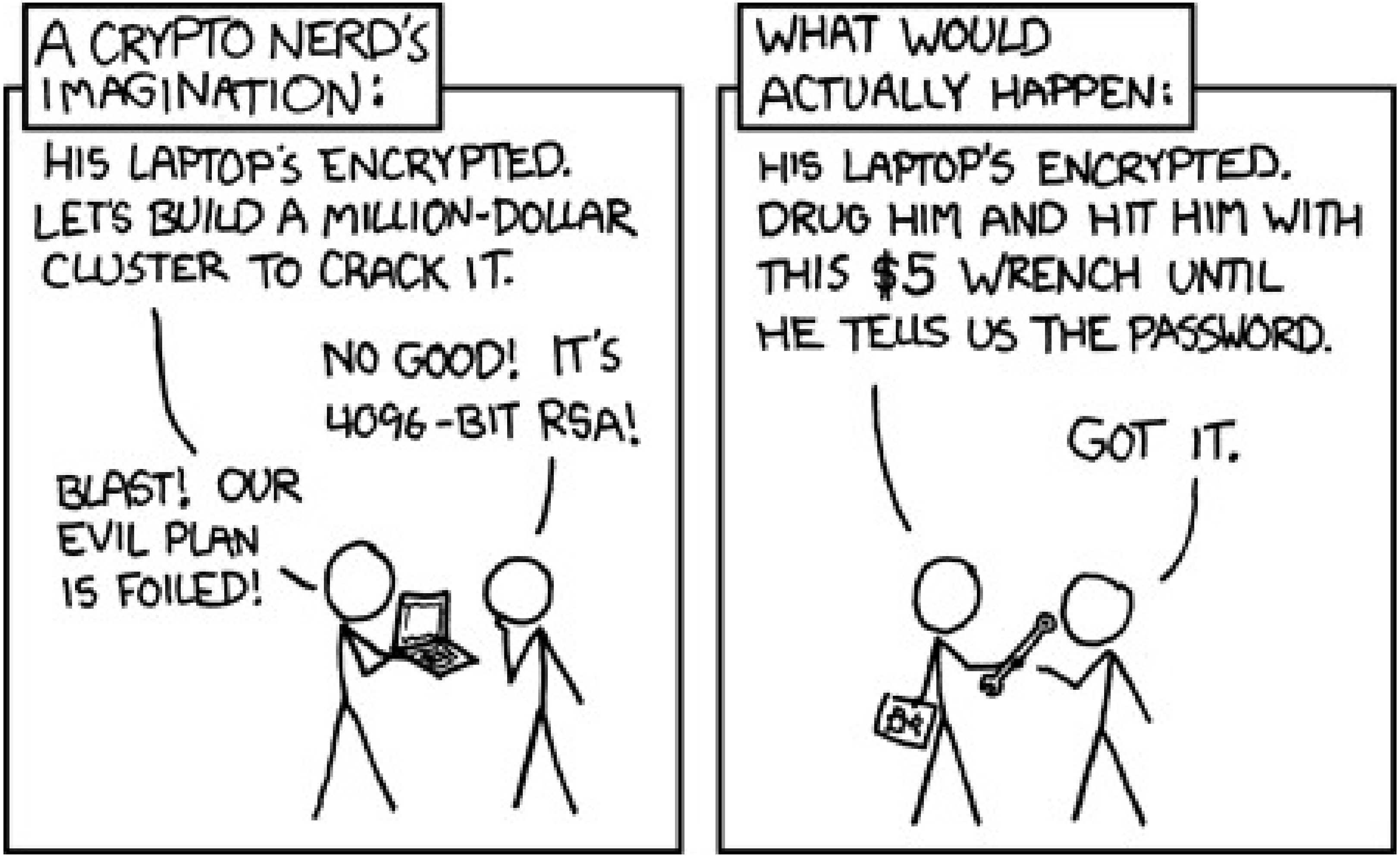
Side-Channel Cryptanalysis

Joseph Bonneau

Security Group

jcb82@cl.cam.ac.uk

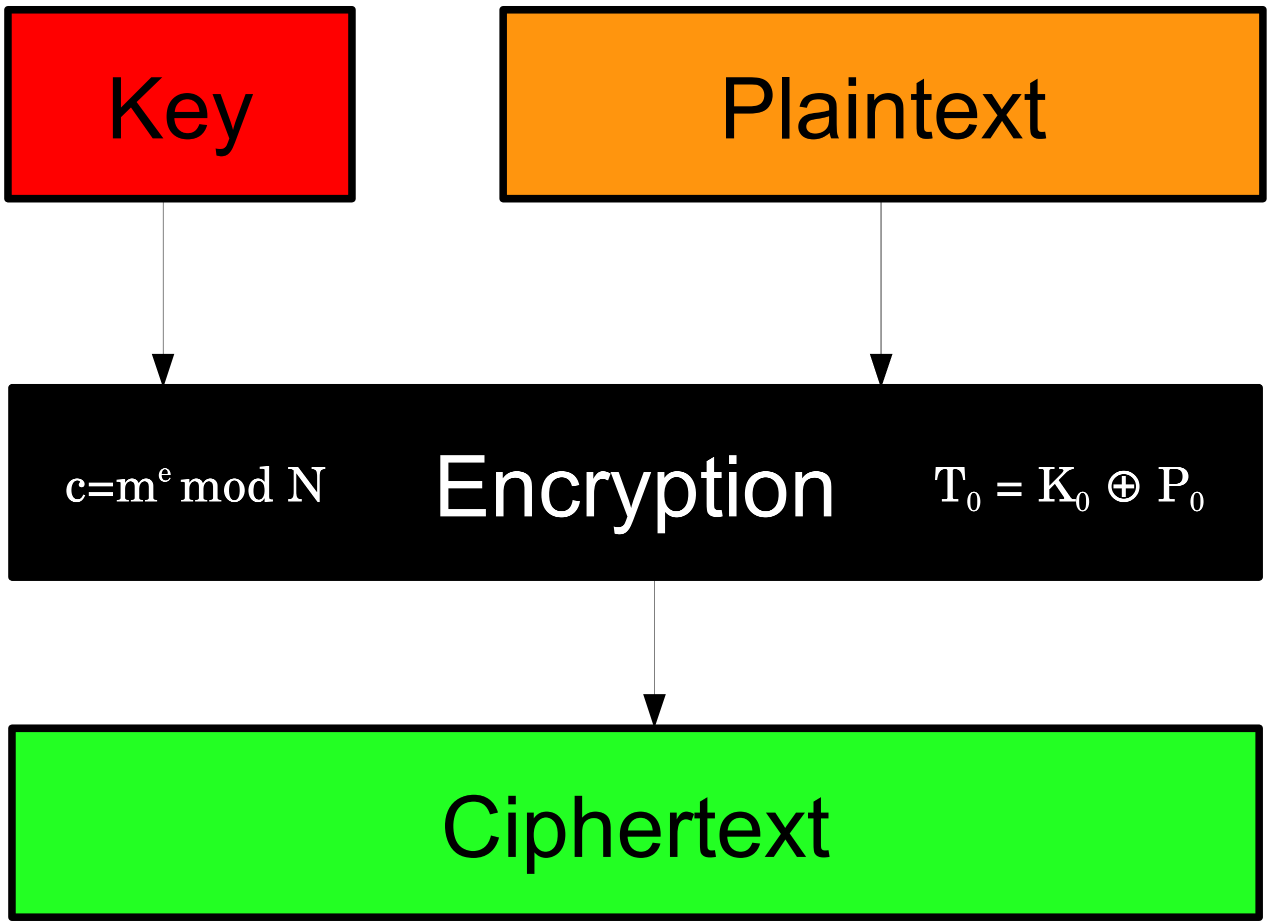
Rule 0: Attackers will **always** cheat



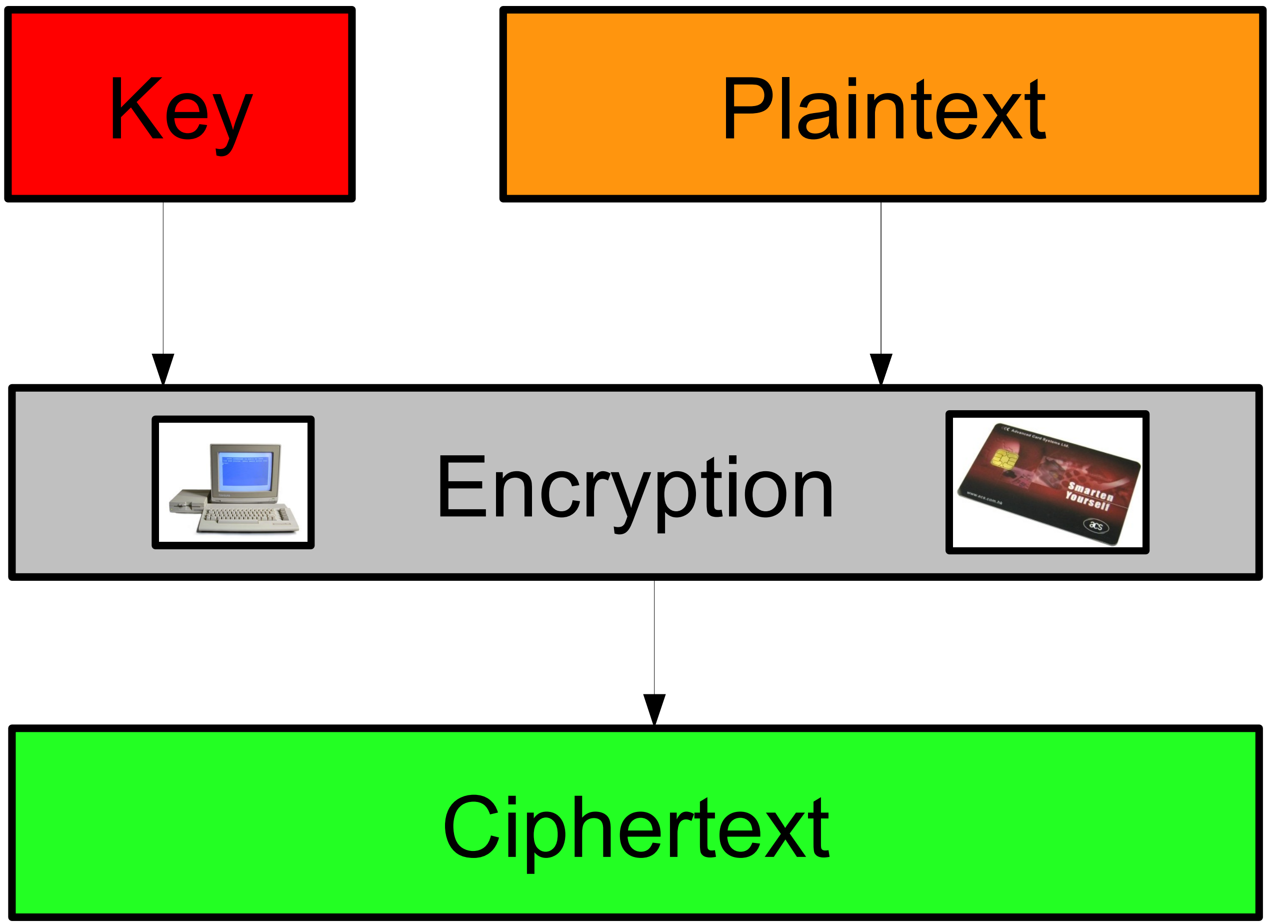
xkcd #538

What is side channel cryptanalysis?

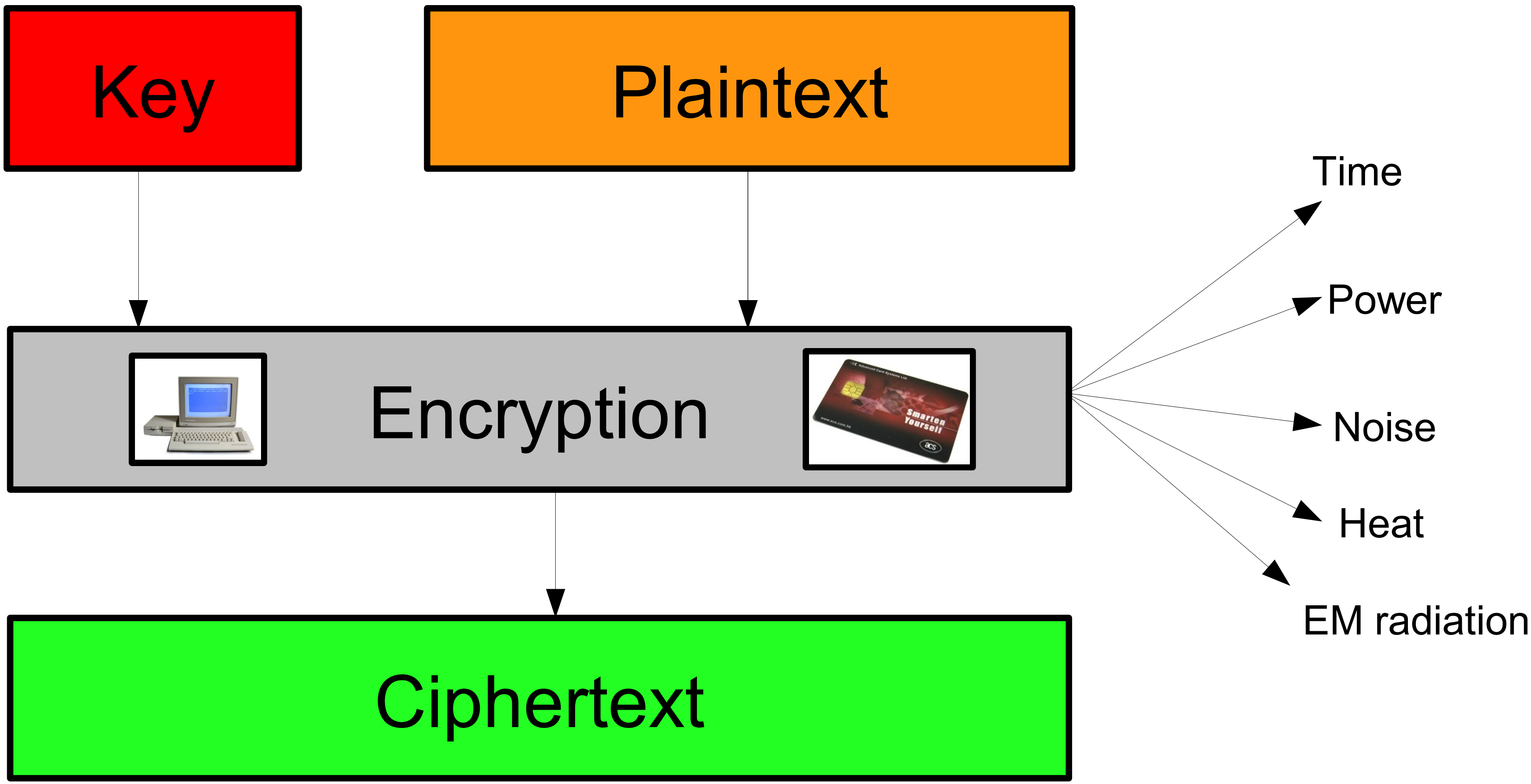
Side Channels: whatever the designers ignored



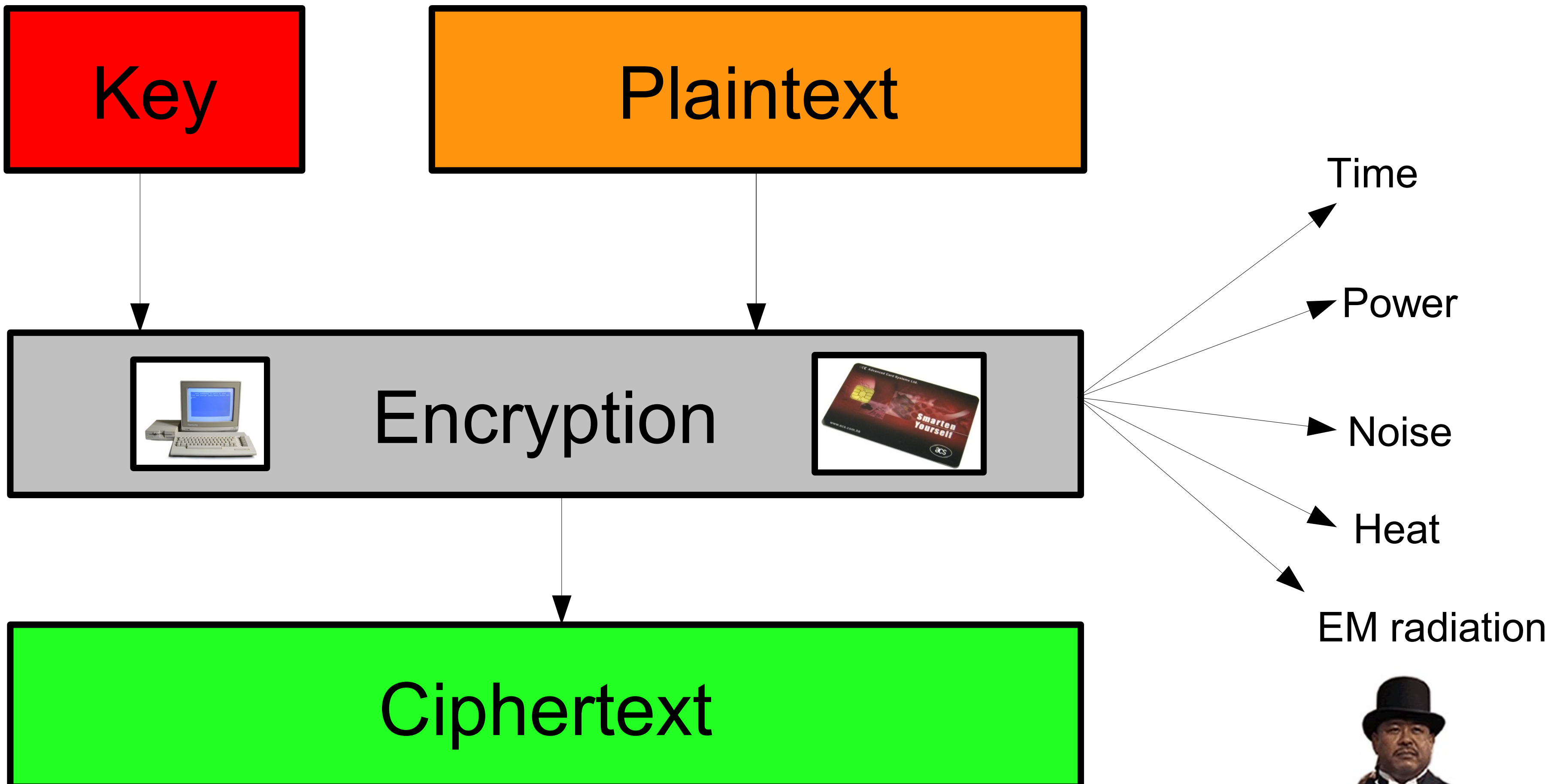
Side Channels: whatever the designers ignored



