 1 to |d|) {
        if (the ith bit of d is 1) {
            result = (result * c) mod N;
        }

        c = c² mod N;
    }

    return result;
}
```

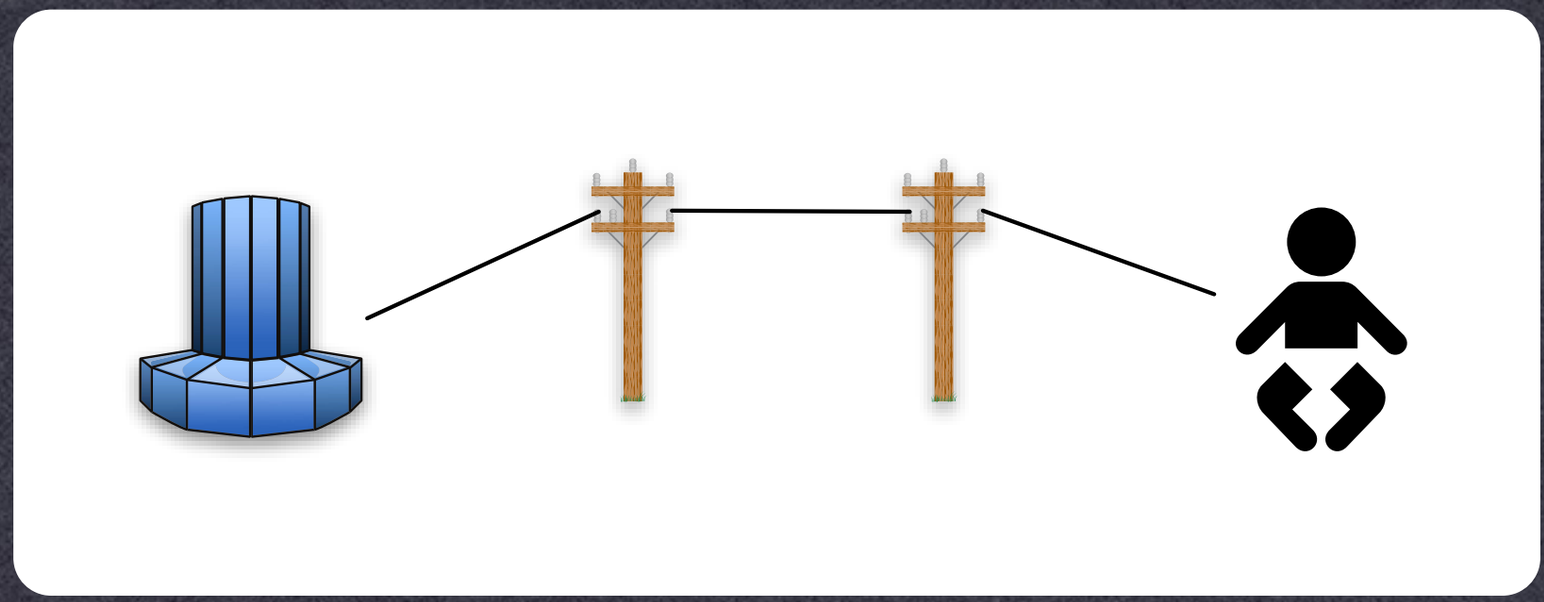
Expensive Operation

Kocher's Timing Attack

- Assume that for some values of (a * b), multiplication takes unusually long
- Given the first b bits, compute intermediate value
  "result" up to that point
- If the next bit of d = 1, then calc is (result * c)
- If this is expected to be slow,
  but response is fast then the next bit of d != 1

Remote Timing Attacks

- Boneh & Brumley
  - Remote attack on RSA-CRT as implemented in OpenSSL
  - Optimization, uses knowledge of p,q



CRT

- If one knows the remainder of division by several integers (p, q) then one can compute the remainder of division by products (p*q)
