Practical Cryptographic Systems

Side Channel Attacks

Instructor: Matthew Green

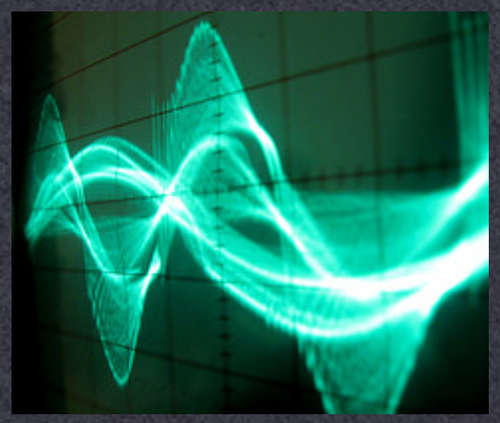
Housekeeping:

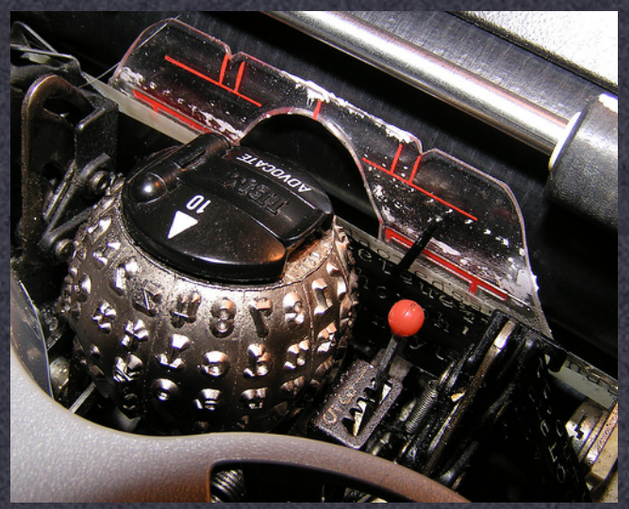
- Exam Weds

Current Events

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



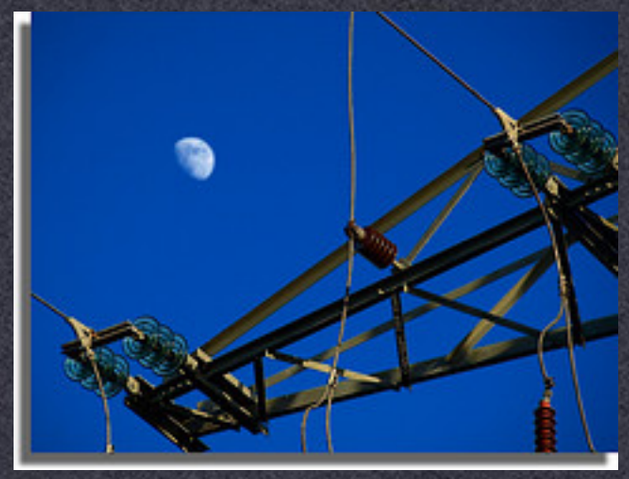


Side Channels

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

Common Examples

- Timing
- Power Consumption
- RF Emissions
- Audio



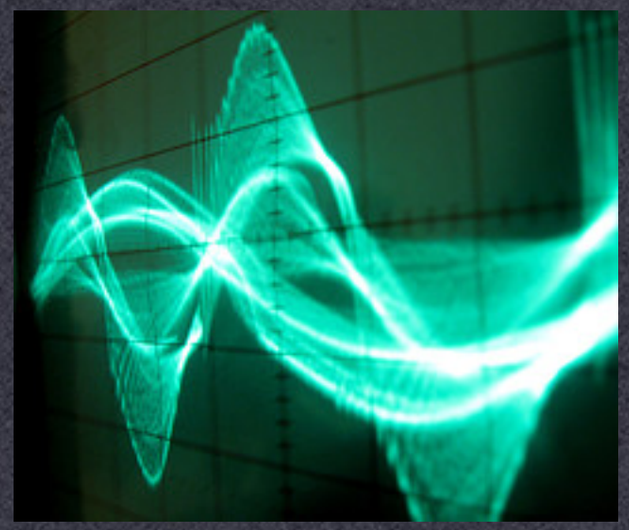


Photo Credits: Clock - Flickr user laffy4k, Waveform - Flickr user altemark, Power Line - Flickr user Kliefi. All used under a Creative Commons license.