Side Channels: whatever the designers ignored



Side Channels: whatever the designers ignored

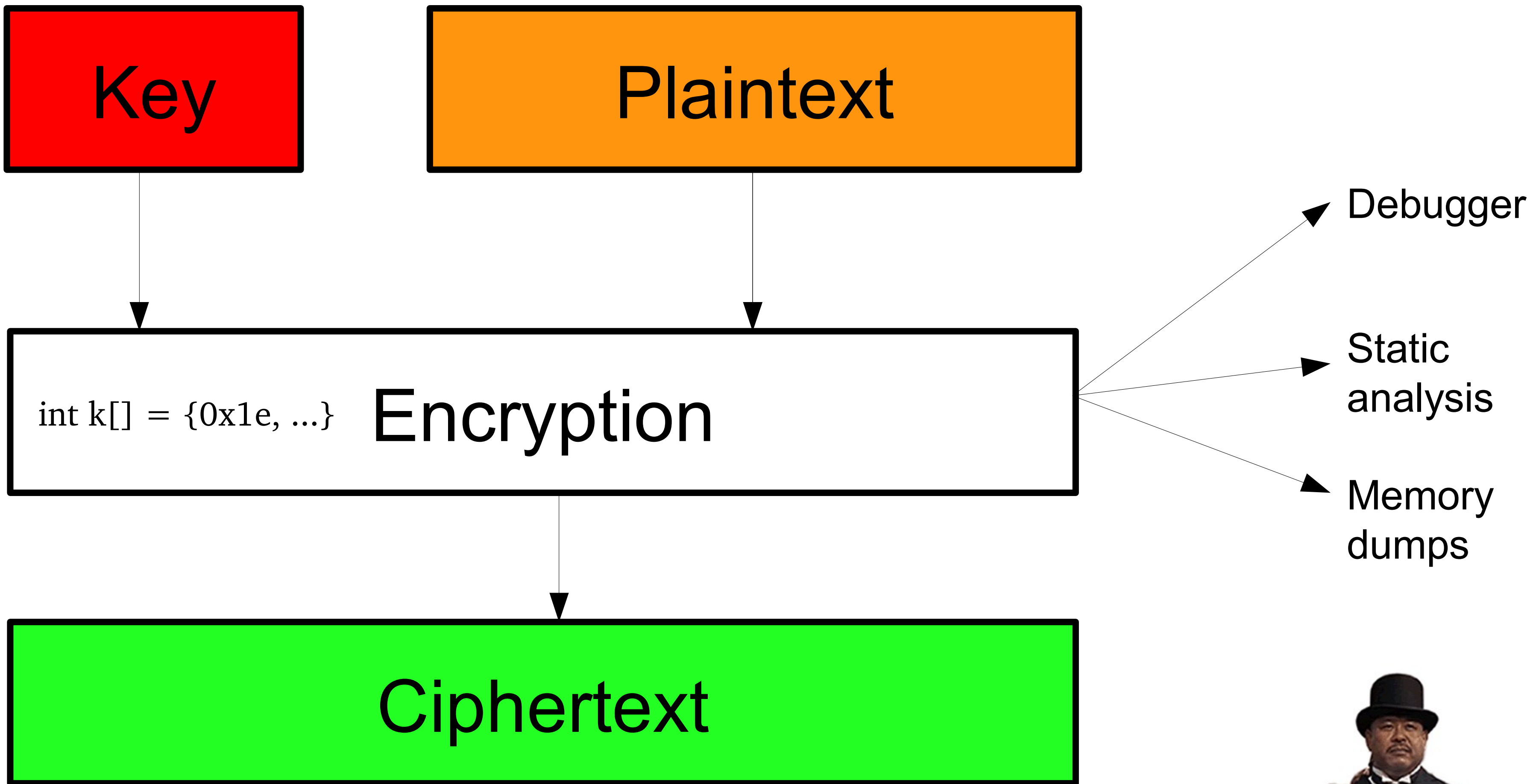


Definition

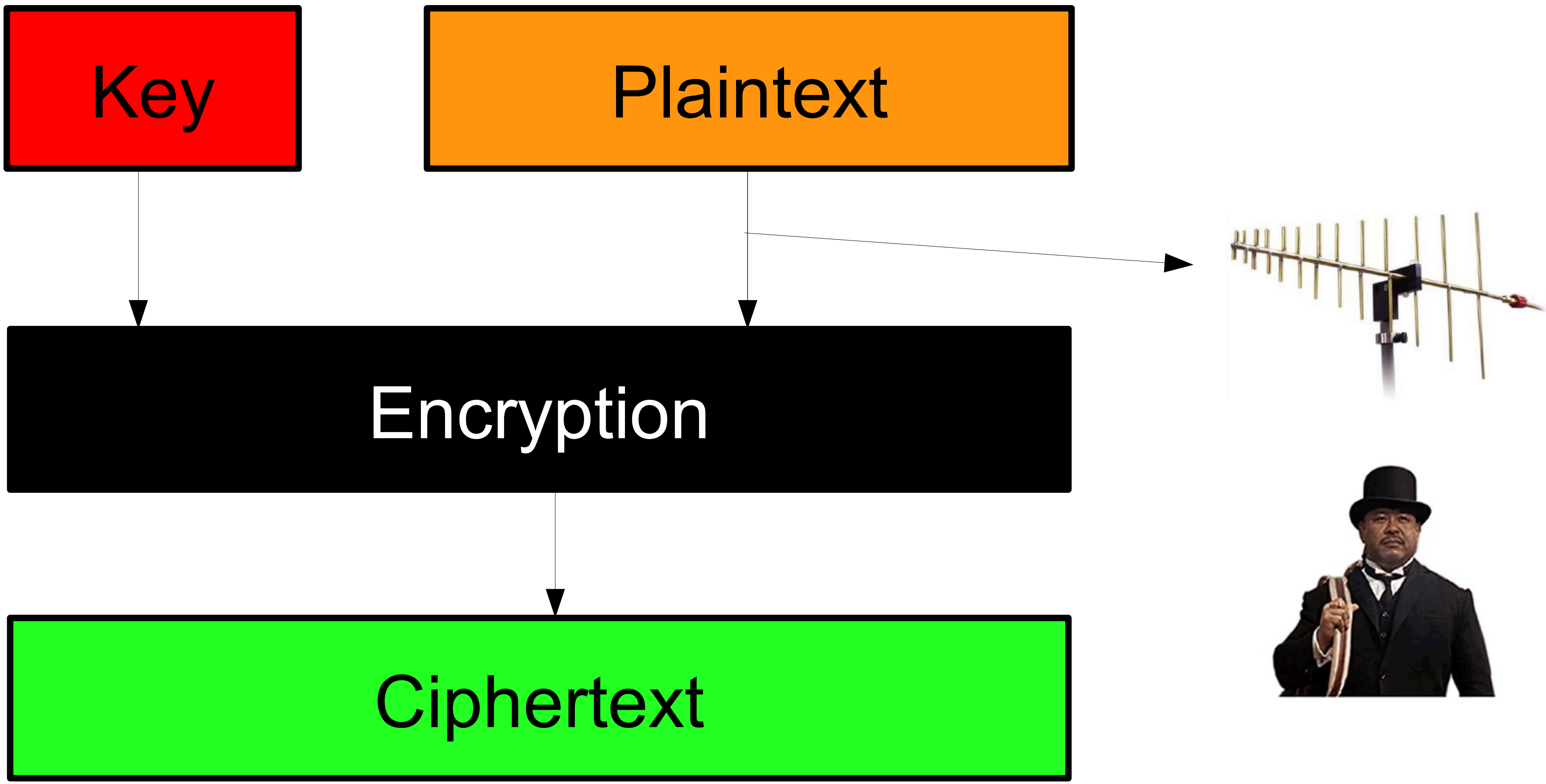
Side-channel cryptanalysis is any attack on a cryptosystem requiring information emitted as a byproduct of the physical implementation.

Related attacks

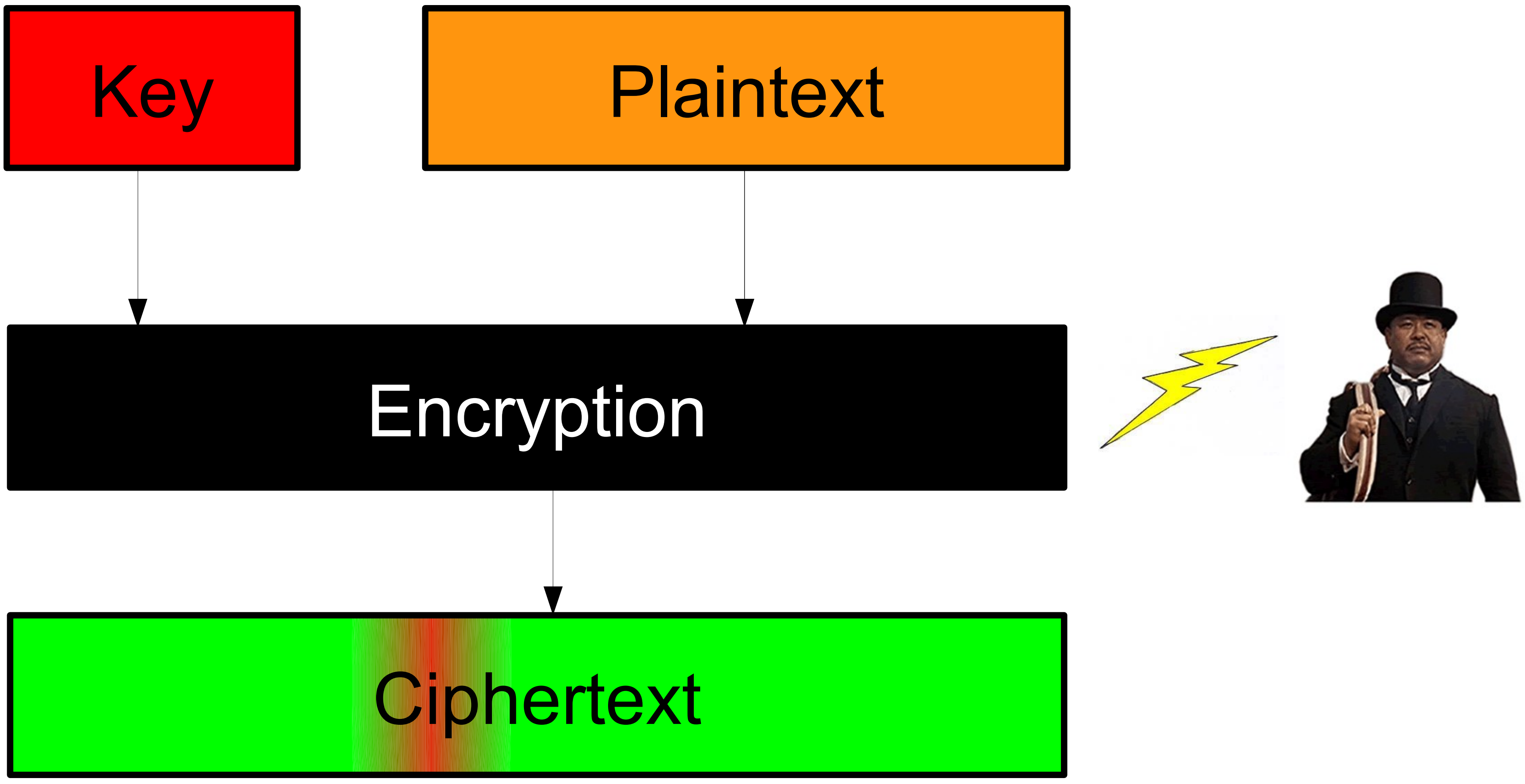
White box cryptanalysis: nothing is hidden



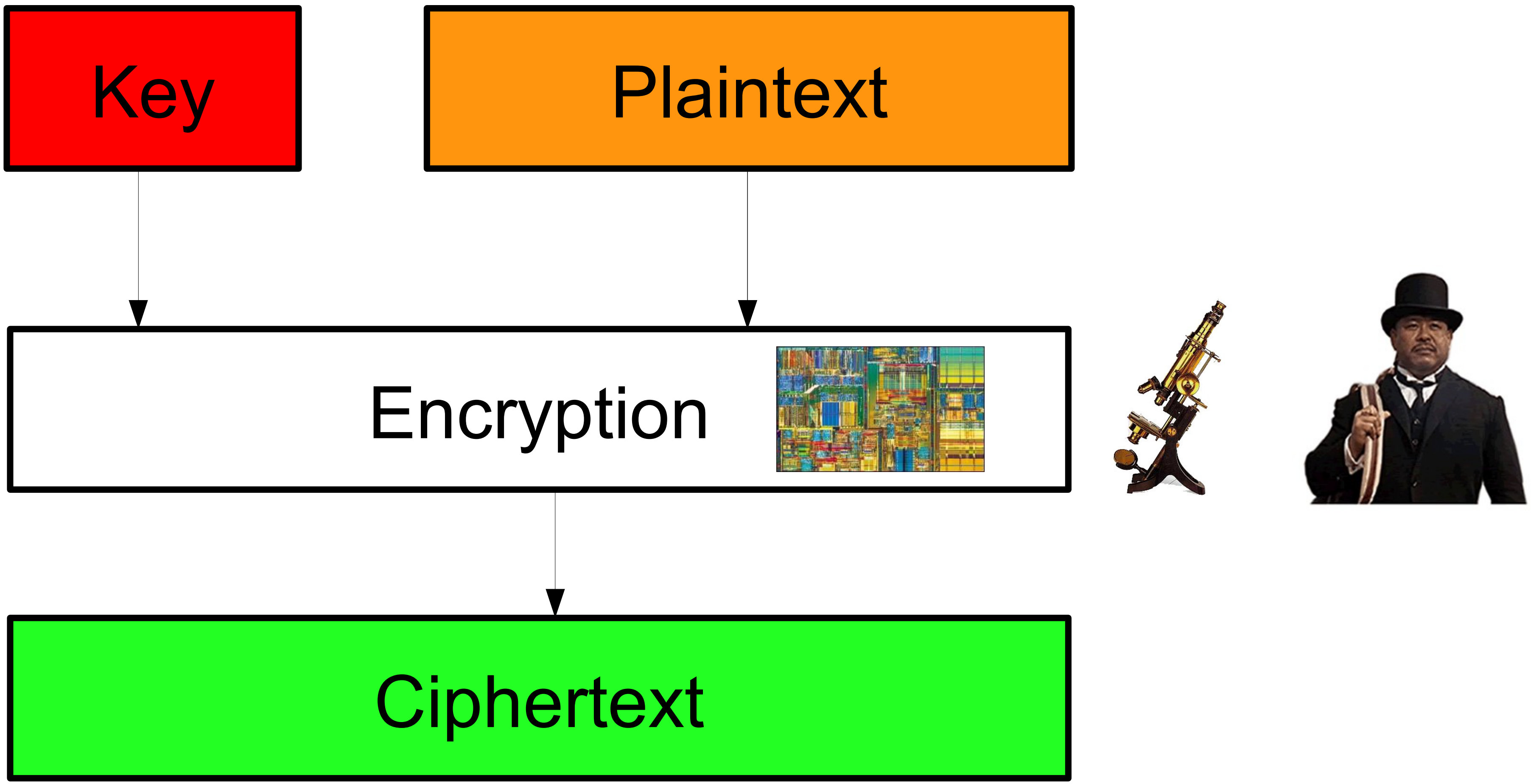
TEMPEST: EM signals containing full secrets