  — assuming p, q are co-prime
- Why does this help us?

Solutions

- Quantization:
  - All operations take the same time
  - Hard to do without sacrificing performance
- Blinding:
  - Prevents attacker from selecting ciphertext (that is processed with the secret key)

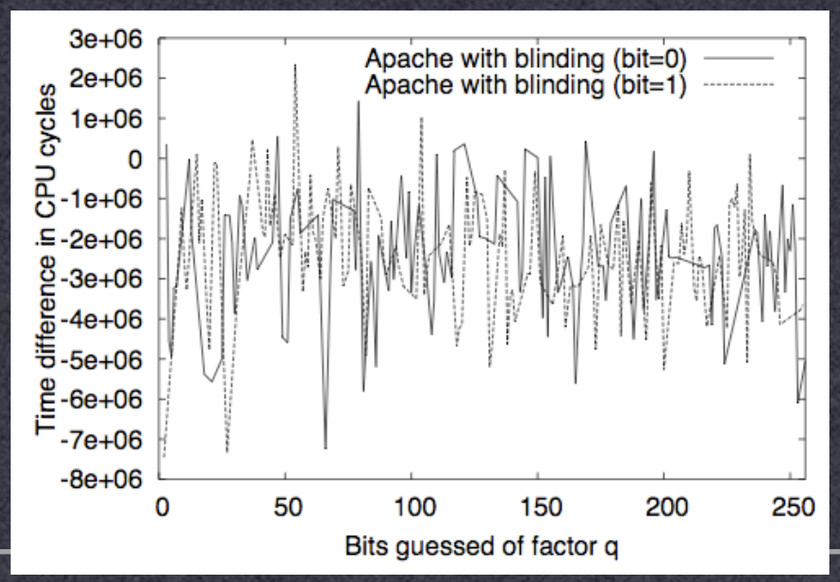
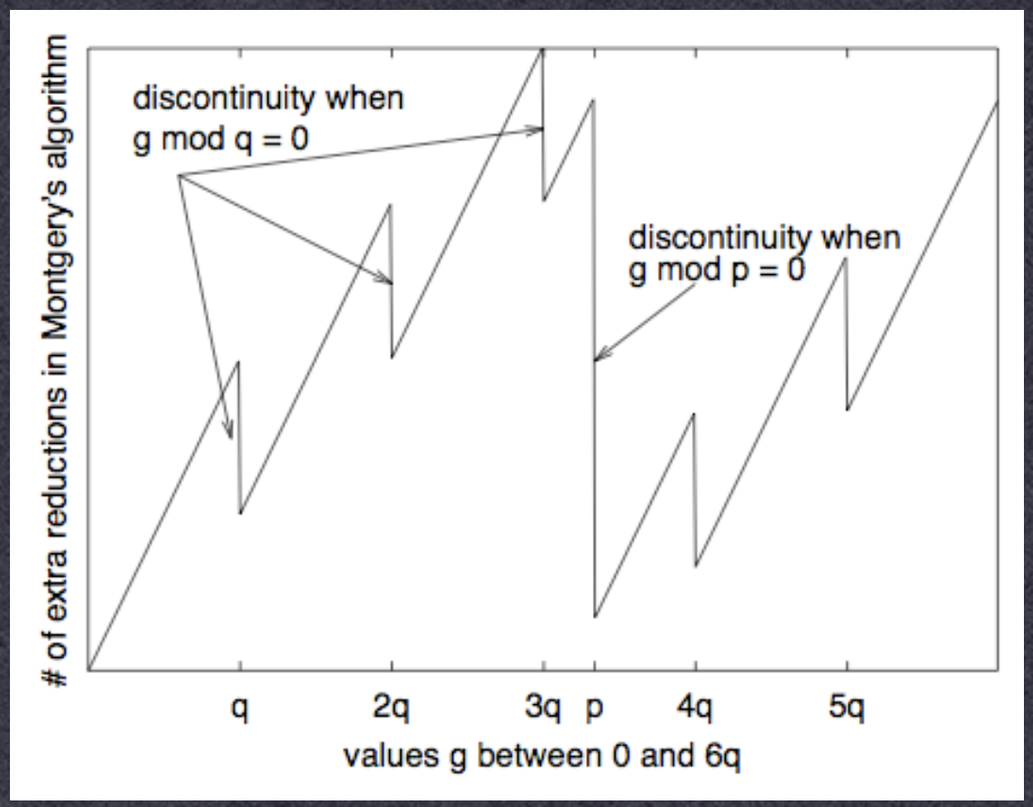
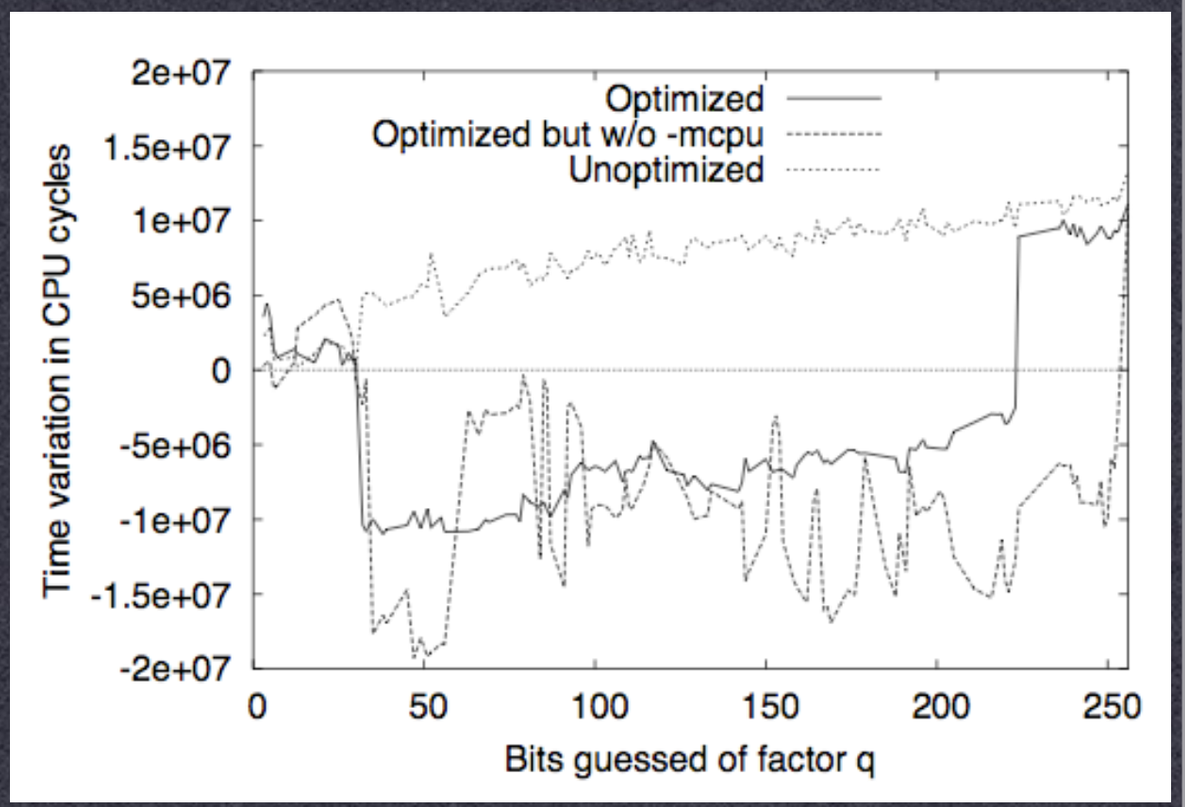


Image Source: Boneh, Brumley: Remote Timing Attacks are Practical

Remote Timing Attacks

- **Boneh & Brumley**
  - **By repeating the timing measurements, they were able to extract a secret key after several million samples**





Source: Boneh, Brumley: Remote Timing Attacks are Practical

Windowed Exponentiation

$$m = c^d \bmod N$$

$$d = 10101010011101010110010010001001$$

- Handles the input in larger “chunks”