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

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

$$d = 10101010011101010110010010001001$$

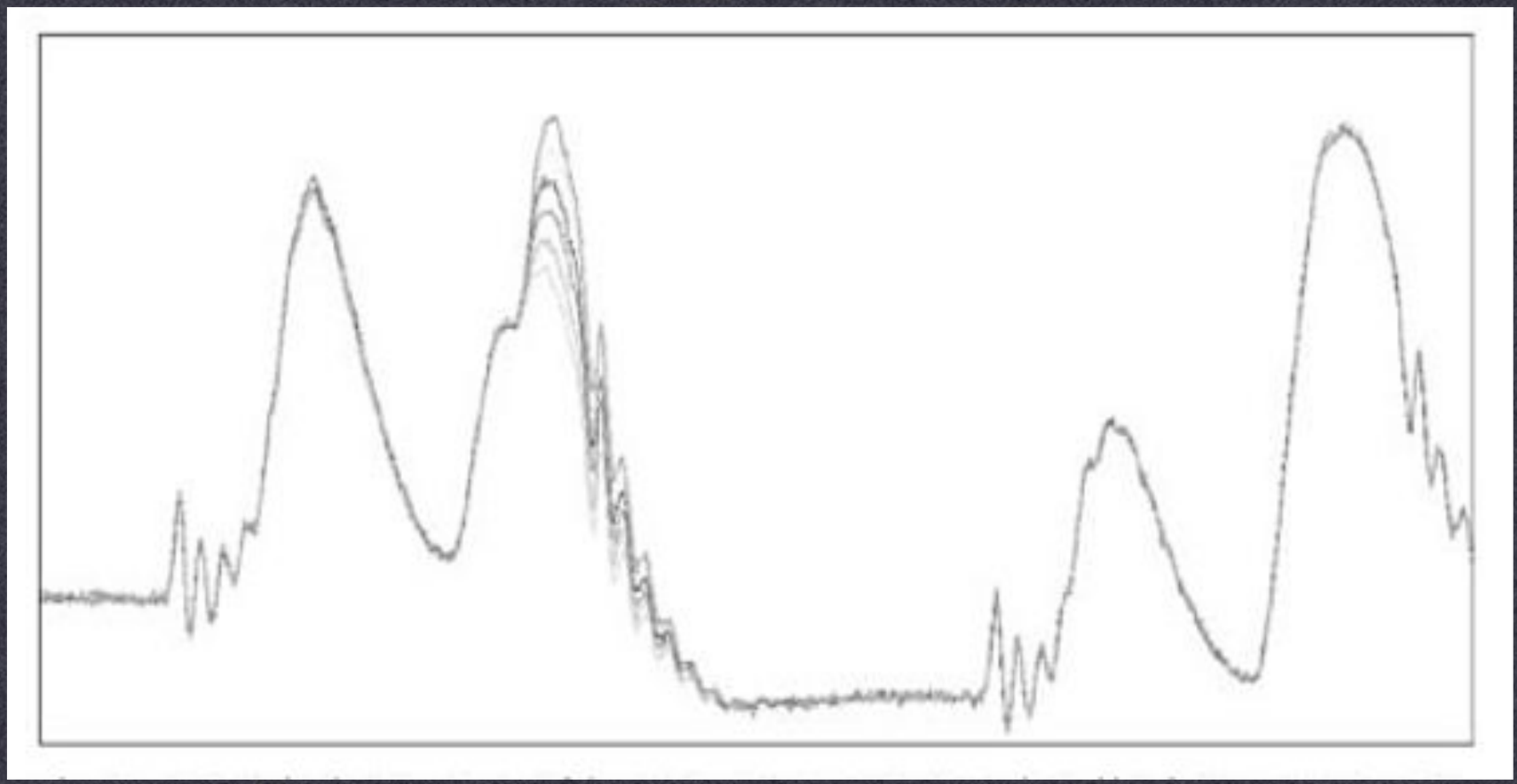
```
modpow(c, d, N) {

    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

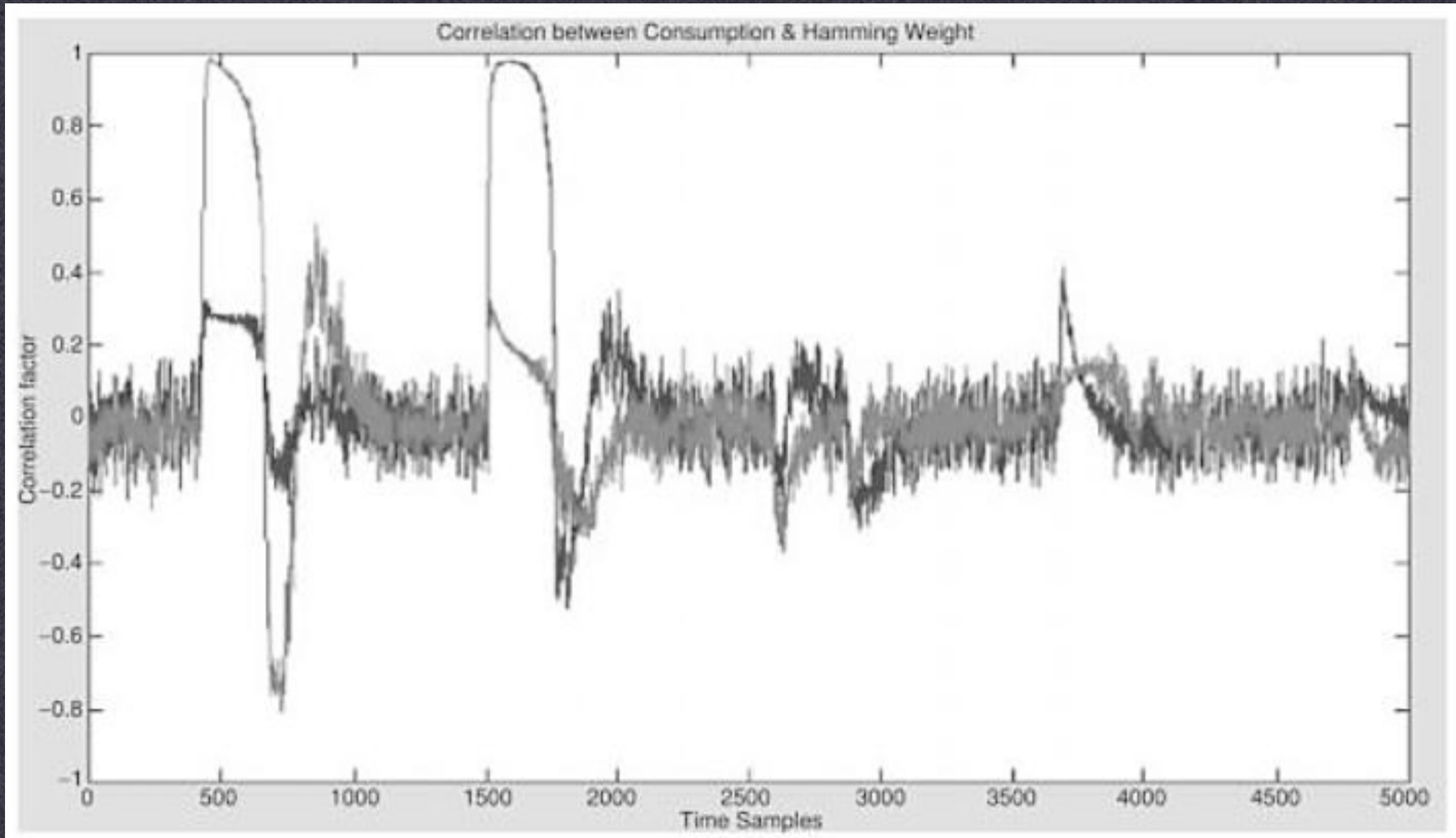
Source: Bidgoli, Handbook of Information Security

Differential Power Analysis



Source: Bidgoli, Handbook of Information Security

Differential Power Analysis



Source: Bidgoli, Handbook of Information Security

Modular Exponentiation

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

$$d = 10101010011101010110010010010001$$