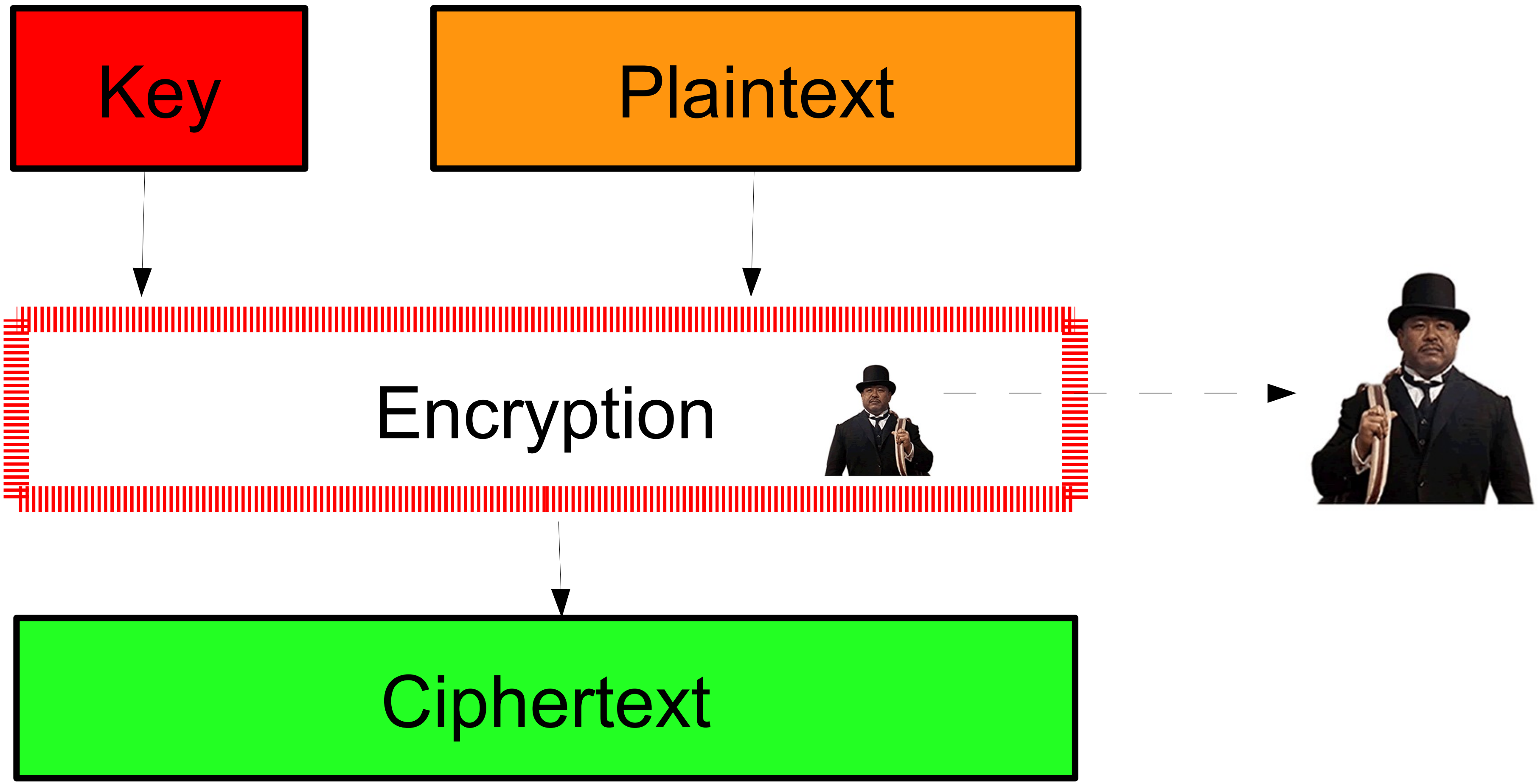
Fault injection: inducing a telling error



Hardware attacks: breaking the box open



Covert channels: attack code running from within



The grandmother of all timing attacks

Insecure password checking routine

```
int check_password(char * test, char * correct){
      return (strcmp(test, correct) == 0);
}
```

Insecure password checking routine

```
int check_password(char * test, char * correct){
      return (strcmp(test, correct) == 0);
}

int strcmp(char *s1, char *s2){
      while (*s1 != '\0' && *s1 == *s2) {
            s1++;
            s2++;
      }

      return ((s1 < s2) ? -1 : (s1 > s2));
 }
```

Insecure password checking routine

```
int check_password(char * test, char * correct){
        return (strcmp(test, correct) == 0);
}

int strcm
        whi
                s1++;
                s2++;
        }

        return ((s1 < s2) ? -1 : (s1 > s2));
 }
```

≈ n · |A| queries

MAC timing attack against Xbox 360

```
C:\Documents and Settings\Administrator\Desktop\360\DGTool>DGTool.exe 1 Infectus
_5759#4_1888_build2.bin

Pairing Data 0x6DF3B8 01
Turn on your Xbox, press any key when the RRoD starts
H[0 00XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX] M 17821 A 17821 D 0 : 0 NEXT
H[0 01XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX] M 17817 A 17819 D -3 : 0 NEXT
H[0 02XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX] M 17822 A 17820 D 3 : 0 NEXT
...
H[0 1AXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX] M 17822 A 17819 D 3 : 0 NEXT
H[0 1BXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX] M 17831 A 17819 D 12 : 11 RPT
H[0 1BXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX] M 17830 A 17819 D 11 : 0 HIT!
H[1 1B00XXXXXXXXXXXXXXXXXXXXXXXXXXXX] M 17830 A 17830 D 0 : 0 NEXT
H[1 1B01XXXXXXXXXXXXXXXXXXXXXXXXXXXX] M 17829 A 17829 D -1 : 0 NEXT
...
H[1 1BFDXXXXXXXXXXXXXXXXXXXXXXXXXXXX] M 17844 A 17830 D 14 : 8 RPT
H[1 1BFDXXXXXXXXXXXXXXXXXXXXXXXXXXXX] M 17839 A 17830 D 9 : 0 HIT!
H[2 1BFD00XXXXXXXXXXXXXXXXXXXXXXXXXX] M 17839 A 17838 D 1 : 0 NEXT
H[2 1BFD01XXXXXXXXXXXXXXXXXXXXXXXXXX] M 17844 A 17841 D 4 : 0 NEXT
...
H[2 1BFDF0XXXXXXXXXXXXXXXXXXXXXXXXXX] M 17856 A 17841 D 15 : 11 RPT
H[2 1BFDF0XXXXXXXXXXXXXXXXXXXXXXXXXX] M 17851 A 17841 D 10 : 0 HIT!
...
H[15 1BFDF0625C214F67CD94DCA3FC47CA55] M 18014 A 17988 D 16 : 0 HIT!

Correct hash: 1BFDF0625C214F67CD94DCA3FC47CA55
Result: BOOT
```

MAC timing attack against Xbox 360

```
C:\Documents and Settings\Administrator\Desktop\360\DGTool>DGTool.exe 1 Infectus
_5759#4_1888_build2.bin

Pairing Data 0x6DF3B8 01
Turn on your Xbox, press any key when the RRoD starts
H[0 00XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX] M 17821 A 17821 D 0 : 0 NEXT
H[0 01XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX] M 17817 A 17819 D -3 : 0 NEXT
H[0 02XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX] M 17822 A 17820 D 3 : 0 NEXT
...
H[0 1AXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX] M 17822 A 17819 D 3 : 0 NEXT
H[0 1BXXXXXXXX
H[0 1BXXXXXXXX
H[1 1B00XXXXXX
H[1 1B01XXXXXXXXXXXXXXXXXXXXXXXXXXXX] M 17829 A 17829 D -1 : 0 NEXT
...
H[1 1BFDXXXXXXXXXXXXXXXXXXXXXXXXXXXX] M 17844 A 17830 D 14 : 8 RPT
H[1 1BFDXXXXXXXXXXXXXXXXXXXXXXXXXXXX] M 17839 A 17830 D 9 : 0 HIT!
H[2 1BFD00XXXXXXXXXXXXXXXXXXXXXXXXXX] M 17839 A 17838 D 1 : 0 NEXT
H[2 1BFD01XXXXXXXXXXXXXXXXXXXXXXXXXX] M 17844 A 17841 D 4 : 0 NEXT
...
H[2 1BFDF0XXXXXXXXXXXXXXXXXXXXXXXXXX] M 17856 A 17841 D 15 : 11 RPT
H[2 1BFDF0XXXXXXXXXXXXXXXXXXXXXXXXXX] M 17851 A 17841 D 10 : 0 HIT!
...
H[15 1BFDF0625C214F67CD94DCA3FC47CA55] M 18014 A 17988 D 16 : 0 HIT!

Correct hash: 1BFDF0625C214F67CD94DCA3FC47CA55
Result: BOOT
```

≈ 2,048 queries

Attacks against RSA

RSA: The original public key algorithm

RSA

SECURITY™

RSA: The original public key algorithm

Private Key:	p, q	(random primes)
	$d \equiv e^{-1} \pmod{\varphi(N)}$	(exponent)
Public Key:	$N = p \cdot q$	(modulus)
	e	(exponent)
Encryption:	$c = m^e \pmod{N}$	
Verification:	$m = c^d \pmod{N}$	
Signing:	$s = m^d \pmod{N}$	
Verification:	$m = s^e \pmod{N}$	

RSA is implemented via square-and-multiply

$$b^{65553} \pmod{N}$$
$$\equiv b^{0x10011} \pmod{N}$$
$$\equiv b^{(10000000000010001)b} \pmod{N}$$
$$\equiv b^{65536} \cdot b^{16} \cdot b^{1} \pmod{N}$$

n digit exponentiation requires **log *n*** squares, up to **log *n*** multiplications

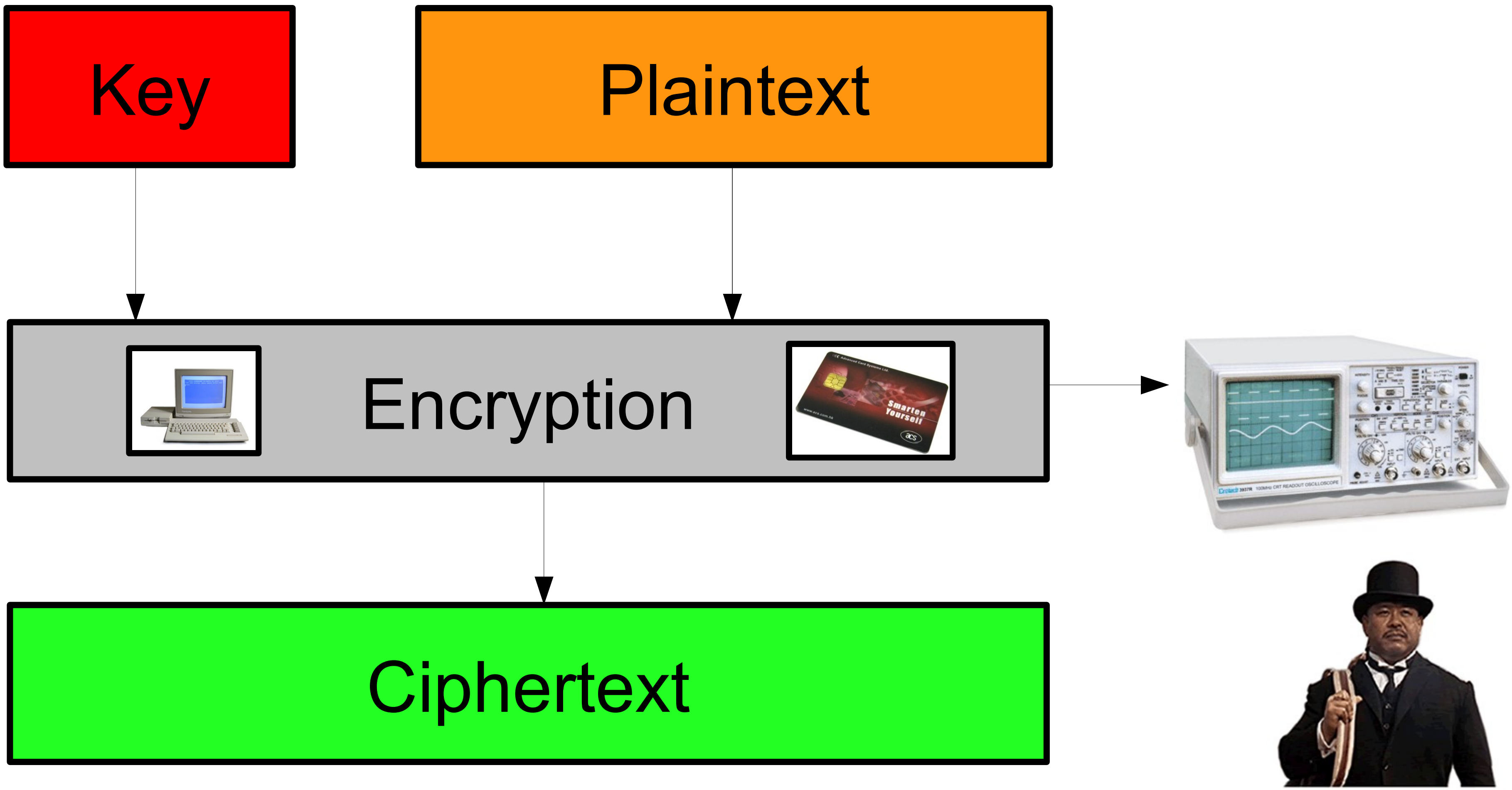
RSA is implemented via square-and-multiply

```
def power(b, e, N):
    result = 1

    for i in range(len(e)- 1, -1):
        result = square(result, N)
        if bit_set(e, i):
            result = mult(result, b, N)

    return result
```

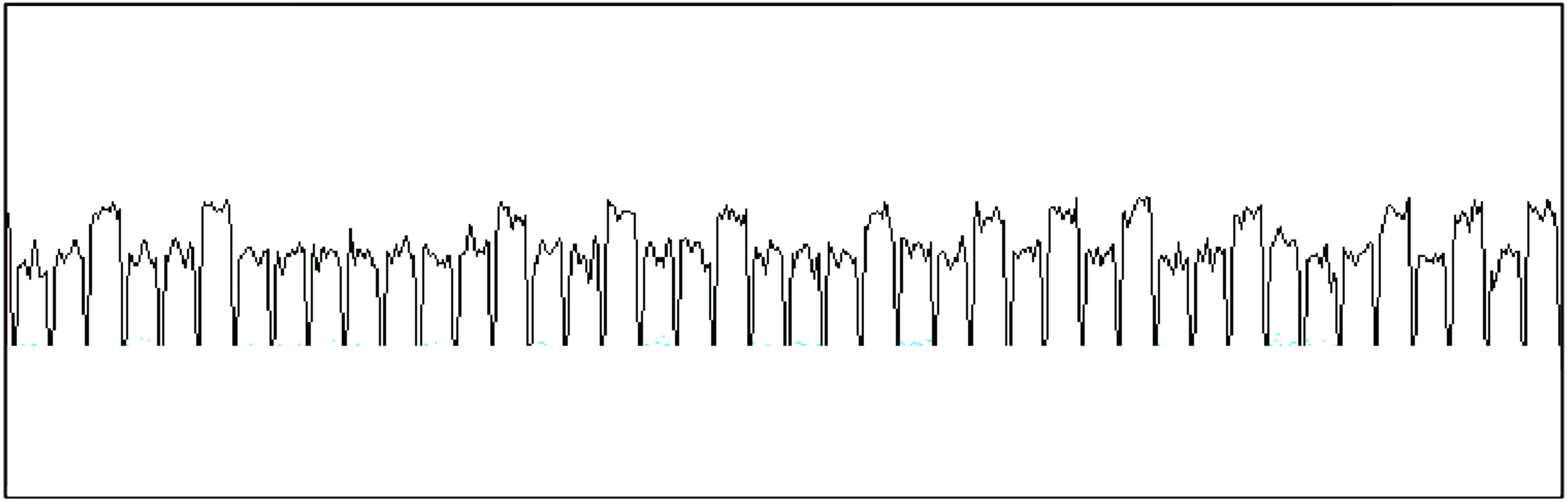
Simple power analysis of RSA

(Paul Kocher et. al 1999)

Simple power analysis setup



What does power consumption reveal?

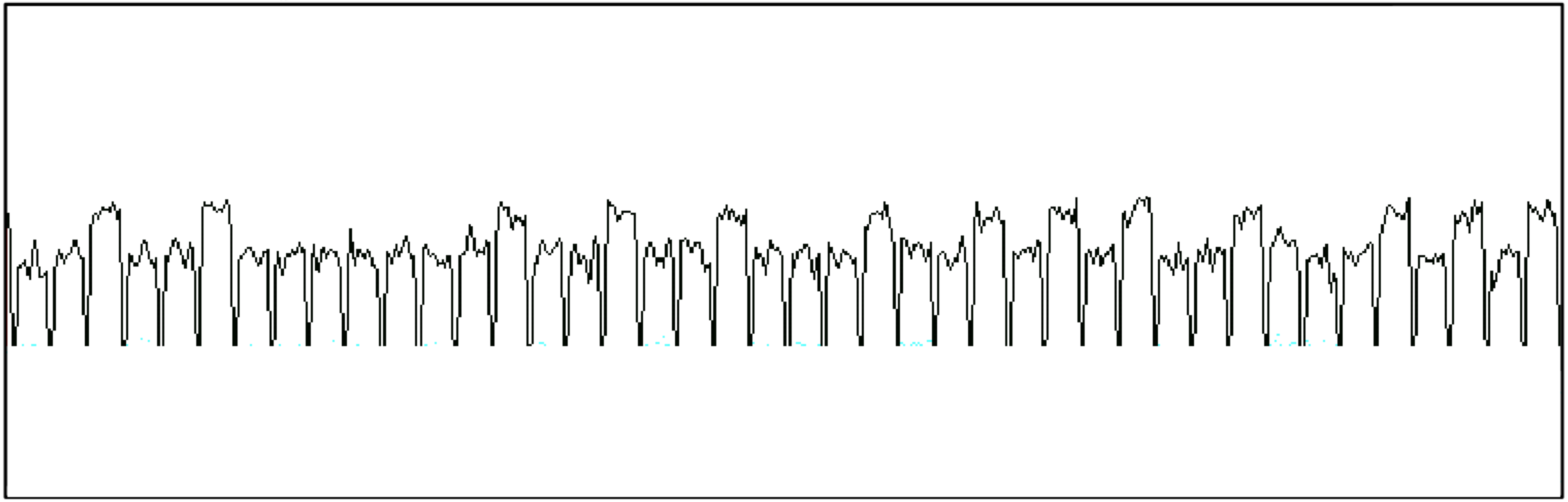


Trace courtesy of Cryptography Research, Inc.