  - Fixed or variable sized
  - Can speed up by pre-computing some values

-e.g., c^2, c^3, ..., c^16

Windowed Exponentiation

$$m = c^d \bmod N$$

$$d = 10101010011101010110010\,0100\,1001$$

- Handles the input in larger “chunks”
  - Fixed or variable sized
  - Can speed up by pre-computing some values
    - e.g., c^2, c^3, ..., c^16

Windowed Exponentiation

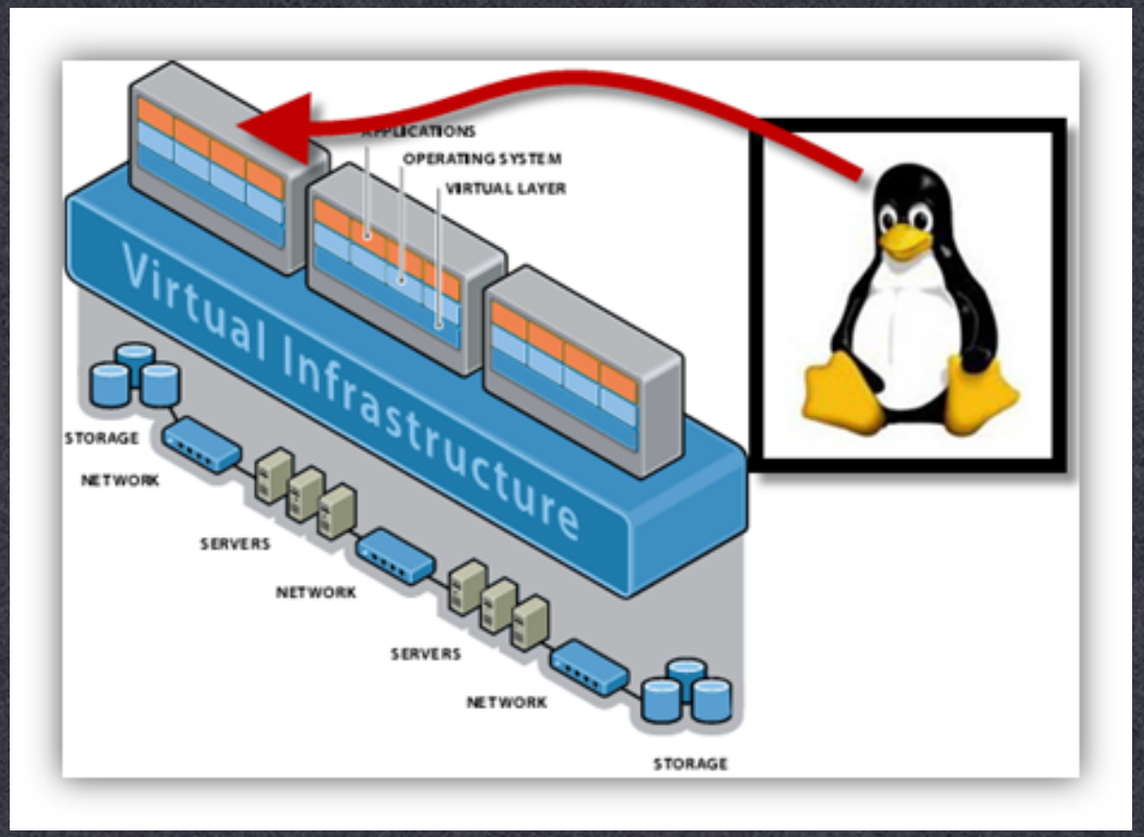
$$m = c^d \ mod \ N$$

$$d = 10101010011101010110010001001001$$

- Handles the input in larger “chunks”
  - Fixed or variable sized
  - Can speed up by pre-computing some values
    - e.g., c^2, c^3, ..., c^16

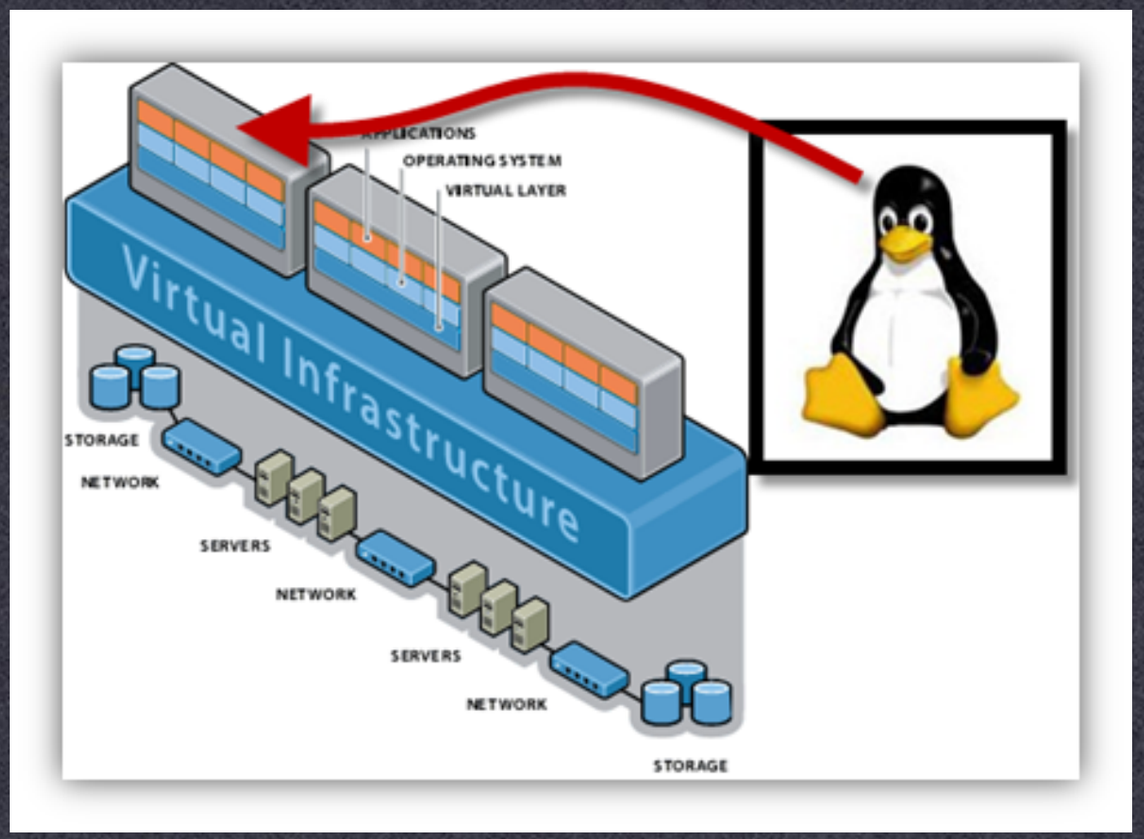
Cache Timing Attacks

- Observation
  - When we use pre-computed tables, this implies a memory access
  - This takes time
  - Unless the data is cached!