```
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

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

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;
}
```

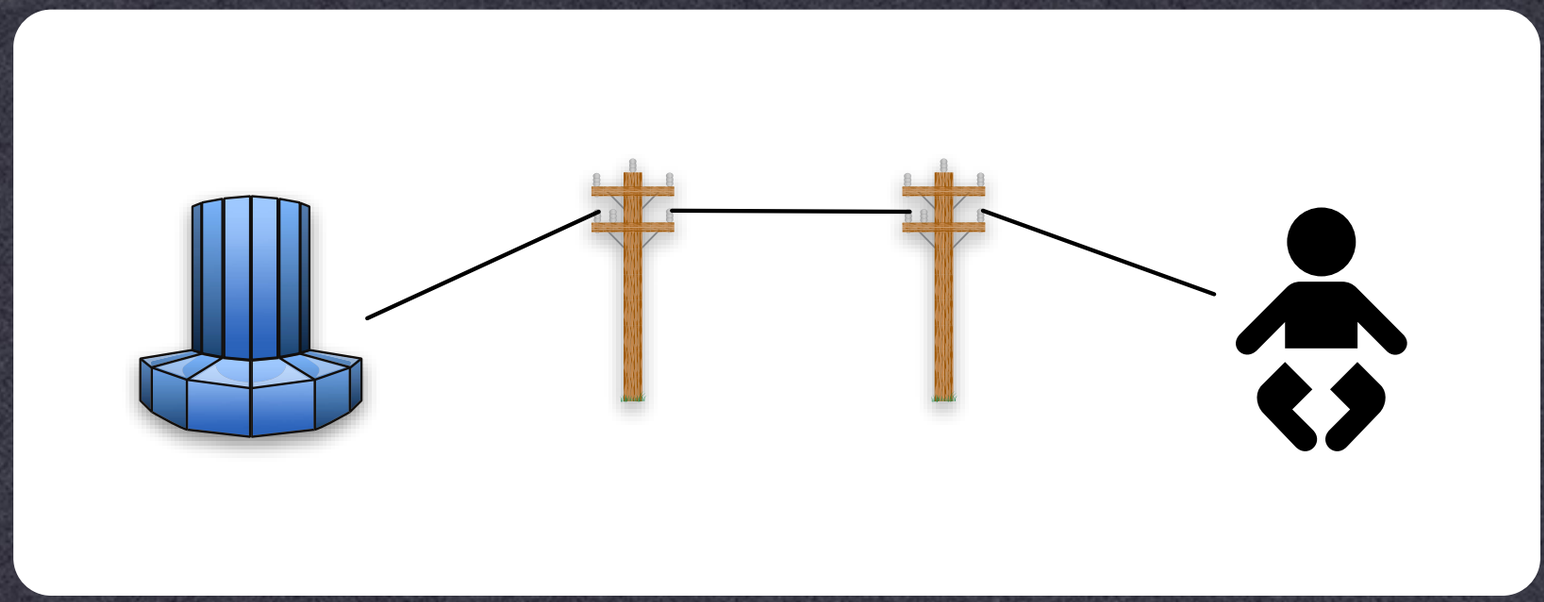
Expensive Operation

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