Each set exponent bit inserts a multiplication

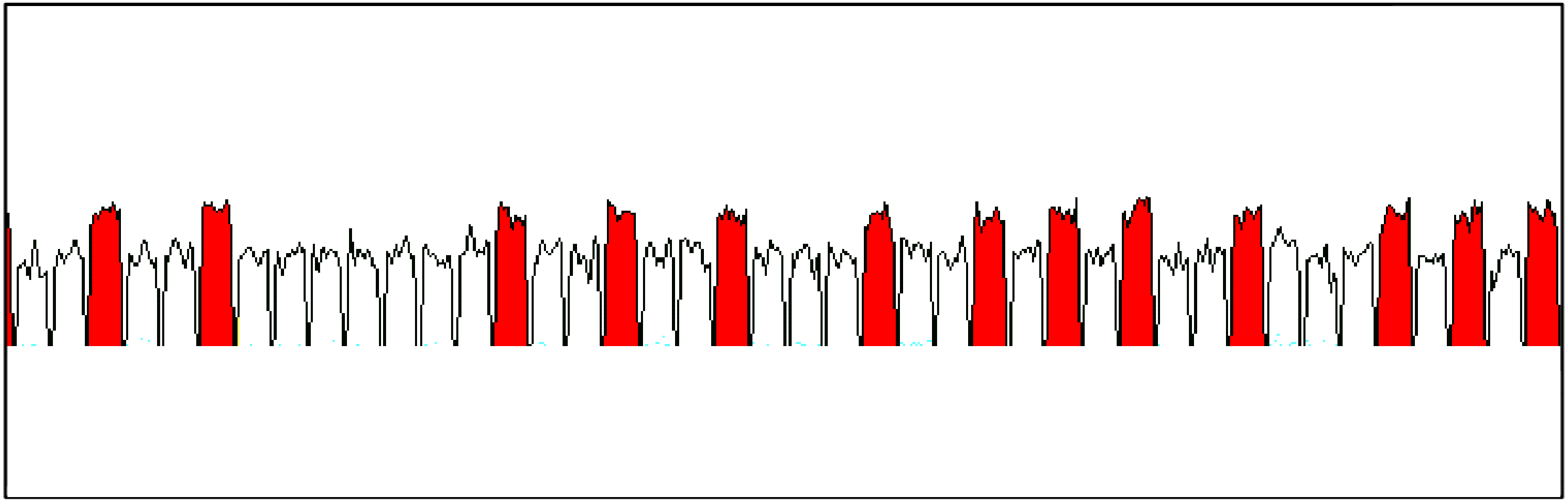
```
def power(b, e, N):
    result = 1

    for i in range(len(e)- 1, -1):
        result = square(result, N)
        if bit_set(e, i):
            result = mult(result, b, N)

    return result
```

Multiplies can often be visually detected



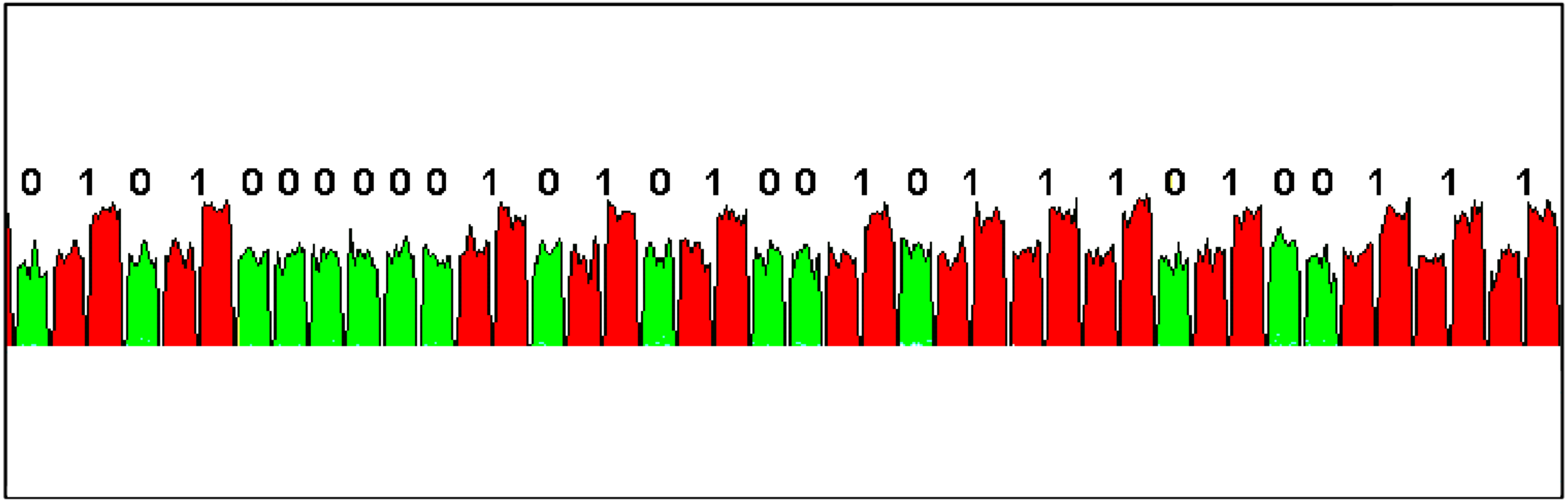
Trace courtesy of Cryptography Research, Inc.

Multiplies can often be visually detected



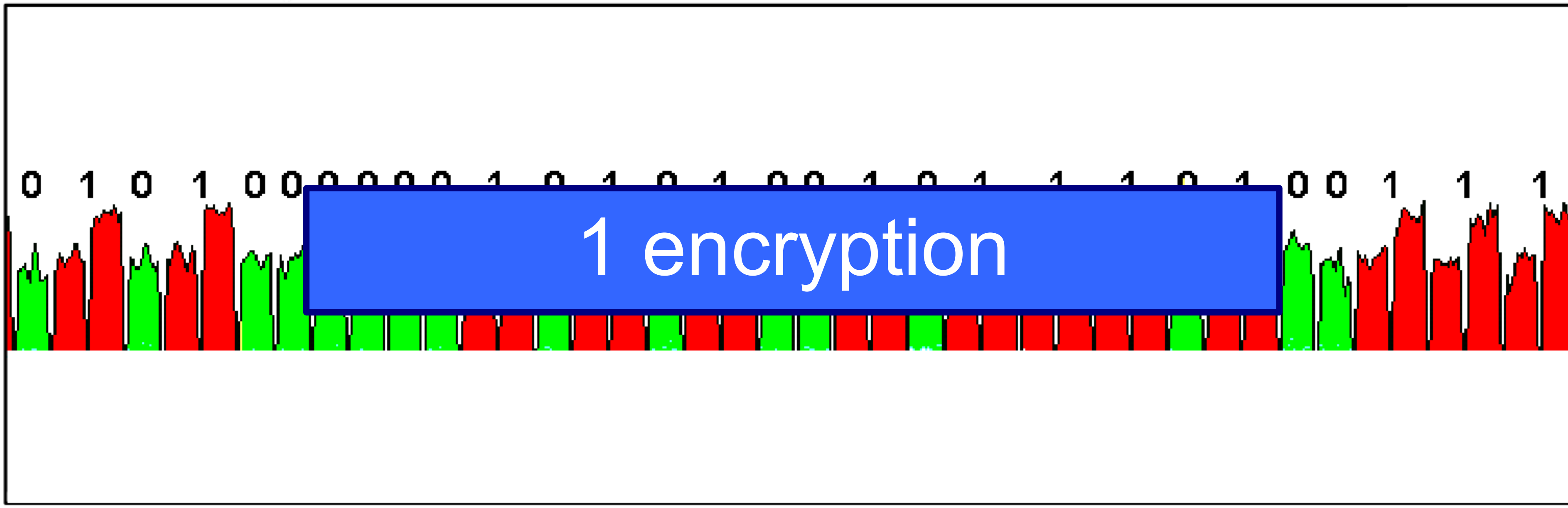
Trace courtesy of Cryptography Research, Inc.

Secret exponent can be easily read out



Trace courtesy of Cryptography Research, Inc.

Secret exponent can be easily read out



Trace courtesy of Cryptography Research, Inc.

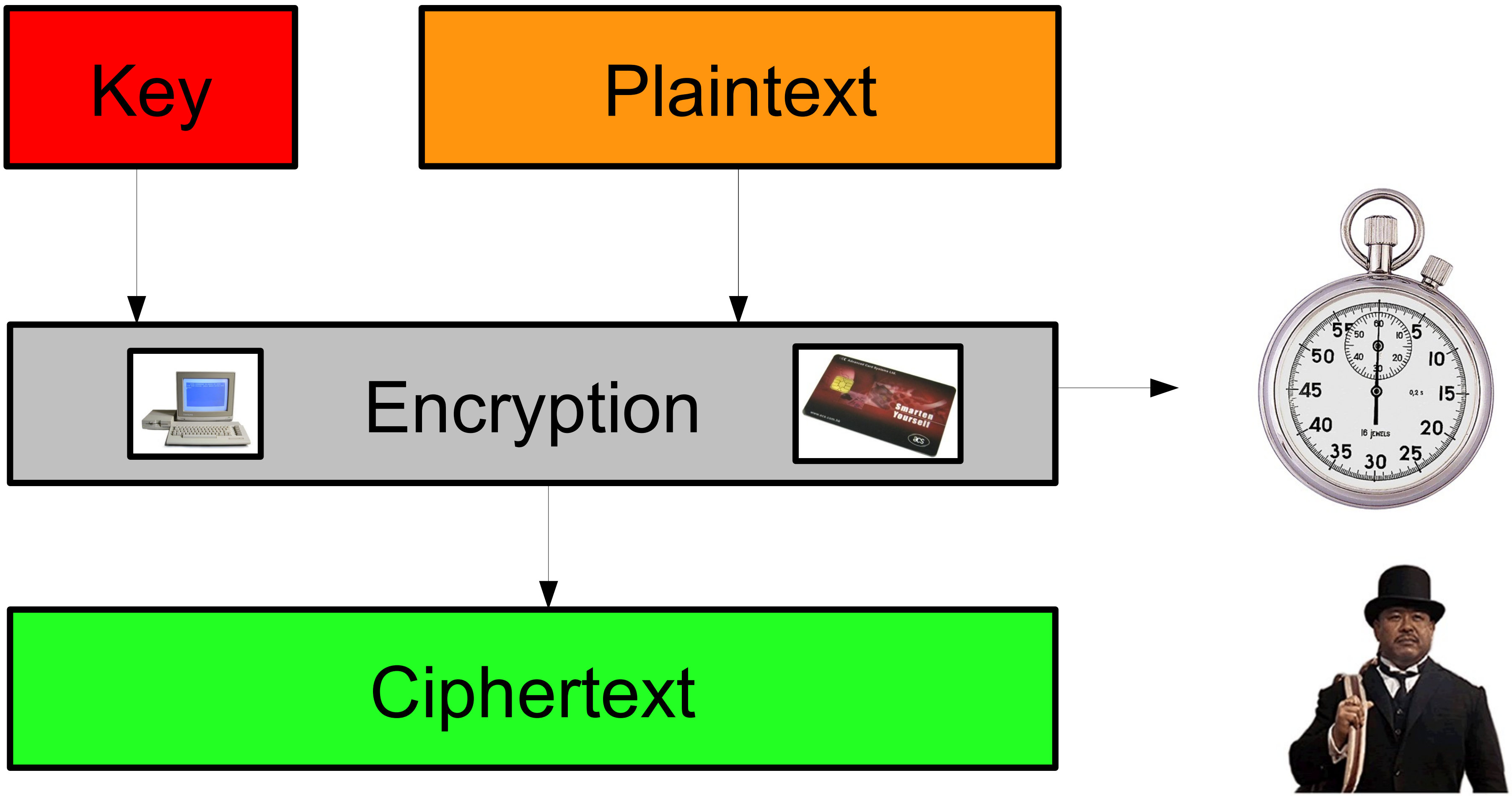
Algorithmic patch: square and **always** multiply

```
def power(b, e, N):
    result = 1
    b = mult(b, r, N)
    for i in range(len(e)- 1, -1):
        result = square(result, N)
        if bit_set(e, i):
            result = mult(result, b, N)
        else:
            result = mult(result, r, N)
    return mult(result, r_inverse, N)
```

Timing attack against RSA

(Paul Kocher 1996)

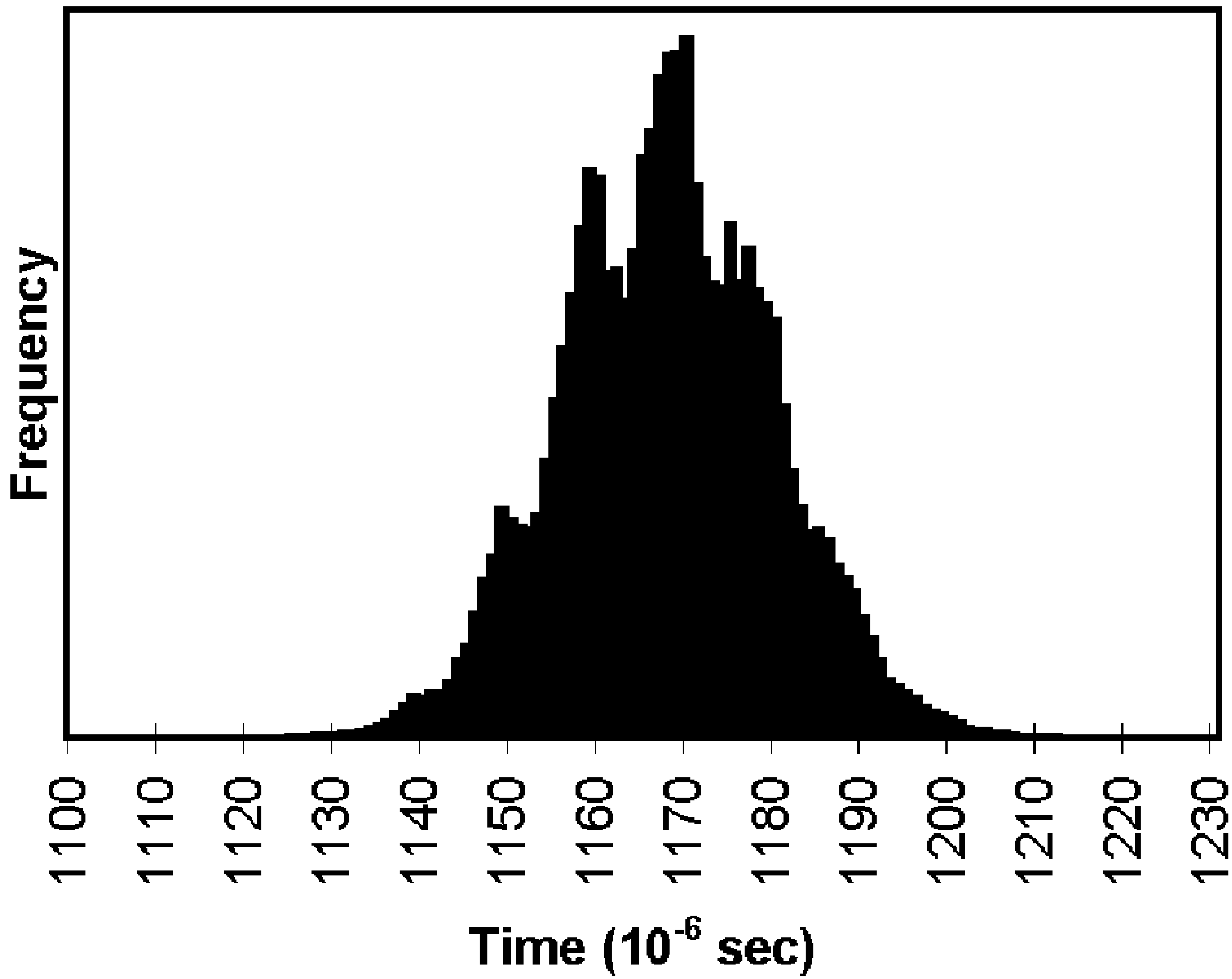
Timing attack setup



Timing leaks Hamming weight of exponent

```
def power(b, e, N):
    result = 1

    for i in range(len(e)- 1, -1):
        result = square(result, N)
        if bit_set(e, i):
            result = mult(result, b, N)

    return result
```

Timing of individual multiplies varies significantly



Kocher 1996

Need a model relating multiplication inputs to time

Example:

$2345 \cdot 6789 \pmod{9997}$

Multiply: $2345 \cdot 6826 = 16006970$

Reduce:

16006970	$- 9997 \cdot 1 \cdot 1000$
$=6009970$	$- 9997 \cdot 6 \cdot 100$
$=11770$	$- 9997 \cdot 0 \cdot 10$
$=11770$	$- 9997 \cdot 1 \cdot 1$
$=1773$	

Attack exponent one bit at a time

```
def power(b, e, N):
    result = 1

    for i in range(len(e)- 1, -1):
        result = square(result, N)
        if bit_set(e, i):
            result = mult(result, b, N)

    return result
```

Attack exponent one bit at a time

T = observed timing of entire algorithm

M = model for time of one multiplication

Bit n-1:   always 1

Attack exponent one bit at a time

T = observed timing of entire algorithm

M = model for time of one multiplication

Bit n-2: Is $T(\texttt{power(r,e,N)}) \propto$

$M(\texttt{mult(1,r,N)})$ +

$M(\texttt{square(r,N)})$ +

$M(\texttt{mult(r}^{2}\texttt{,r,N)})$ +

$M(\texttt{square(r}^{3}\texttt{,N)})$?

Attack exponent one bit at a time

T = observed timing of entire algorithm

M = model for time of one multiplication

Bit n-2: Is T(`power(r,e,N)`) $\propto$

~~M(`mult(1,r,N)`) +~~

~~M(`square(r,N)`) +~~

M(`mult(r`^{2}`,r,N)`) +

M(`square(r`^{3}`,N)`)?