Cache Timing Attacks

- AES
  - When we use pre-computed tables, this implies a memory access
  - This takes time
  - Unless the data is cached!



Hypervisor attacks

- **Observation**
  - **This applies to code too!**

SquareMult(x, e, N):
  let $e_n, \ldots, e_1$ be the bits of e
$y \leftarrow 1$
for $i = n$ down to 1 {
  $y \leftarrow$ Square(y) (S)
  $y \leftarrow$ ModReduce(y, N) (R)
  if $e_i = 1$ then {
    $y \leftarrow$ Mult(y, x) (M)
    $y \leftarrow$ ModReduce(y, N) (R)
  }
}
return y

The 9 Lives of Bleichenbacher's CAT: New Cache ATtacks on TLS Implementations

Eyal Ronen*, Robert Gillham†, Daniel Genkin‡, Adi Shamir¶, David Wong§, and Yuval Yarom†**

*Tel Aviv University, †University of Adelaide, ‡University of Michigan, ¶Weizmann Institute, §NCC Group, **Data61

Listing 2. Pseudocode of RSA_

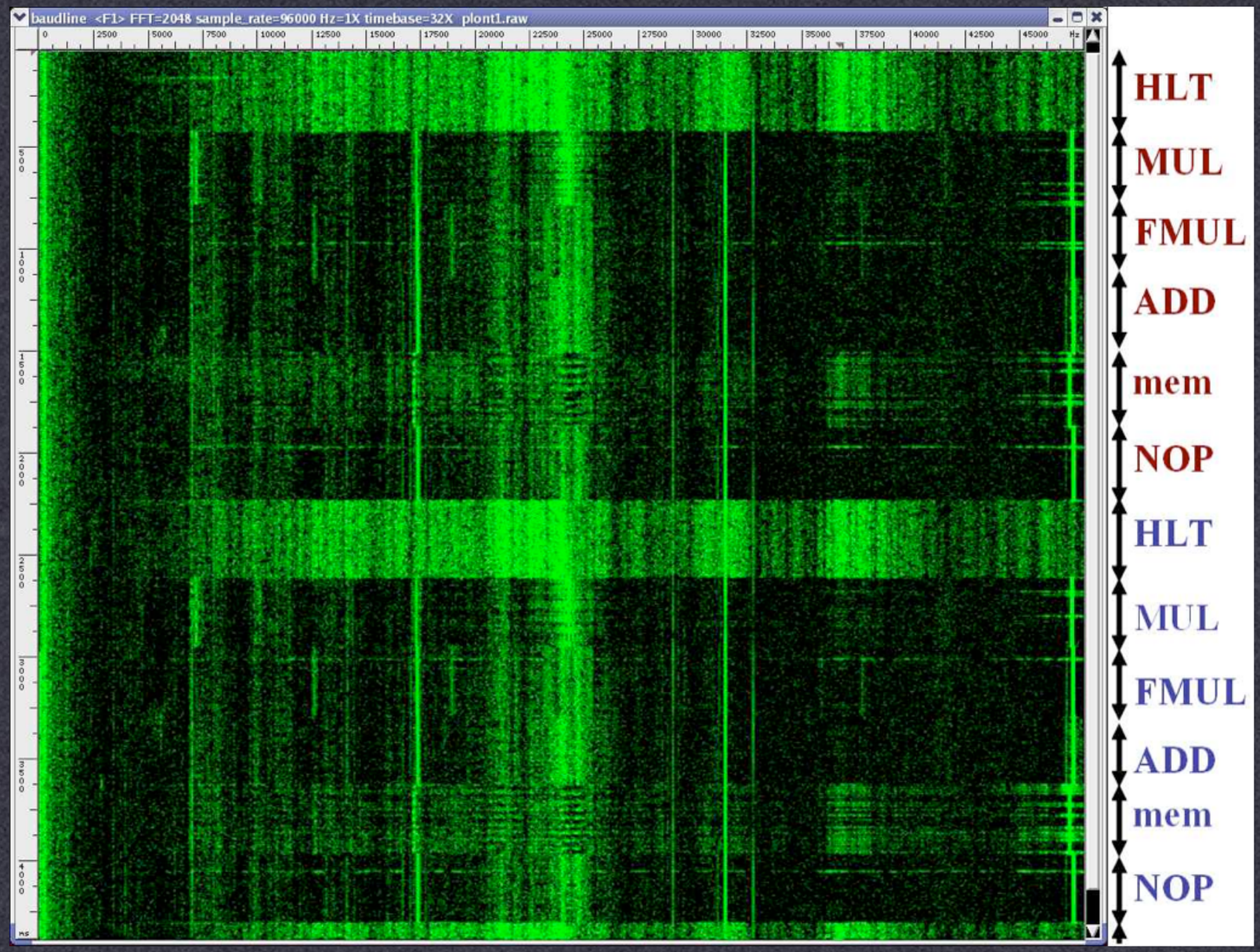
```
1  int SSLDecodeRSAKeyExchange(keyExchange, ctx){
2    keyRef = ctx->signingPrivKeyRef;
3    src  = keyExchange.data;
4    localKeyModulusLen = keyExchange.length;
5    ... // addtional inilization code omitted
6
7    err = sslRsaDecrypt(keyRef, src,
       localKeyModulusLen,
8      ctx->preMasterSecret.data,
       SSL_RSA_PREMASTER_SECRET_SIZE, &outputLen);
9    if(err != errSSLSuccess) {
10     /* possible Bleichenbacher attack */
11     sslLogNegotiateDebug("SSLDecodeRSAKeyExchange:
12                                 RSA decrypt fail");
13   } else if(outputLen !=
14                   SSL_RSA_PREMASTER_SECRET_SIZE) {
15     sslLogNegotiateDebug("SSLDecodeRSAKeyExchange:
       premaster secret size error");
16     // not passed back to caller
17     err = errSSLProtocol;
18   }
19   if(err == errSSLSuccess) {
20       ... // (omitted for brevity)
21   }
22   if(err != errSSLSuccess) {
23     ... // (omitted for brevity)
24     sslRand(&tmpBuf);
25   }
26   /* in any case, save premaster secret (good or
       bogus) and proceed */
27   return errSSLSuccess;
28 }
```

attack [69], as implemented in the Mastik toolkit [68].