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)

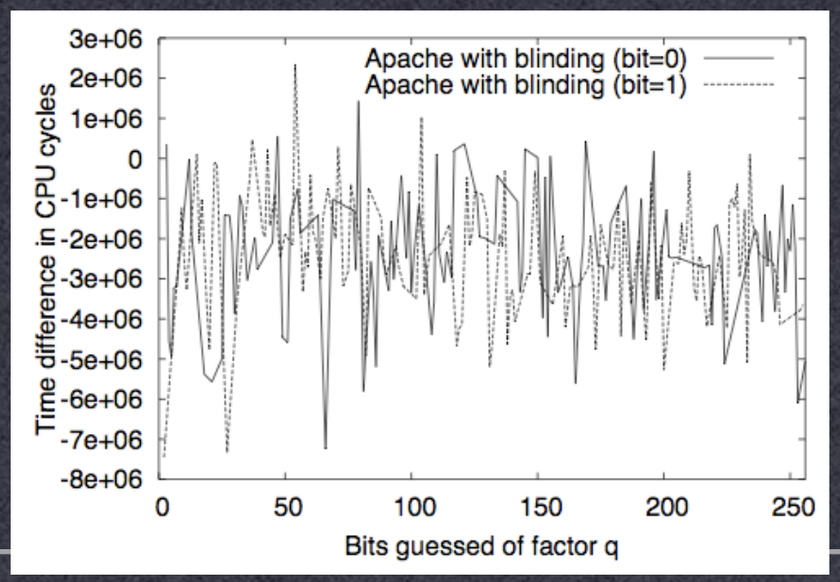
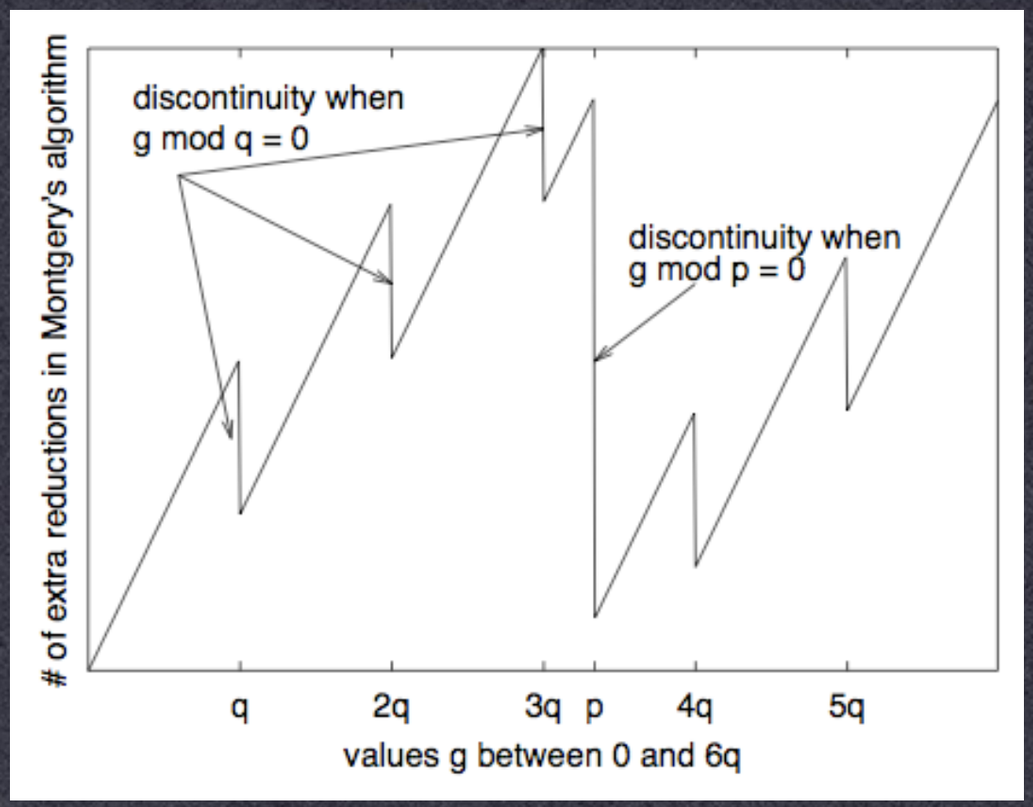
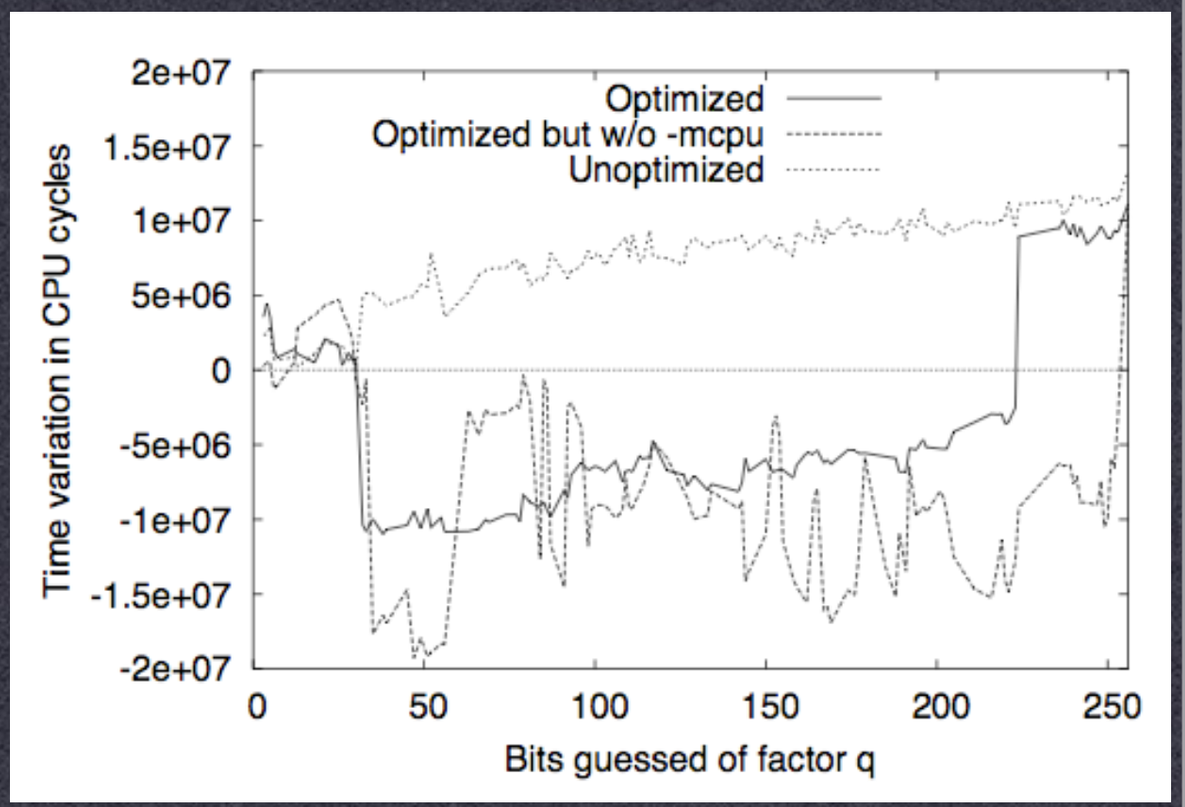


Image Source: Boneh, Brumley: Remote Timing Attacks are Practical