Attack exponent one bit at a time

T = observed timing of entire algorithm

M = model for time of one multiplication

Bit n-3: Is T($\texttt{power(r,e,N)}$) $\propto$

M($\texttt{mult(1,r,N)}$)·e[n-1] +

M($\texttt{square(r,N)}$) +

M($\texttt{mult(r}^{2\cdot e[n-1:n-1]}\texttt{,r,N)}$)·e[n-2] +

M($\texttt{square(r}^{e[n-1:n-2]}\texttt{,N)}$)

M($\texttt{mult(r}^{2\cdot e[n-1:n-2]}\texttt{,r,N)}$) +

M($\texttt{square(r}^{e[n-1:n-2]||1}\texttt{,N)}$)?

Attack exponent one bit at a time

T = observed timing of entire algorithm

M = model for time of one multiplication

Bit n-3: Is T(power(r,e,N)) ∝

~~M(mult(1,r,N))·e[n-1] +~~

~~M(square(r,N)) +~~

~~M(mult($r^{2\cdot e[n-1:n-1]}$,r,N))·e[n-2] +~~

~~M(square($r^{e[n-1:n-2]}$,N))~~

M(mult($r^{2\cdot e[n-1:n-2]}$,r,N)) +

M(square($r^{e[n-1:n-2]||1}$,N))?

Attack exponent one bit at a time

T = observed timing of entire algorithm

M = model for time of one multiplication

Bit n-i: Is T(`power(r,e,N)`) $\propto$
M(`mult(`$r^{2 \cdot e[n-1:n-i]}$`,r,N)`) +
M(`square(`$r^{e[n-1:n-i]||1}$`,N)`)?

Attack exponent one bit at a time

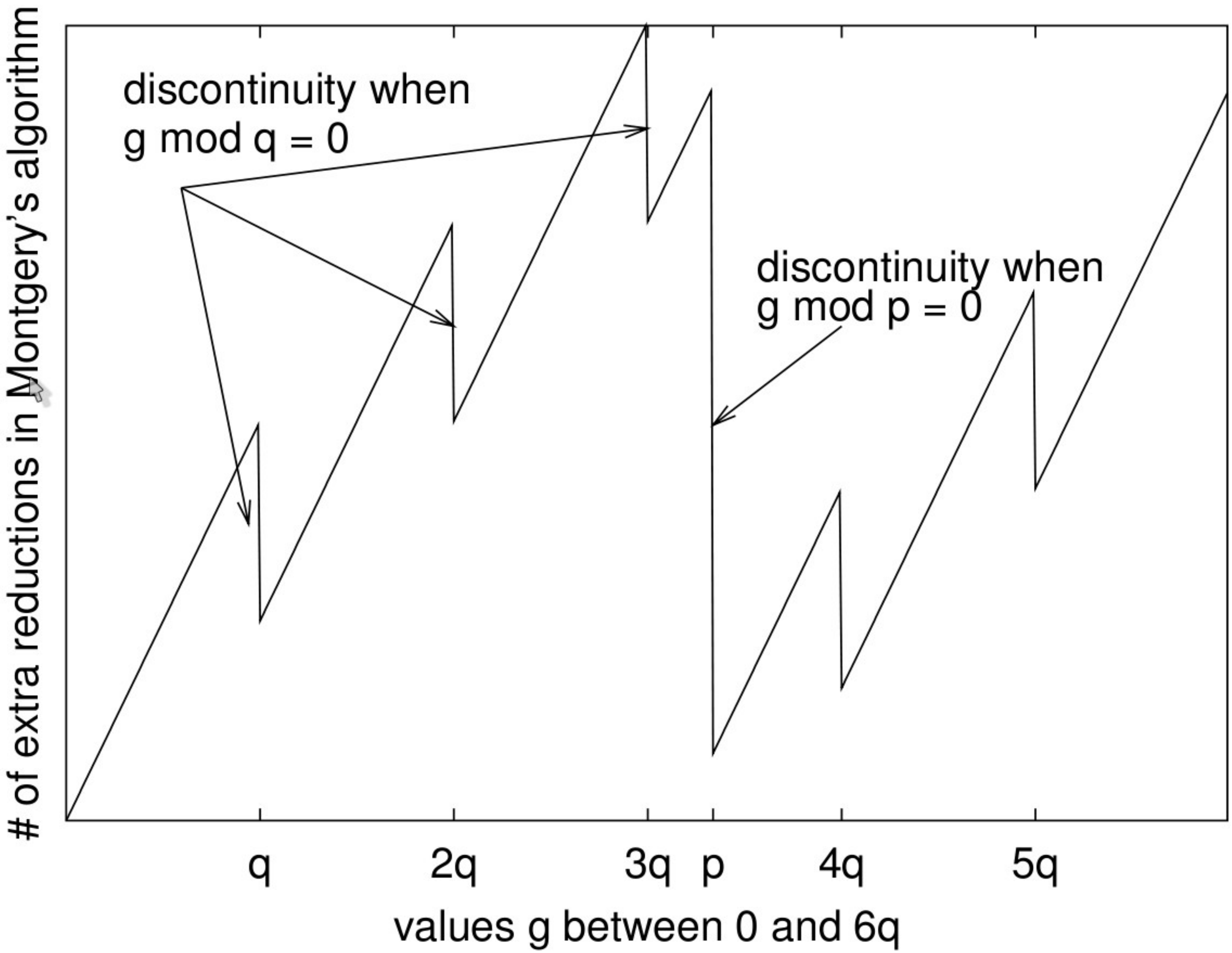
T = observed timing of entire algorithm

M = model for time of one multiplication

Bit n-i: Is T(`power(r,e,N)`) ∝

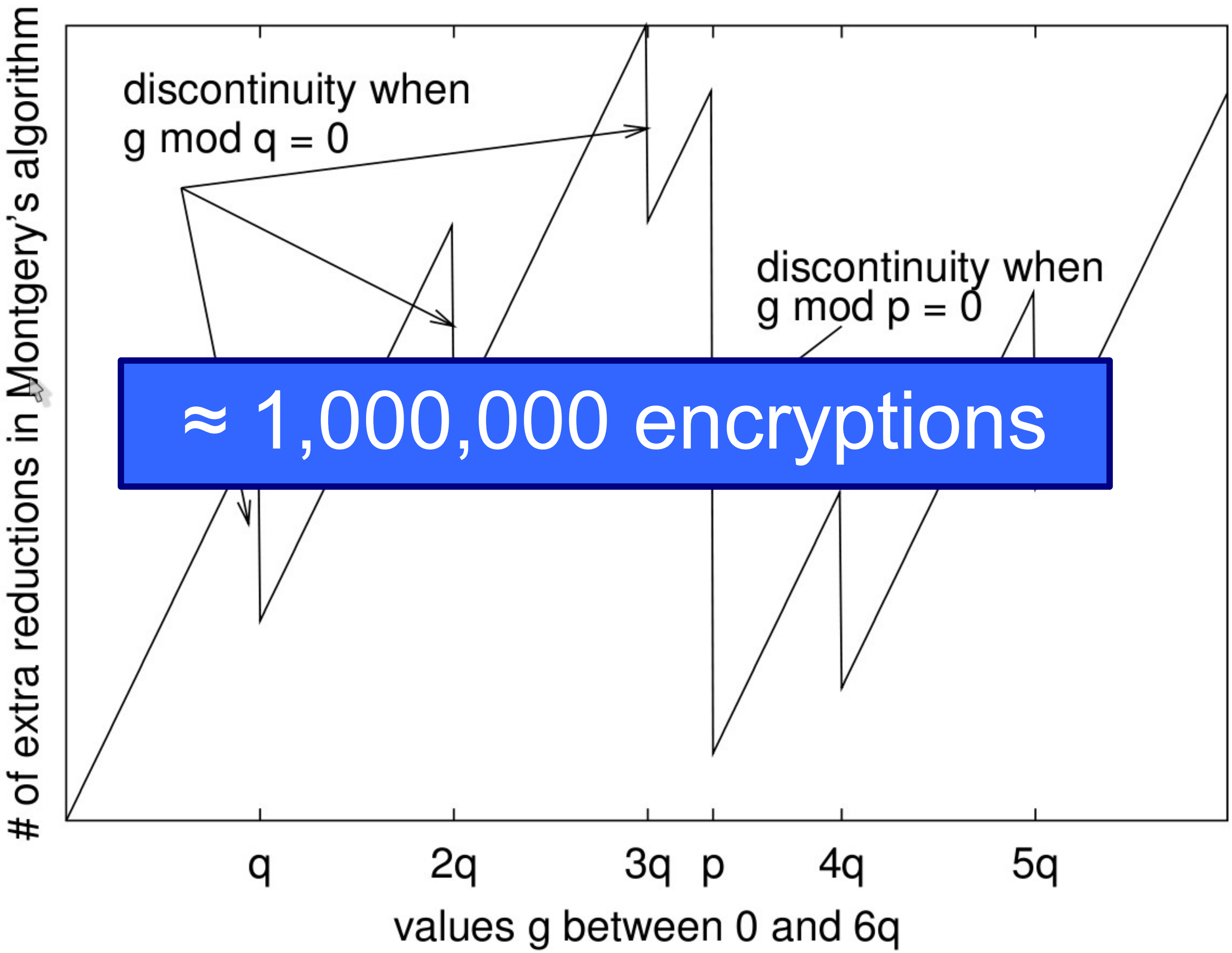
≈ 2,500 encryptions

More complicated attacks work across a LAN



Boneh and Brumley, 2003

More complicated attacks work across a LAN



≈ 1,000,000 encryptions

Boneh and Brumley, 2003

Blinded RSA provides generic defense

Private Key: p, q (random primes)

$d \equiv e^{-1} \pmod{\varphi(N)}$ (exponent)

Public Key: $N = p \cdot q$ (modulus)

e (exponent)

Signing: $s = m^d \pmod{N}$

Blind Signing: $r_1 = r_0^{\,e} \pmod{N}$

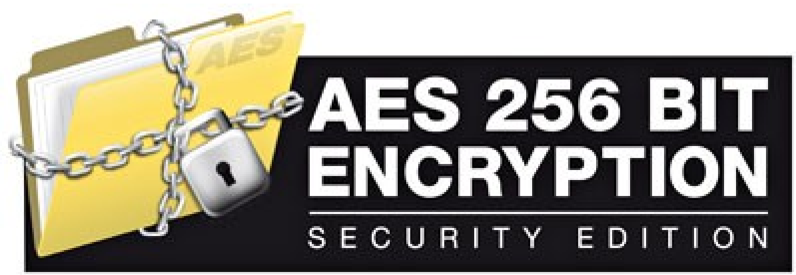
$s = r_0^{-1} (r_1 \cdot m)^d \pmod{N}$

Attacks against AES (Rijndael)

AES is cryptography's standard block cipher

AES

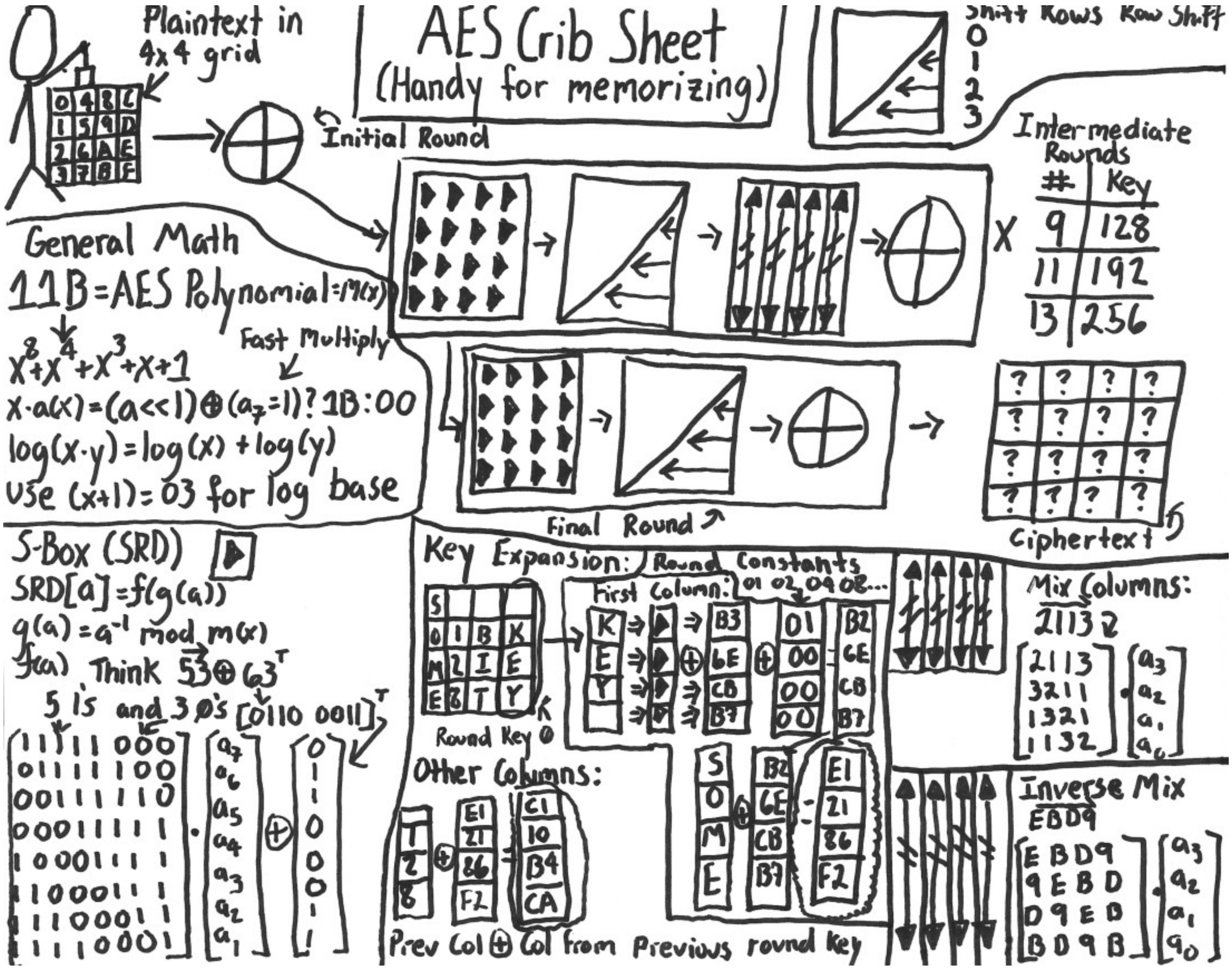
A Crypto Algorithm for the Twenty-first Century . . .