Monitoring Locations. To reduce the likelihood of errors, we monitor both the call-site to RSAerr (Line 25 of Listing 2) and the code of the function RSAerr. Monitoring each of these locations may generate false positives, i.e. indicate access when the plaintext is PKCS #1 v1.5 conforming. The former results in false positives because the call to RSAerr shares the cache line with the surrounding code, that is always invoked. The latter results in false positives when unrelated code logs an error. By only predicting a non-conforming plaintext if *both* locations are accessed within a short interval, we reduce the likelihood of false positives. We note that this technique is very different to the approach of Genkin et al. [30] of monitoring two memory locations to reduce false negative errors due to a race between the victim and the attacker [6]. Unlike us, they assume access if *any* of the monitored locations is accessed.

Experimental Results. Overall, our technique achieves a

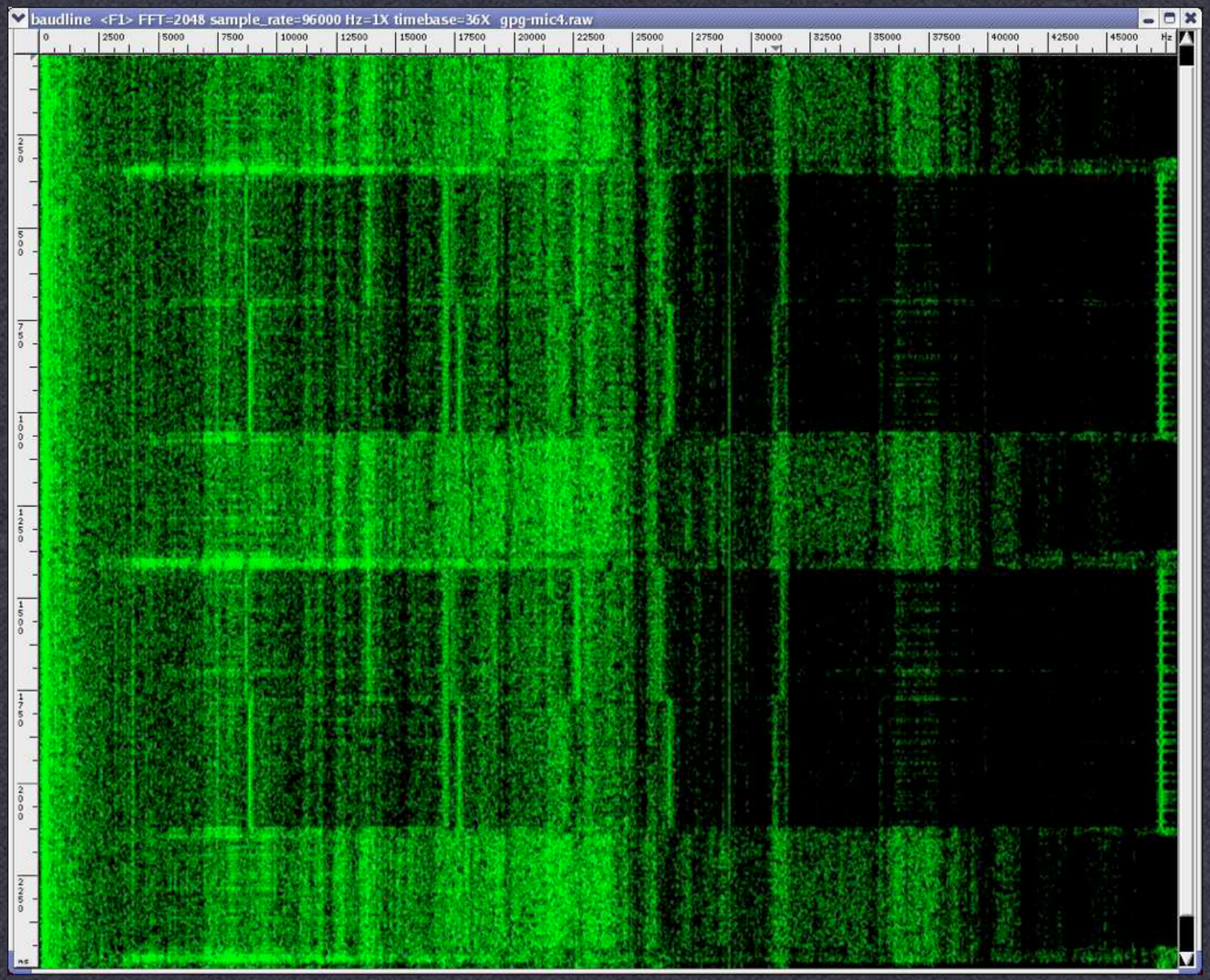
Acoustic Side Channels



Source: Tromer, Acoustic Cryptanalysis: http://people.csail.mit.edu/tromer/acoustic/

Acoustic Side-Channels

GPG RSA-CRT signature #1:



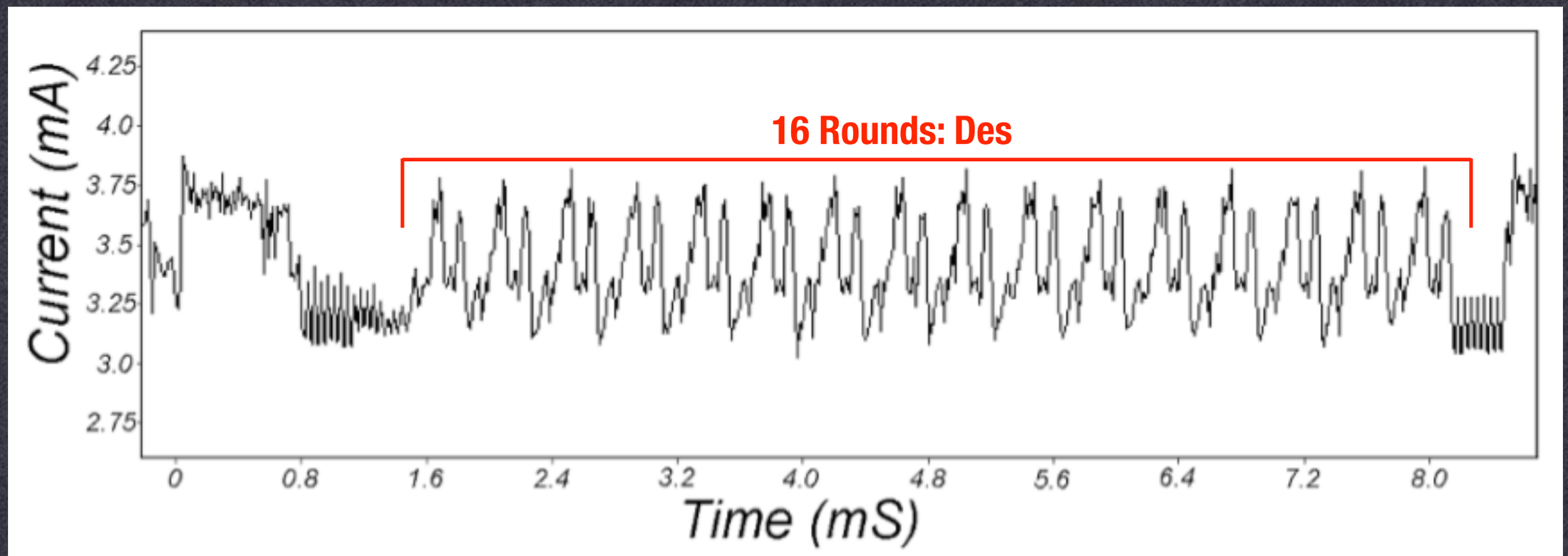
mod q

mod p

(repeated)

Source: Tromer, Acoustic Cryptanalysis: http://people.csail.mit.edu/tromer/acoustic/

Reverse Engineering



Source: Kocher et al. 1999

Back to KeyLoq

- **KeyLoq attack has an unhappy coda:**
  - **Turns out that all keys for a given manufacturer are derived from the same MK**

$$k = pad(ID, seed) \oplus MK$$

  - **Key derivation function based on XOR :(**