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

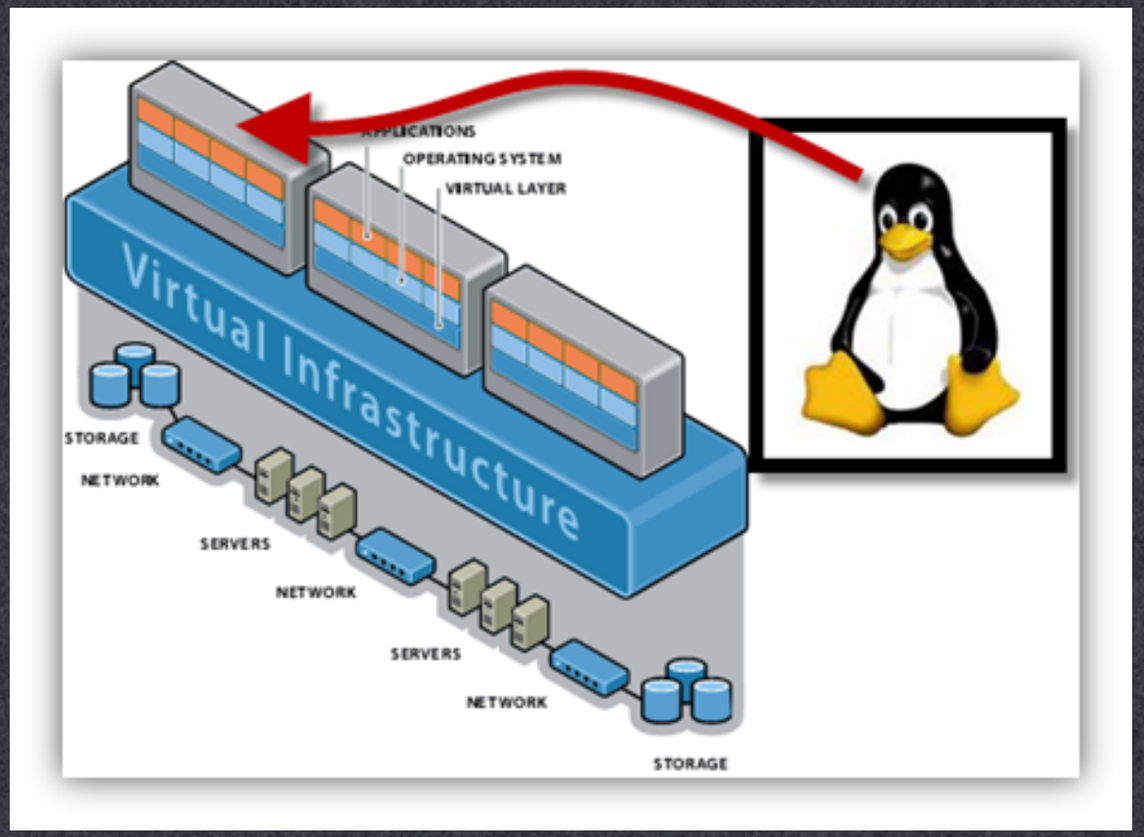
$$m = c^d \ mod \ N$$

$$d = 10101010011101010110010001001001$$

- 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

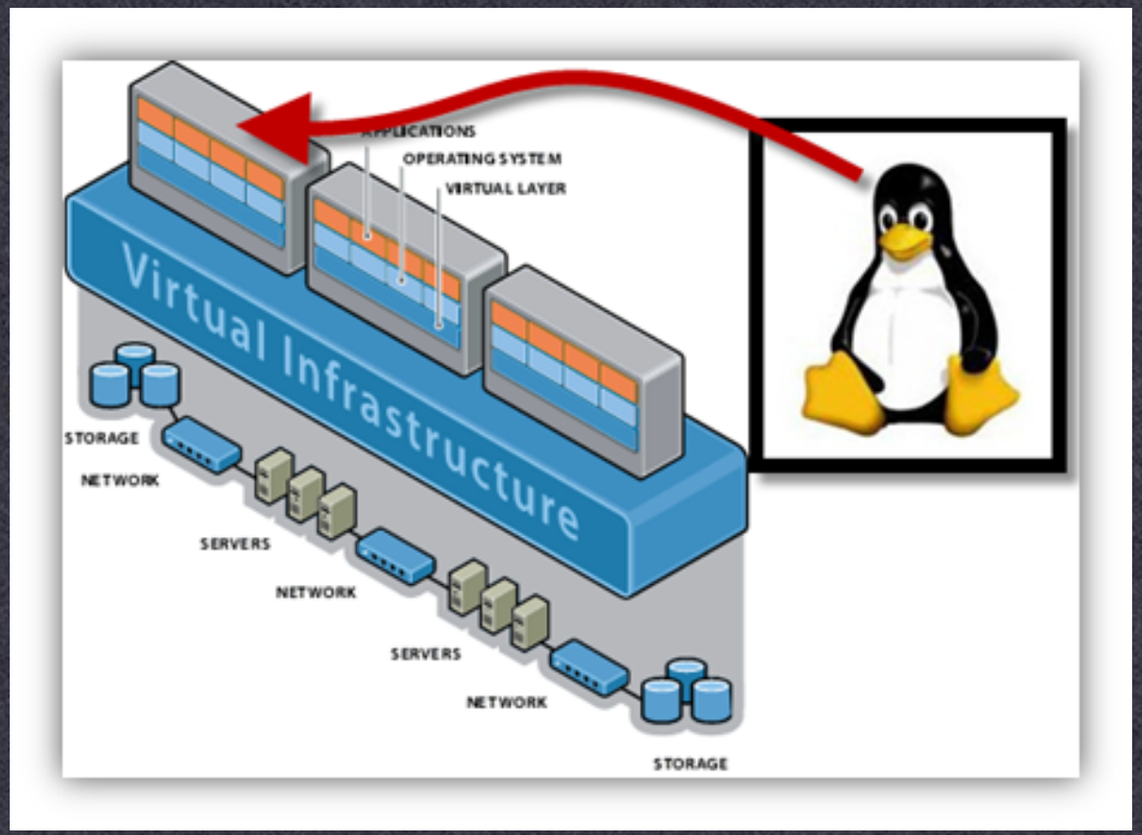
Cache Timing Attacks