AES SOFTWARE

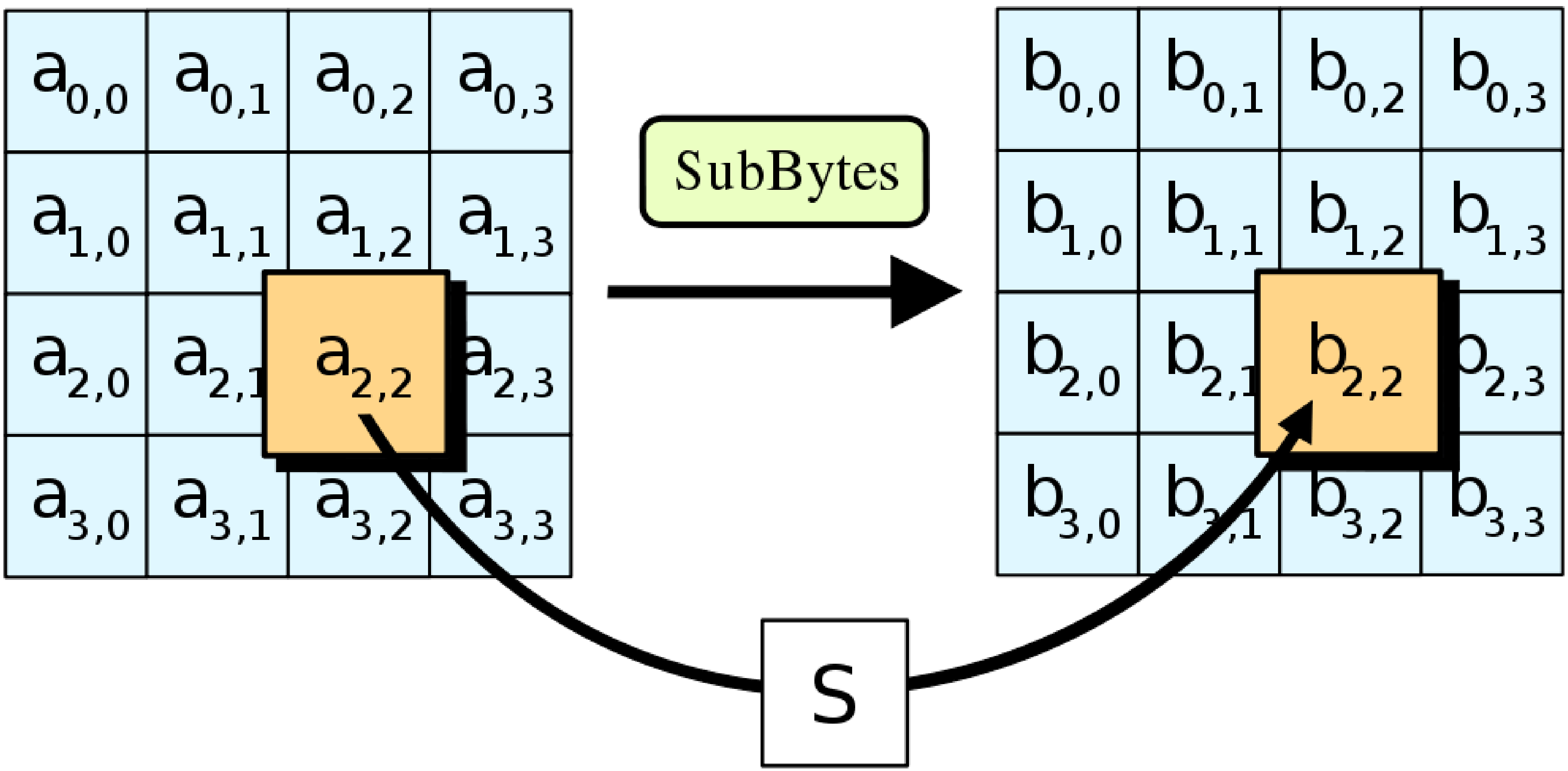
aes password manager

AES is very complicated



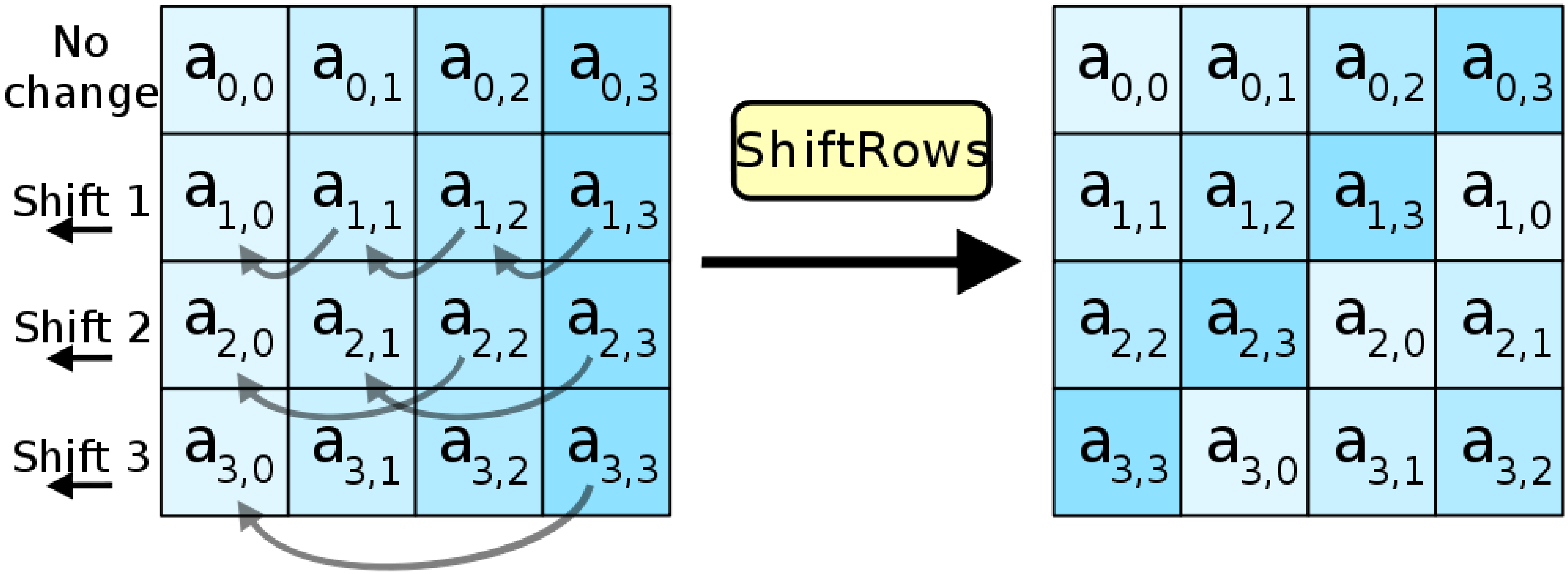
Jeff Moser

AES is very complicated



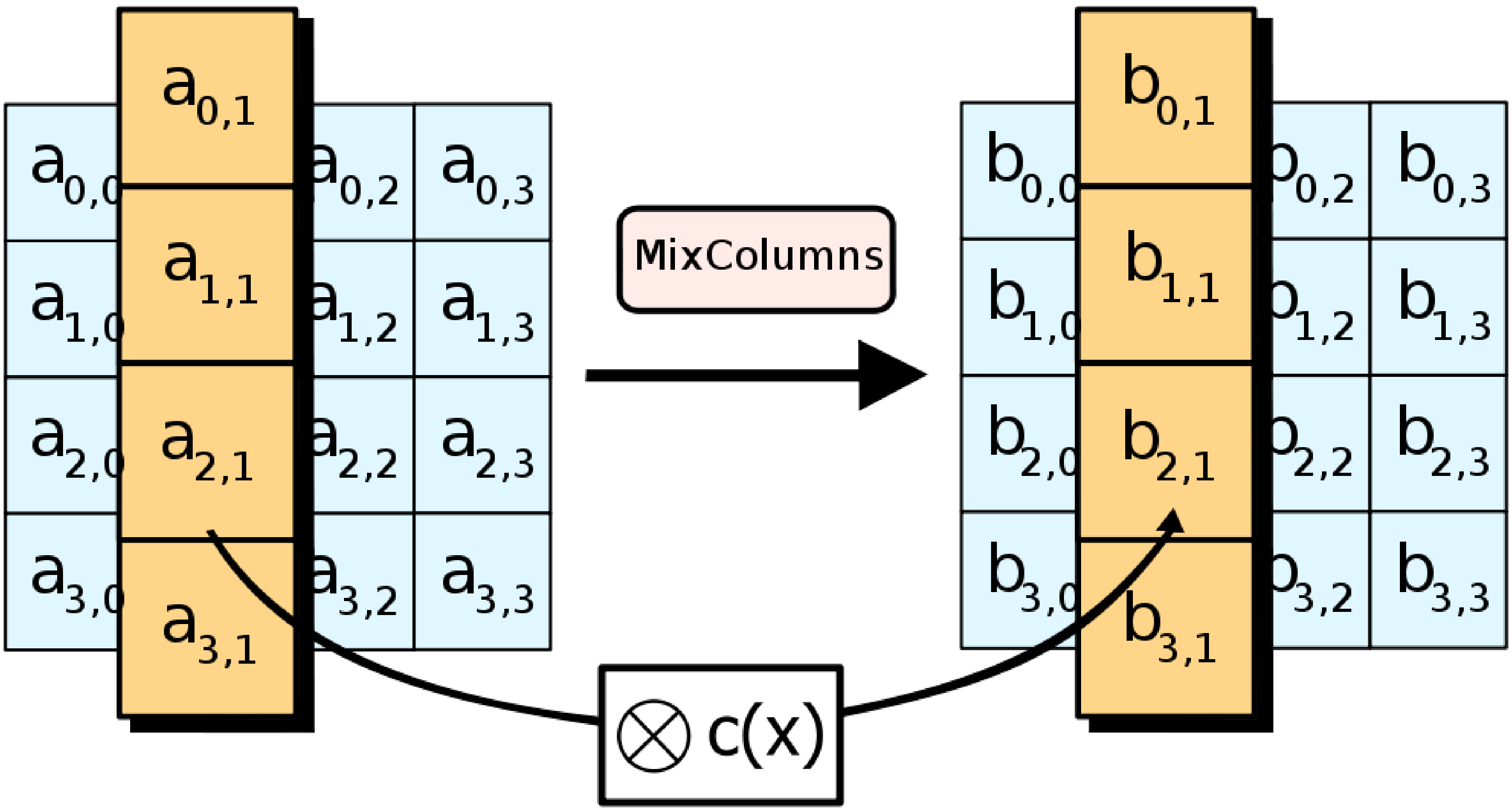
Wikipedia

AES is very complicated



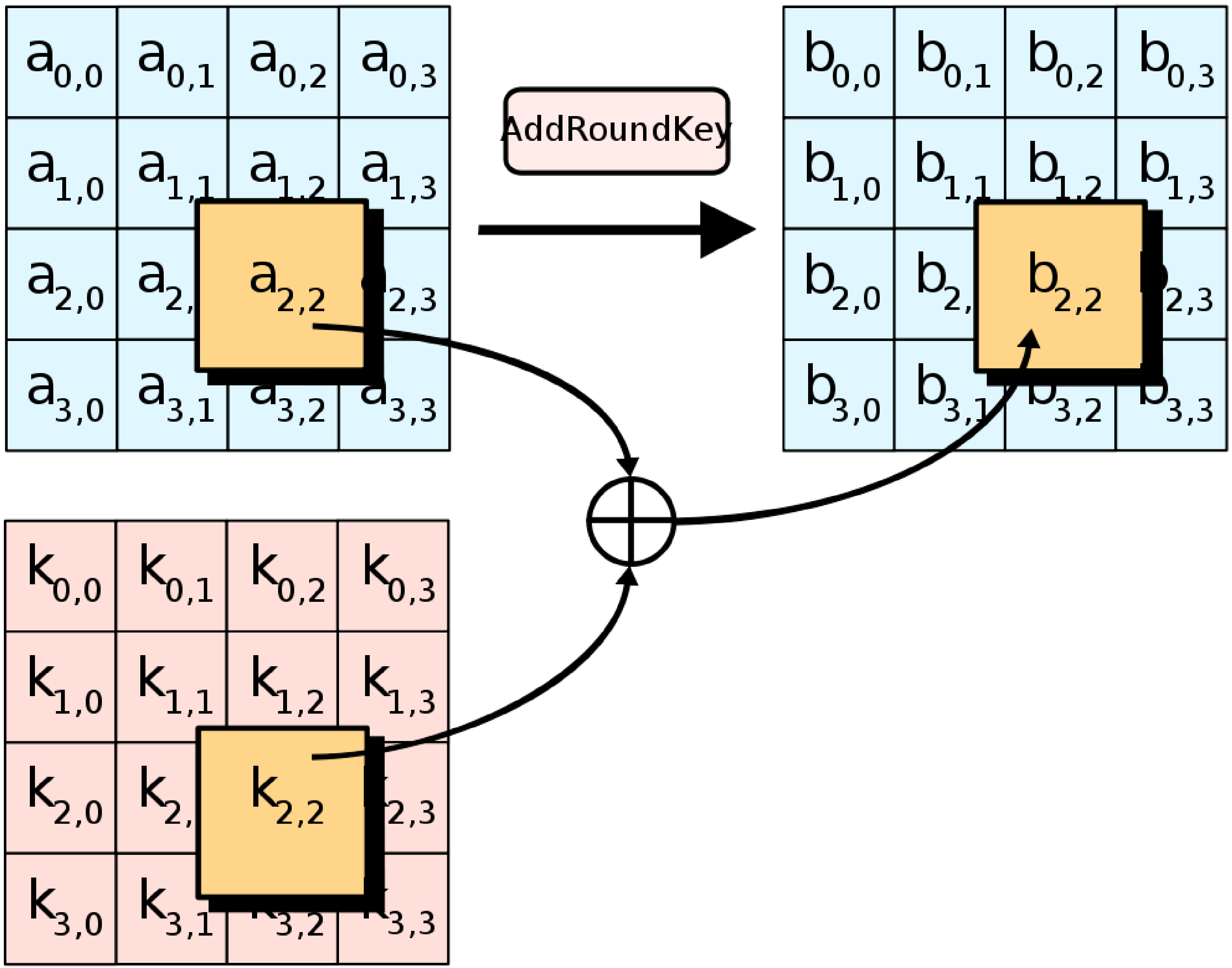
Wikipedia

AES is very complicated



Wikipedia

AES is very complicated



Wikipedia

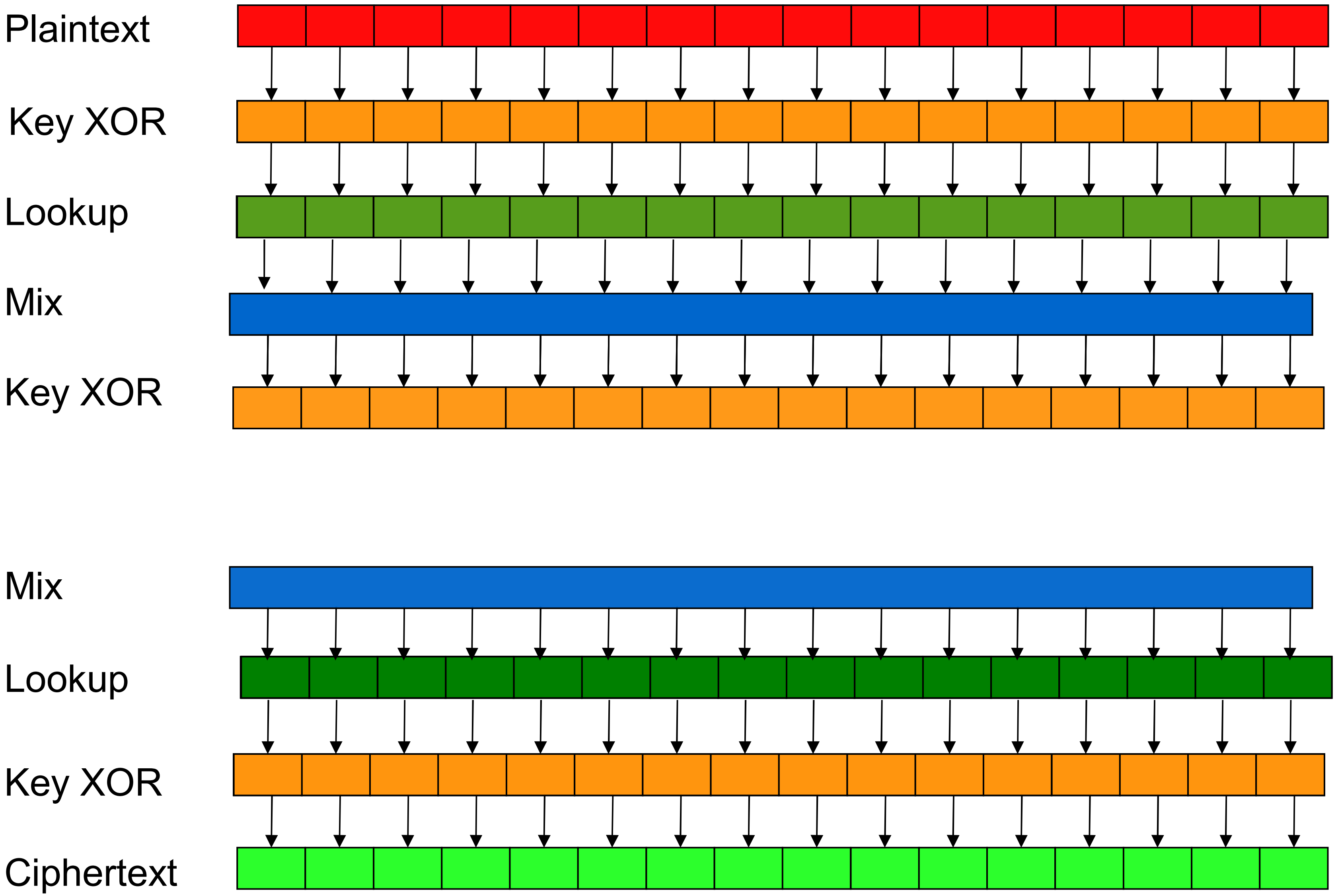
AES is designed for very efficient implementation

```
t0 =   Te0[(s0 >> 24)        ] ^
       Te1[(s1 >> 16) & 0xff] ^
       Te2[(s2 >>  8) & 0xff] ^
       Te3[(s3      ) & 0xff] ^
       rk[0];
t1 =   Te0[(s1 >> 24)        ] ^
       Te1[(s2 >> 16) & 0xff] ^
       Te2[(s3 >>  8) & 0xff] ^
       Te3[(s0      ) & 0xff] ^
       rk[1];
t2 =   Te0[(s2 >> 24)        ] ^
       Te1[(s3 >> 16) & 0xff] ^
       Te2[(s0 >>  8) & 0xff] ^
       Te3[(s1      ) & 0xff] ^
       rk[2];
t3 =   Te0[(s3 >> 24)        ] ^
       Te1[(s0 >> 16) & 0xff] ^
       Te2[(s1 >>  8) & 0xff] ^
       Te3[(s2      ) & 0xff] ^
       rk[3];
```