  - **By observing 2 interactions between key/car we can derive the MK and thus steal any car**

Speculative execution

- A new class of side-channel attacks
  - Not specific to cryptography
  - But worth mentioning because they are so powerful!

Speculative execution

- A new class of side-channel attacks
  - Not specific to cryptography
  - But worth mentioning because they are so powerful
  - Spectre, Meltdown, Foreshadow

How do we make CPUs faster?

- So many of the hardware gains are used up
- Today, we get major performance from parallelization and optimization: reduce memory latency, pipelining, speculative execution

How do we make CPUs faster?

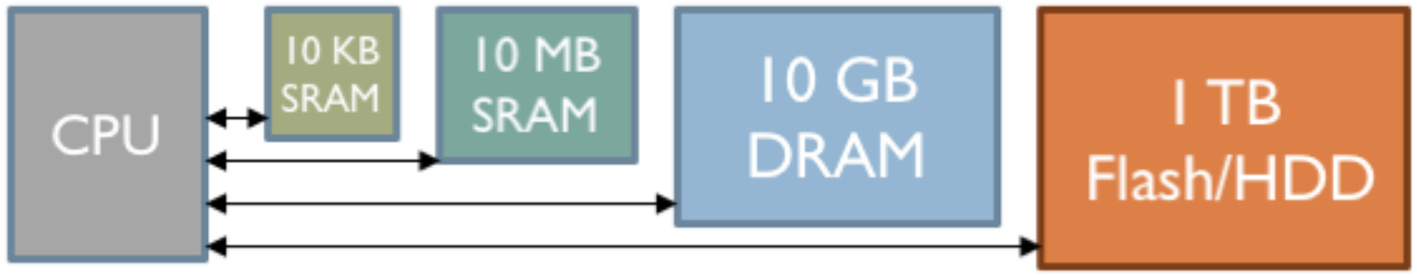
- Basically all of the hardware gains are used up
- Today, we get major performance from parallelization and optimization: reduce memory latency, pipelining, speculative execution

Many slides and examples borrowed and inspired by Kocher: Spectre attack slides
https://www.ieee-security.org/TC/SP2019/SP19-Slides-pdfs/Paul_Kocher_01_-_Spectre_Attacks-IEEE-SecurityPrivacy_v05.pdf

Memory Hierarchy Interface

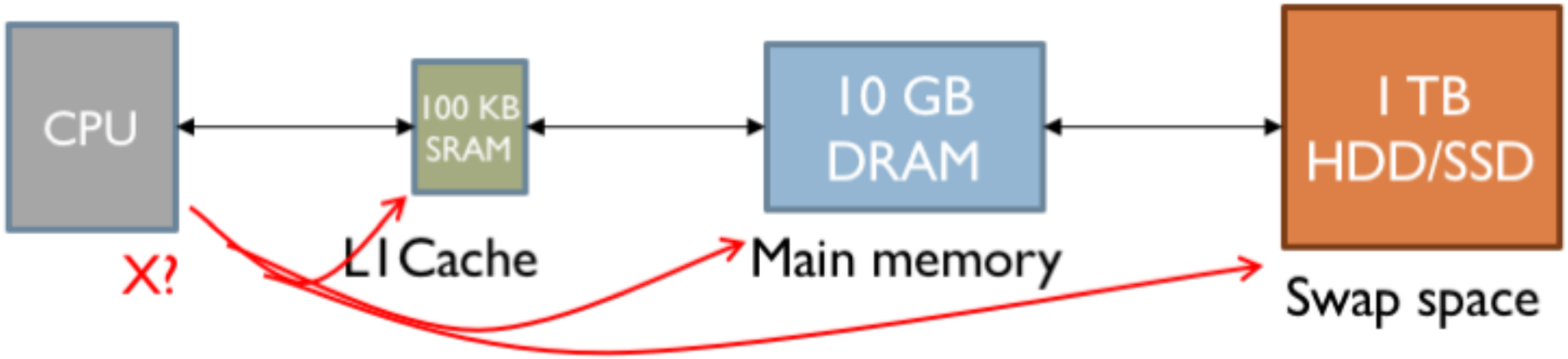
Approach 1: Expose Hierarchy

- Registers, SRAM, DRAM, Flash, Hard Disk each available as storage alternatives
- Tell programmers: "Use them cleverly"



Approach 2: Hide Hierarchy

- Programming model: Single memory, single address space
- Machine transparently stores data in fast or slow memory, depending on usage patterns



From: https://computationstructures.org/lectures/caches/caches.html#15

Consider this program

- What do we do if x is not in cache yet?