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



AES implementation

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

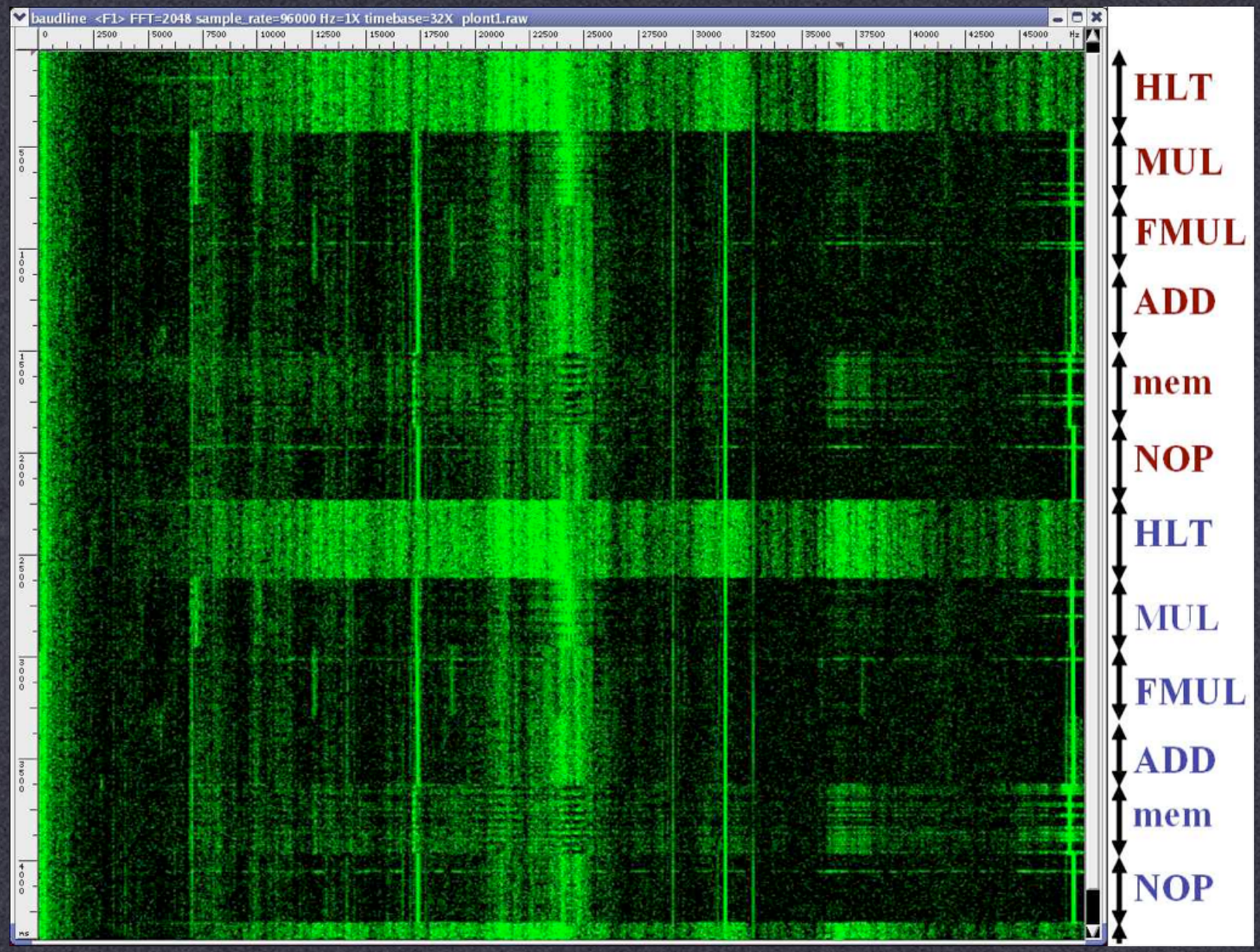


Hypervisor attacks

- **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