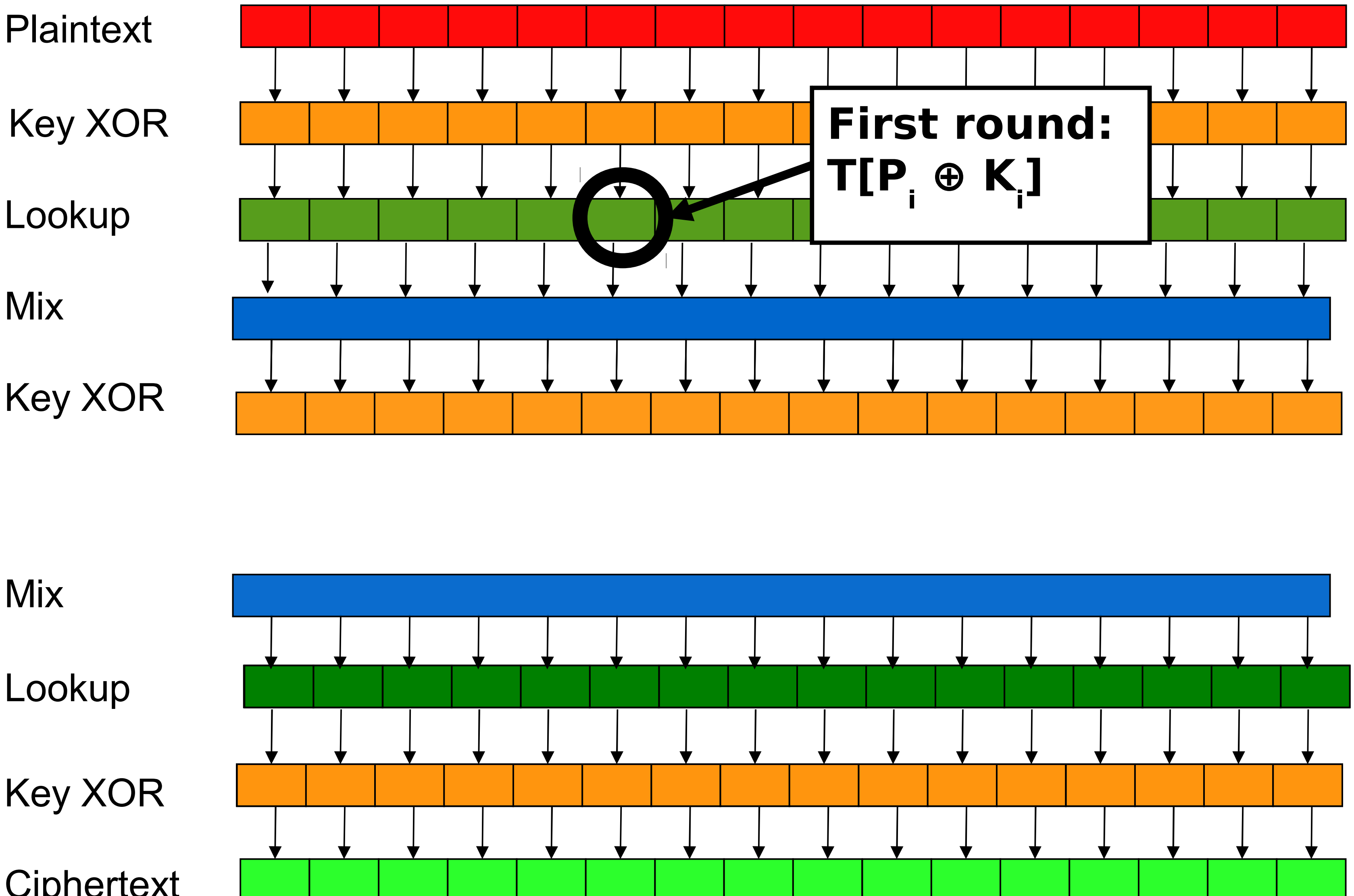
AES utilises large pre-computed lookup tables

```
static const u32 Te0[256] =
{
  0xc66363a5U, 0xf87c7c84U, 0xee777799U, 0xf67b7b8dU,
  0xfff2f20dU, 0xd66b6bbdU, 0xde6f6fb1U, 0x91c5c554U,
  0x60303050U, 0x02010103U, 0xce6767a9U, 0x562b2b7dU,
  0xe7fefe19U, 0xb5d7d762U, 0x4dababe6U, 0xec76769aU,
  ...
  0x824141c3U, 0x299999b0U, 0x5a2d2d77U, 0x1e0f0f11U,
  0x7bb0b0cbU, 0xa85454fcU, 0x6dbbbbd6U, 0x2c16163aU,
};
```

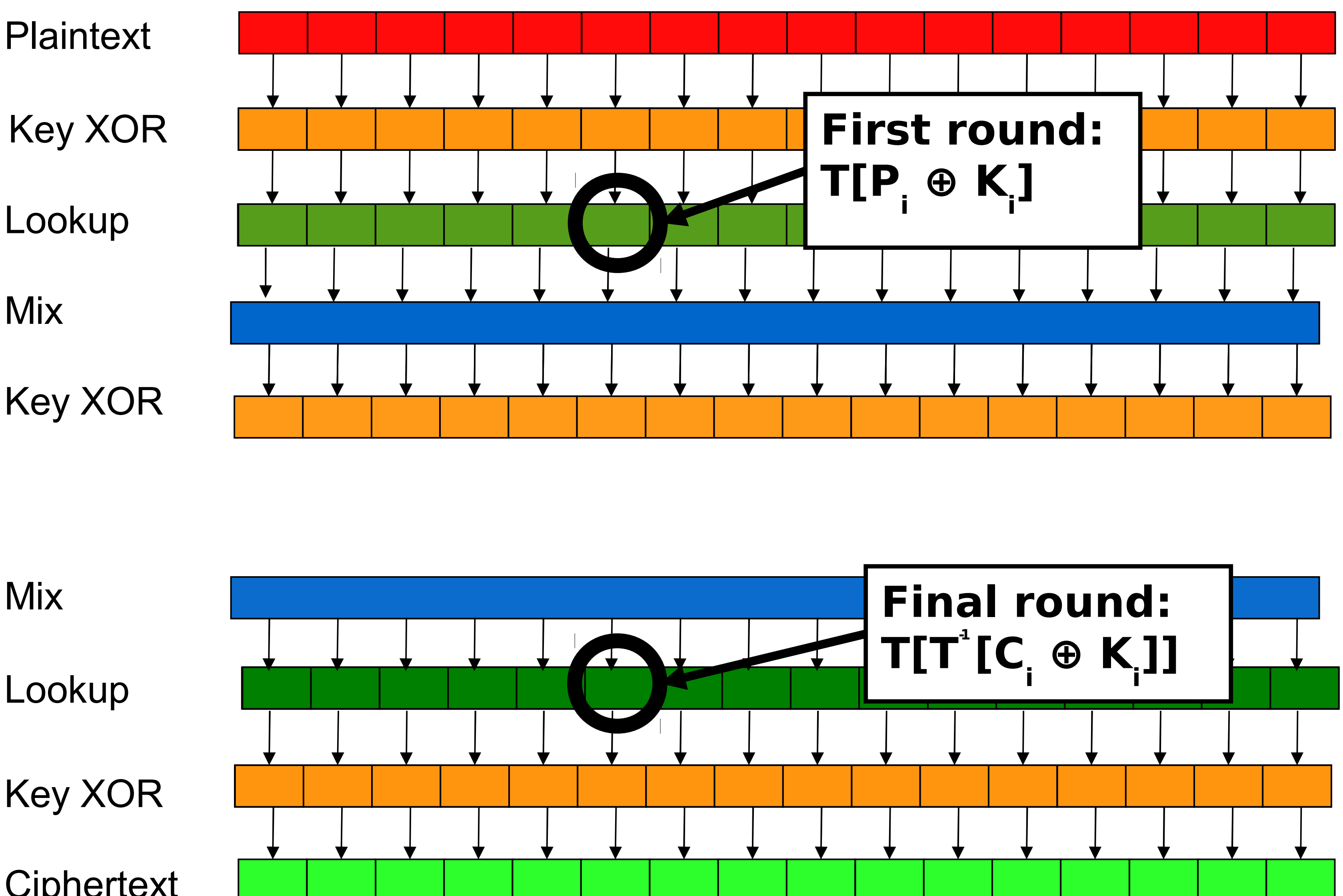
Lookups into shared cache are vulnerable



Lookups into shared cache are vulnerable



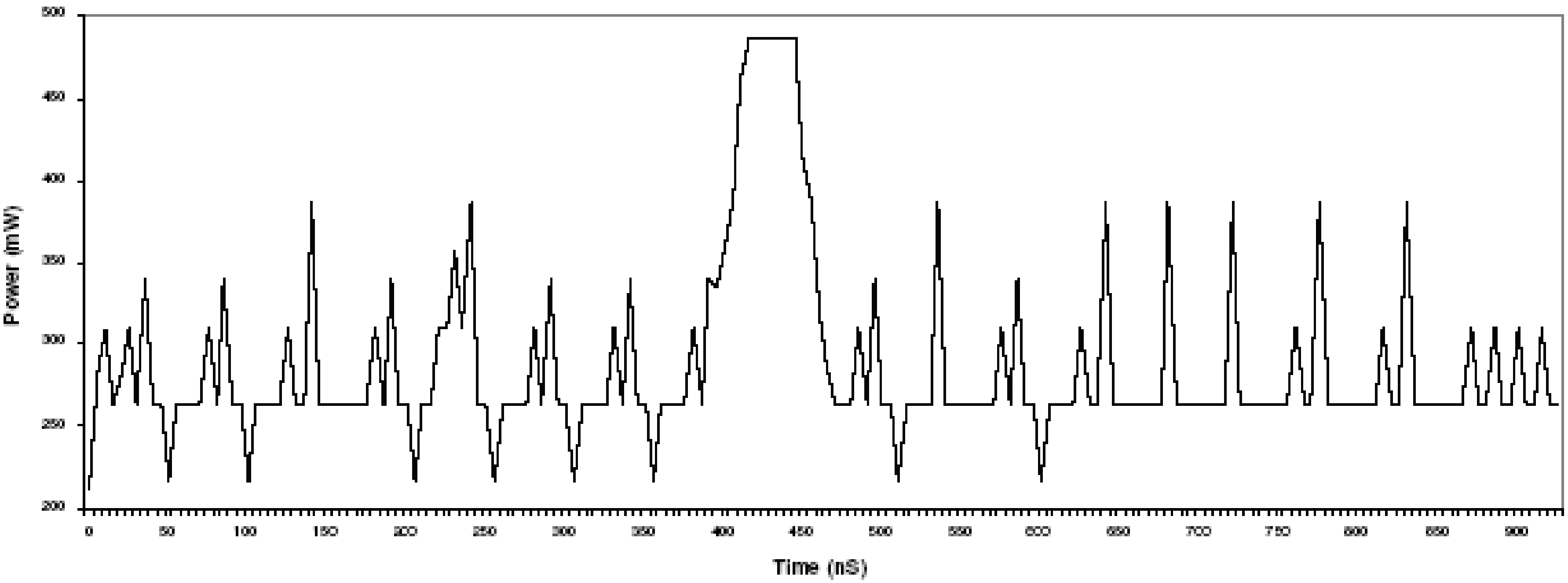
Lookups into shared cache are vulnerable



Simple power analysis of AES

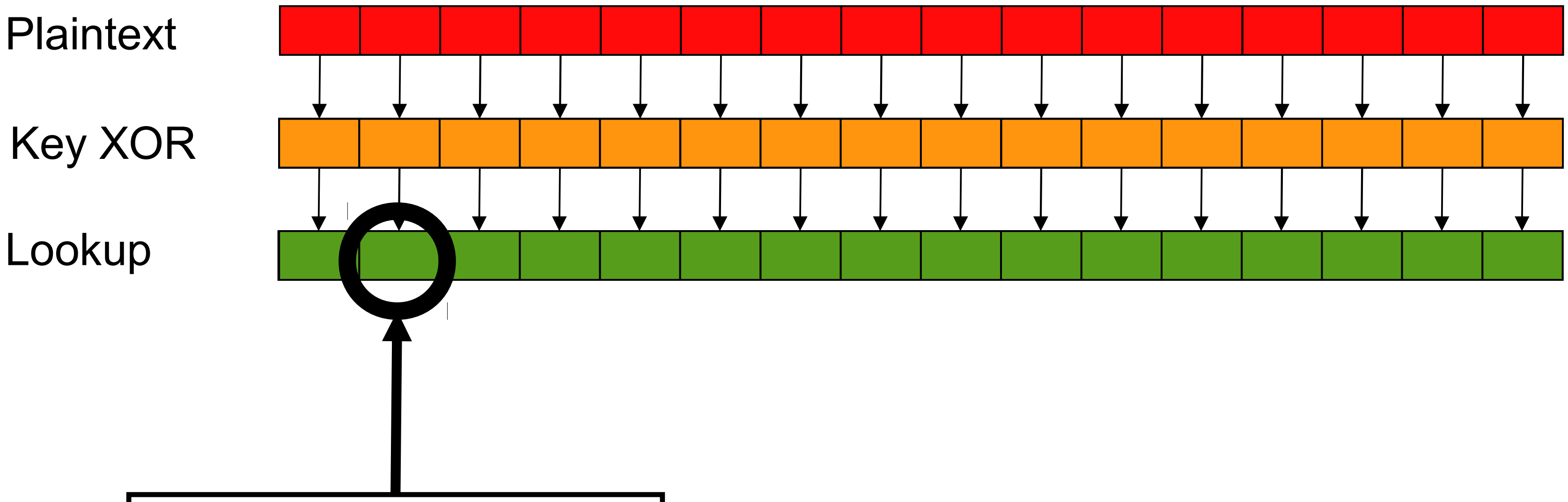
(Bertoni et. Al, 2005;
Bonneau 2006)

Cache hit/miss is very obvious in power trace



Bertoni et. al, 2005

Every miss yields many constraints



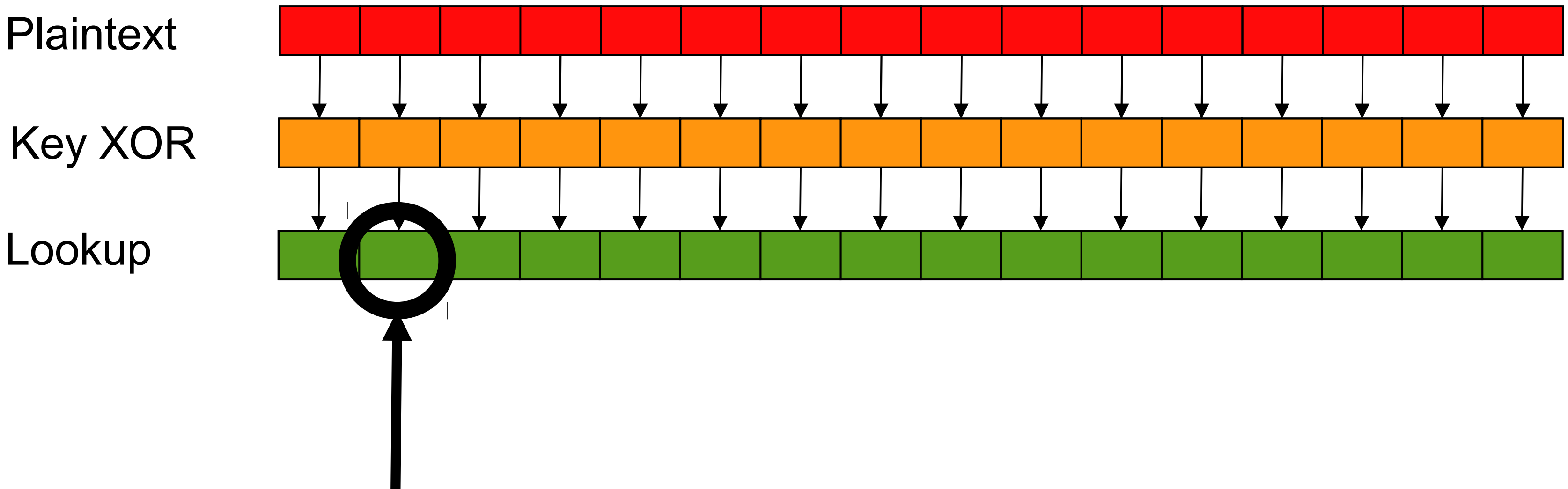
Miss?

$$P_0 \oplus K_0 \neq P_1 \oplus K_1$$

Hit?

$$P_0 \oplus K_0 \stackrel{?}{=} P_1 \oplus K_1$$

Every miss yields many constraints



Miss?

$$P_0 \oplus K_0 \neq P_1 \oplus K_1$$

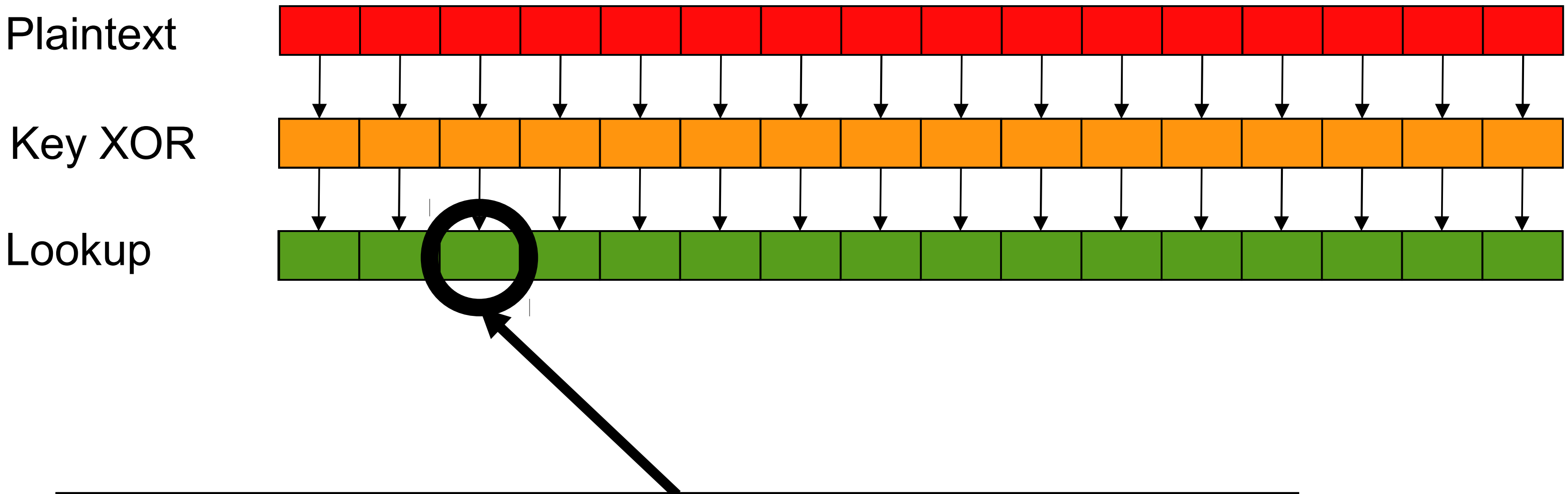
$$P_0 \oplus P_1 \neq K_0 \oplus K_1$$

Hit?

$$P_0 \oplus K_0 \stackrel{?}{=} P_1 \oplus K_1$$

$$P_0 \oplus P_1 \stackrel{?}{=} K_0 \oplus K_1$$

Every miss yields many constraints



Miss?

$$P_0 \oplus P_2 \neq K_0 \oplus K2 \;\wedge\; P_1 \oplus P_2 \neq K_1 \oplus K_2$$