```
if (x == 1) {
    abc…
} else {
    xyz…
}
```

Consider this program

- What do we do if x is not in cache yet?

```
if (x == 1) {
    abc…
} else {
    xyz…
}
```

slow solution: wait until x loads from DRAM

Consider this program

- What do we do if x is not in memory yet?

```
if (x == 1) {
    abc…
} else {
    xyz…
}
```

fast solution: speculatively execute both branches (or one guess), discard invalid work

Conditional branch (Variant 1) attack

```
if (x < array1_size)
    y = array2[array1[x]*512];
```

Attack scenario:

- Code runs in a trusted context
- Adversary wants to read memory and controls unsigned integer `x`
- Branch predictor will expect `if()` to be true (e.g. because prior calls had `x` < `array1_size`)
- `array1_size` and `array2[]` are not in cache

Memory & Cache Status

`array1_size = 00000008`

Memory at `array1` base address:
8 bytes of data (value doesn't matter)
[… lots of memory up to `array1` base+N…]
09 F1 98 CC 90 . . . (something secret)

```
array2[ 0*512]
array2[ 1*512]
array2[ 2*512]
array2[ 3*512]
array2[ 4*512]
array2[ 5*512]
array2[ 6*512]
array2[ 7*512]
array2[ 8*512]
array2[ 9*512]
array2[10*512]
array2[11*512]
  •••
```

Contents don't matter
only care about cache ***status***

Uncached Cached

Many slides and examples borrowed and inspired by Kocher: Spectre attack slides
https://www.ieee-security.org/TC/SP2019/SP19-Slides-pdfs/Paul_Kocher_01_-_Spectre_Attacks-IEEE-SecurityPrivacy_v05.pdf

Conditional branch (Variant 1) attack

```
if (x < array1_size)
    y = array2[array1[x]*512];
```

Attacker calls victim code with `x`=N (where N > 8)

- Speculative exec while waiting for `array1_size`
  - Predict that if() is true
  - Read address (`array1` base + `x`) w/ out-of-bounds `x`
  - Read returns secret byte = **09** (fast – in cache)
  - Request memory at (`array2` base + **09***512)
  - Brings `array2[`**09**`*512]` into the cache
  - Realize if() is false: discard speculative work
- Finish operation & return to caller

Attacker times reads from `array2[i*512]`

- Read for `i`=**09** is fast (cached), revealing secret byte

Memory & Cache Status

`array1_size = 00000008`

Memory at `array1` base address:
8 bytes of data (value doesn't matter)
[… lots of memory up to `array1` base+N…]
09 F1 98 CC 90... (something secret)

`array2[ 0*512]`
`array2[ 1*512]`
`array2[ 2*512]`
`array2[ 3*512]`
`array2[ 4*512]`
`array2[ 5*512]`
`array2[ 6*512]`
`array2[ 7*512]`
`array2[ 8*512]`
`array2[ 9*512]`
`array2[10*512]`
`array2[11*512]`
…

Contents don't matter
only care about cache ***status***

Uncached Cached

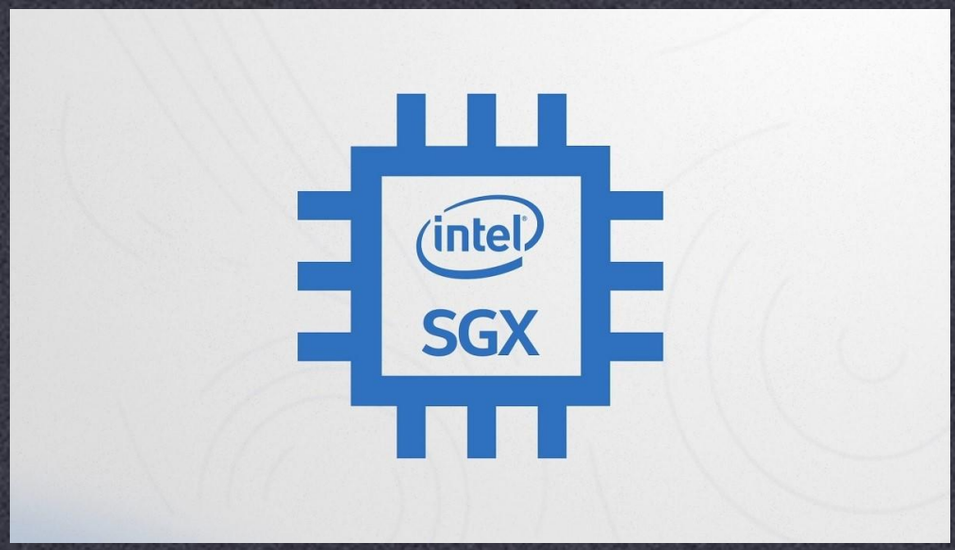
Many slides and examples borrowed and inspired by Kocher: Spectre attack slides
https://www.ieee-security.org/TC/SP2019/SP19-Slides-pdfs/Paul_Kocher_01_-_Spectre_Attacks-IEEE-SecurityPrivacy_v05.pdf

Why does this matter?