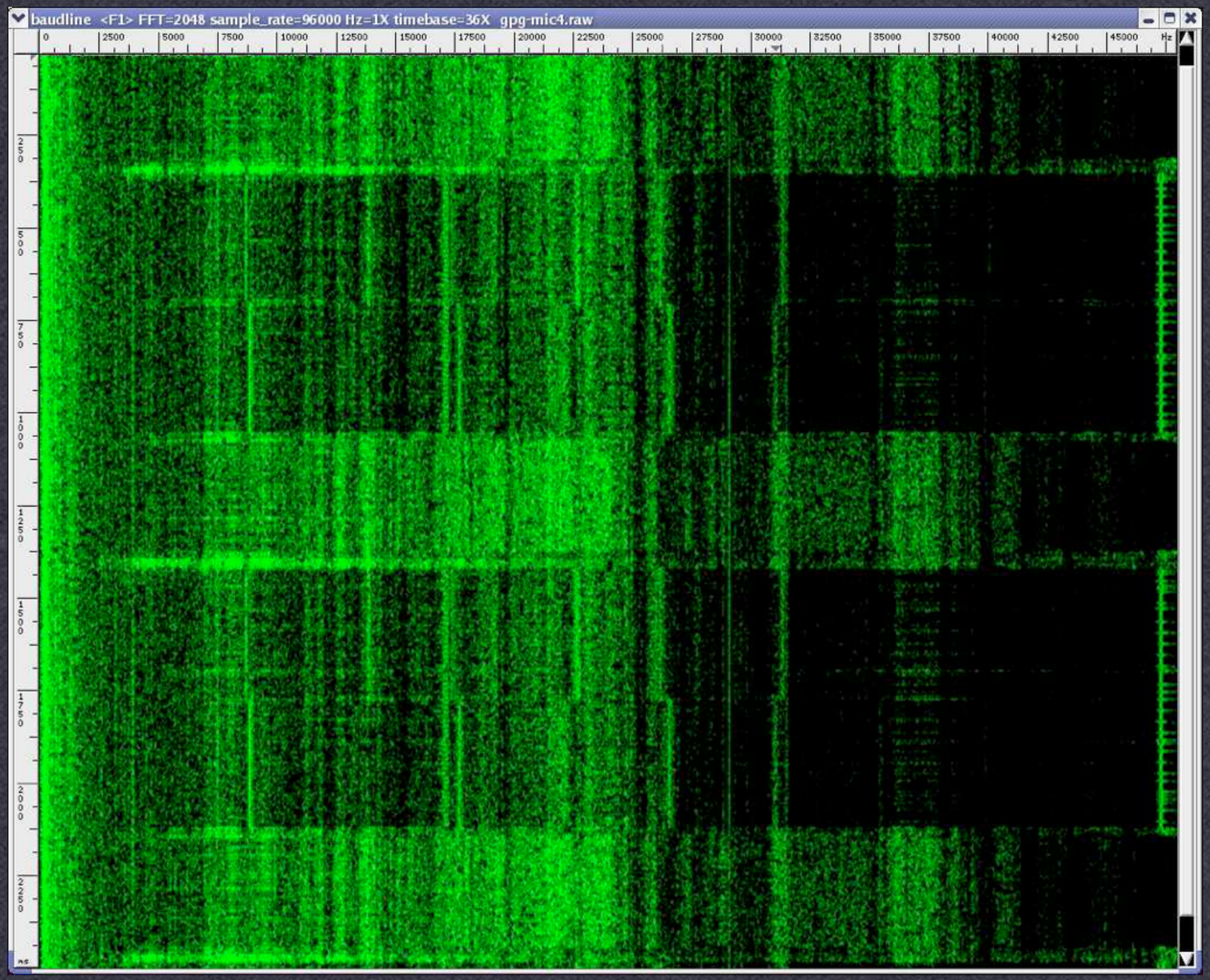
Acoustic Side Channels



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

Acoustic Side-Channels

GPG RSA-CRT signature #1:



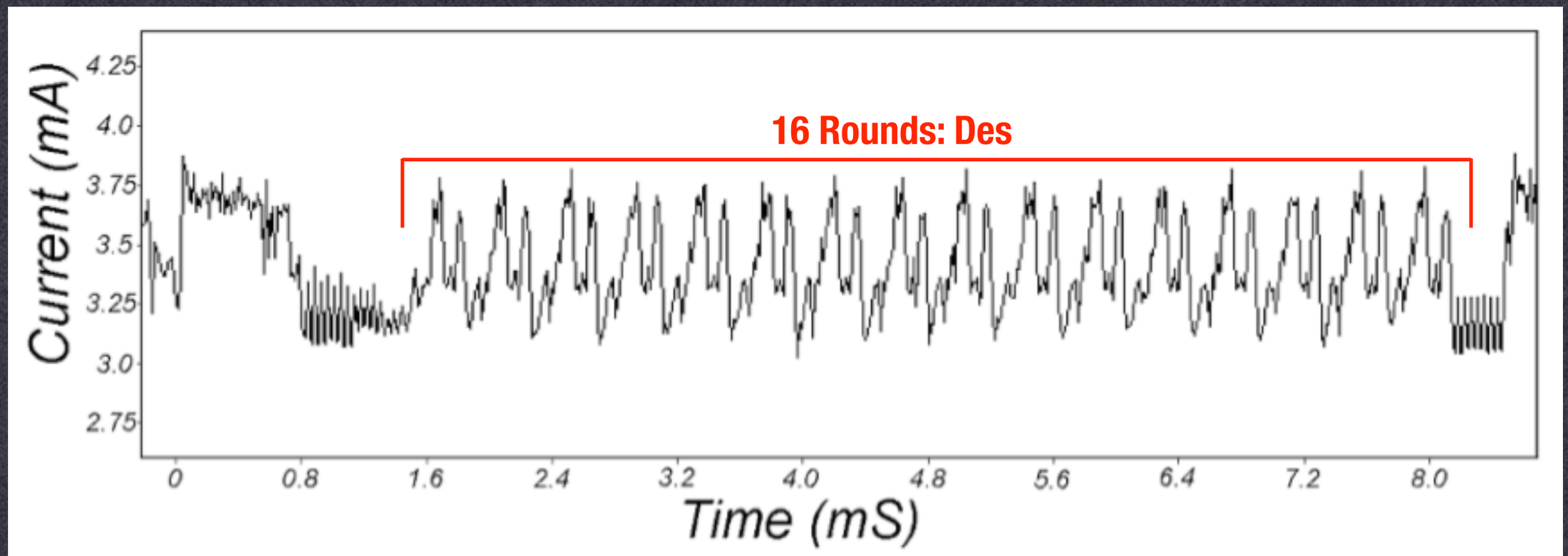
mod q

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**