- We can learn information about data that we should not be able to access
- For example:
  - Kernel secrets
  - Secrets in the same process (keys)
  - Other applications
  - It <u>should</u> run user space code
    But attacker can cause it to leak information about data in the kernel

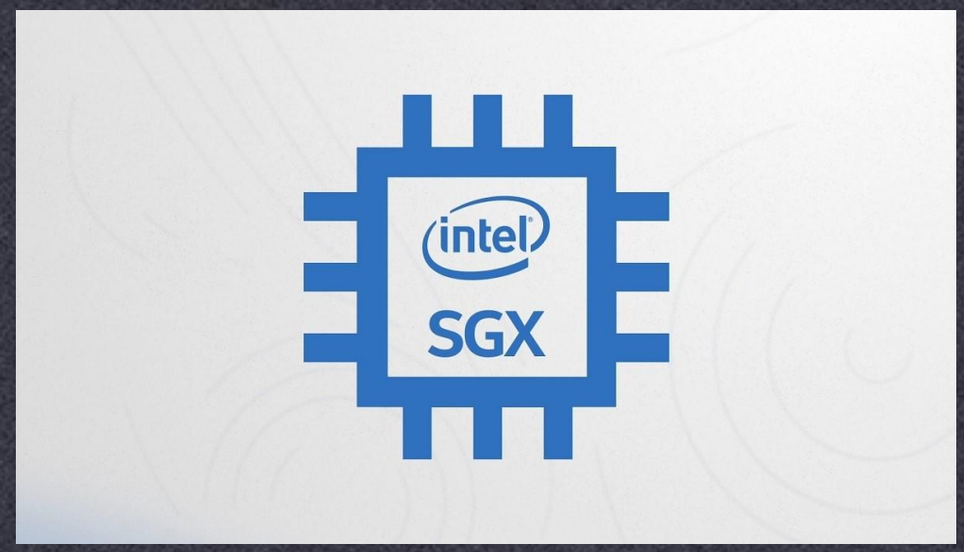
Intel SGX

- "Virtual trusted hardware" inside of an Intel CPU
- Basic ideas:
  - To run an SGX program ("enclave") turn in an isolation mode that is even stronger than Ring0
  - OS cannot read enclave memory, enclave cannot read OS/app memory
  - Enforced by microcode



What software can we run?

- Enclave software has no access to I/O (network, keyboard, mouse). It has to interface with a real application via a strict API
- Every SGX (“enclave program”) is hashed, then digitally signed under a certificate chain held by Intel



How does the enclave store data persistently?

- Enclave can’t access disks directly
- It has to send data out to an application via the API
- But if that program gets compromised, how does the enclave verify it’s safe?

How does the enclave store data persistently?

- Answer: lots of crypto
- Ever enclave gets its own secret encryption keys (generated by SGX base OS)
- Keys are generated as a combination of:
  Enclave software hash +
  System unique identifier +
  A system root secret key known to SGX

How does the enclave store data persistently?

- Each enclave can “seal” (encrypt/MAC) data under this secret key, send it out to application
- Application can store the encrypted/authenticated data for the enclave (e.g., on disk), hand back to enclave next time it boots

How does the enclave store data persistently?

- Each enclave can “seal” (encrypt/MAC) data under this secret key, send it out to application
- Application can store the encrypted/authenticated data for the enclave (e.g., on disk), hand back to enclave next time it boots
- What about replay attacks?

Intel SGX attestation

- This all works great when I'm running software on my own computer
- What happens if I want to use Intel SGX in the cloud?
- How do I know my software is really being operated by SGX and not some malicious software pretending to be SGX?

Solution: attestation

- Every Intel processor has a secret signing key baked in at the factory*
- An enclave can ask the SGX OS to sign any block of data ("attestation")
- Signature includes:
  - Enclave hash (enforced by SGX OS)
  - Arbitrary data blob specified by application
  - Any other measurements, e.g., RAM hash
  - Certificate signed by Intel

So TL;DR

- As long as every single Intel signing key stays secret, a signature produced by a random SGX-enabled processor can be trusted
- Converse: if anyone steals a single SGX signing key (e.g., by attacking hardware) they can fake all these signatures, pretend to be an SGX enclave when they're not
- How do we deal with that?

Foreshadow

- As long as every single Intel signing key stays secret, a signature produced by a random SGX-enabled processor can be trusted
- Converse: if anyone steals a single SGX signing key (e.g., by attacking hardware) they can fake all these signatures, pretend to be an SGX enclave when they're not
- How do we deal with that?

FORESHADOW

Breaking the Virtual Memory Abstraction with Transient Out-of-Order Execution

Read the paper | Cite | Watch a demo

Introduction

Foreshadow is a speculative execution attack on Intel processors which allows an attacker to steal sensitive information stored inside personal computers or third party clouds. Foreshadow has two versions, the original attack designed to extract data from SGX enclaves and a Next-Generation version which affects Virtual Machines (VMs), hypervisors (VMM), operating system (OS) kernel memory, and System Management Mode (SMM) memory.

Foreshadow AaaS @ForeshadowAaaS · Aug 15, 2018

Hi, I'm the Foreshadow AaaS bot! :-) I react to tweets I'm tagged in by providing a genuine Intel attestation that can be verified against Intel Attestation Service (IAS).

8 23 50

Show this thread

Replies

Foreshadow AaaS @ForeshadowAaaS · Aug 15, 2018

Here is your attestation that "RT: Hi, I'm the Foreshadow AaaS bot! :-) I react to tweets I'm tagged in by providing a genuine " is a genuine SGX enclave github.com/TeeAaas/Foresh...

1 1