Table of possible key byte differences refined

	K0	K1	K2	...	K15
K0	00	{27,e0}	{35}	{23,70,c4}	{65}
K1		00	{32,45,89}	{5f,f3}	{0a,db}
K2			00	{86}	{17,64,9c}
...				00	{42,d5}
K15					00

Table of possible key byte differences refined

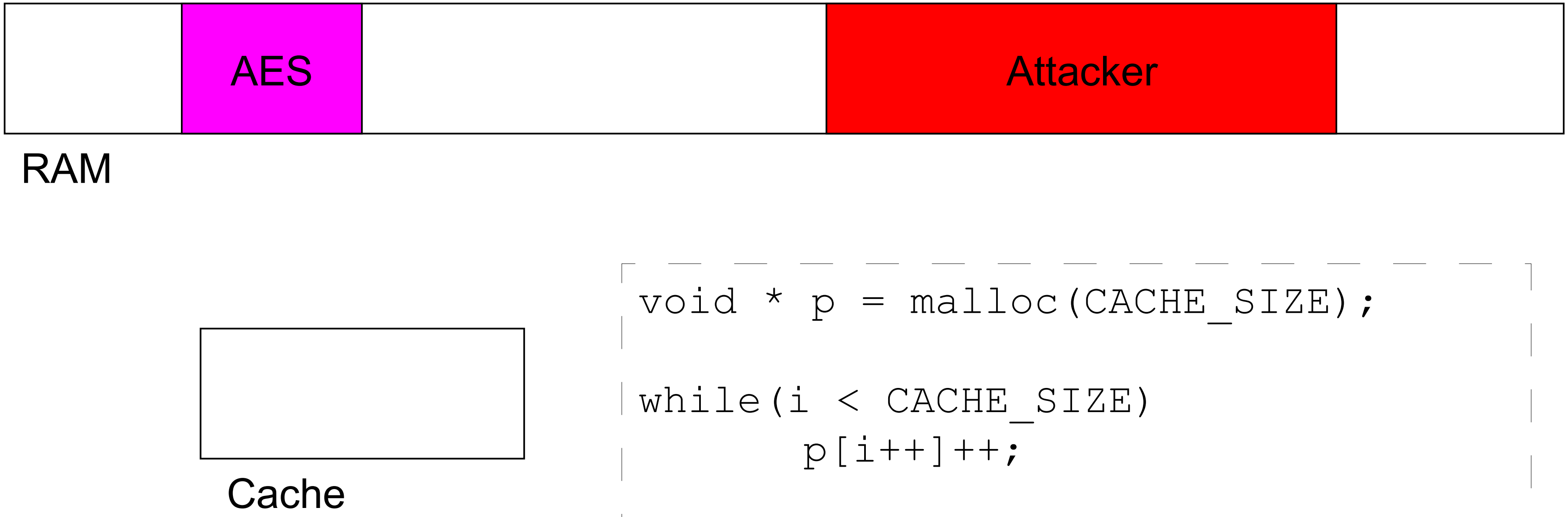
	K0	K1	K2	...	K15
K0	00	{27,e0}	{35}	{23,70,c [illegible]	{65}
K1		[illegible]	[illegible] 9}	[illegible]	{0a,db}
K2			00	{86}	{17,64,9c}
...				00	{42,d5}
K15					00

≈ 100 encryptions

Cache observation attack

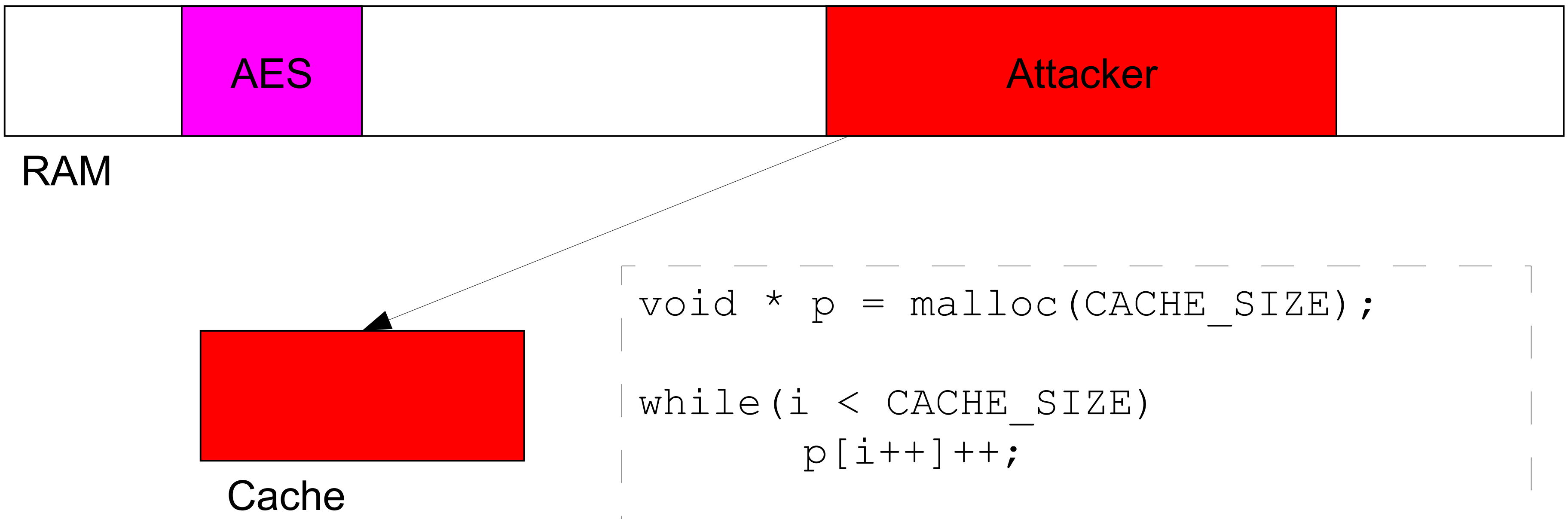
(Osvik et. al, 2006)

1) Attacker "primes" the cache with known data



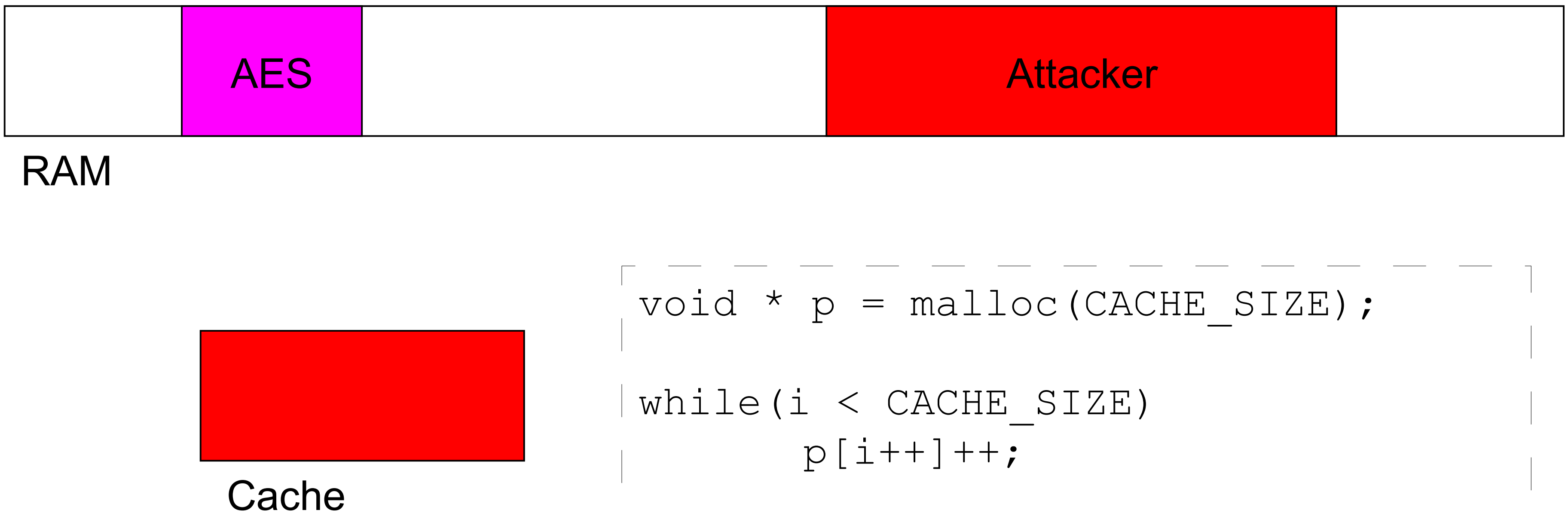

```
void * p = malloc(CACHE_SIZE);

while(i < CACHE_SIZE)
       p[i++]++;
```

1) Attacker "primes" the cache with known data



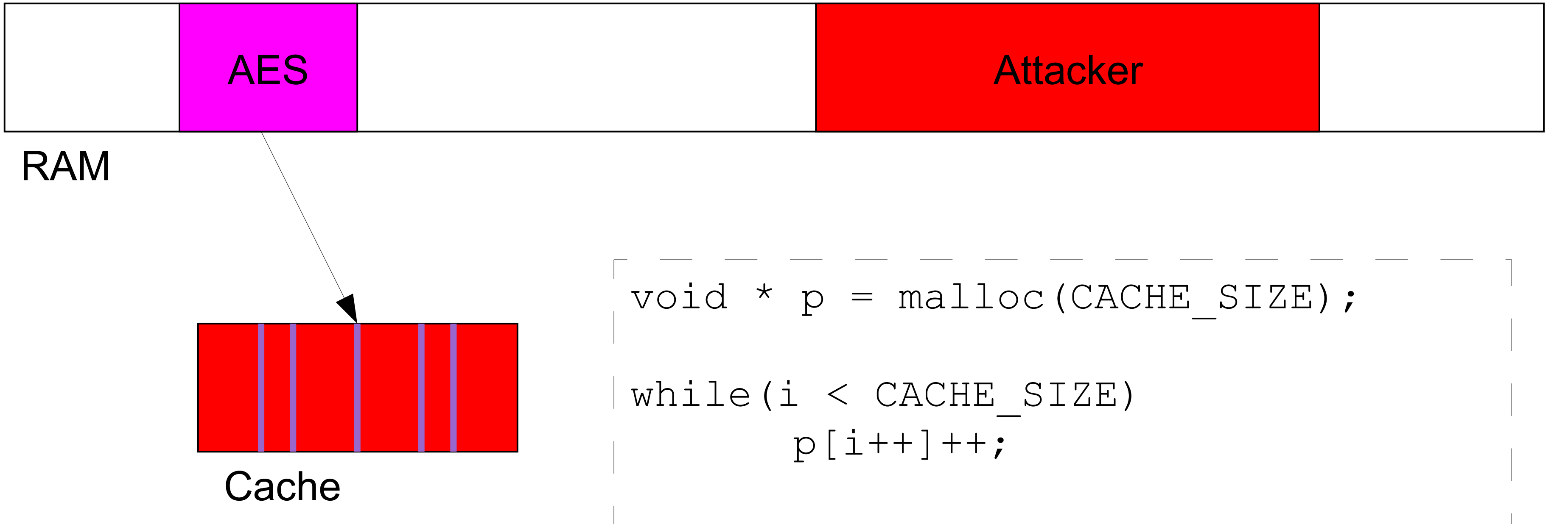

```
void * p = malloc(CACHE_SIZE);

while(i < CACHE_SIZE)
      p[i++]++;
```

2) Attacker triggers AES encryption



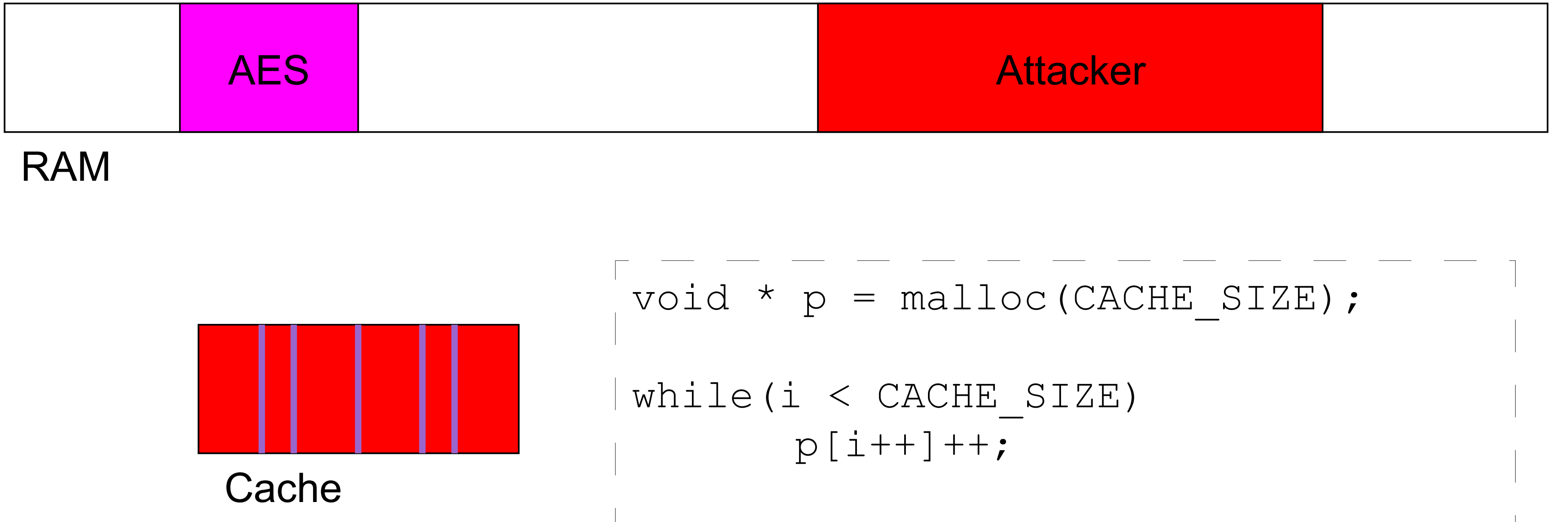

```
void * p = malloc(CACHE_SIZE);

while(i < CACHE_SIZE)
        p[i++]++;

aes_encrypt(random_p());
```

3) AES loads some cache lines



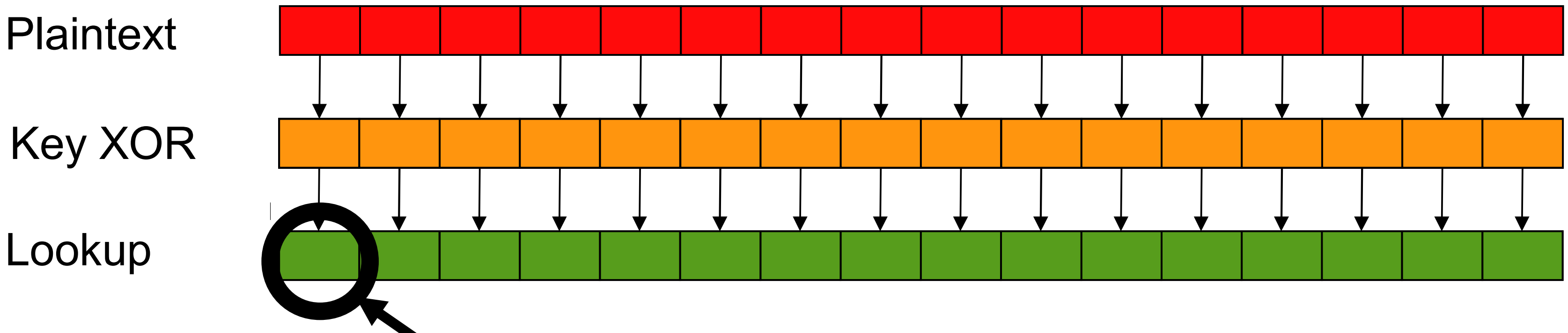

```
void * p = malloc(CACHE_SIZE);

while(i < CACHE_SIZE)
        p[i++]++;

aes_encrypt(random_p());
```

4) Attacker can test which lines were touched



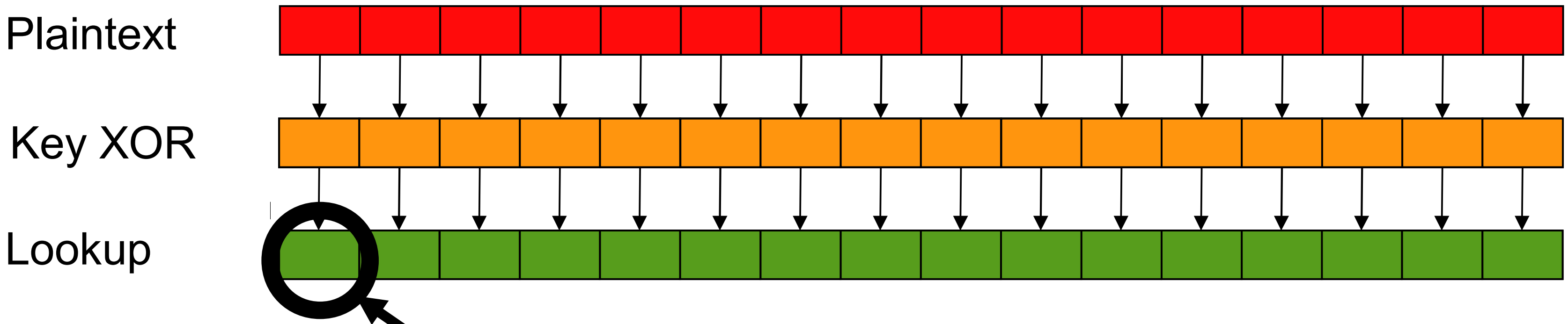

```
void * p = malloc(CACHE_SIZE);

while(i < CACHE_SIZE)
        p[i++]++;

aes_encrypt(random_p());

while(i < CACHE_SIZE)
        t[i++] = timed_read(p, i);
```

5) All untouched lines yield constraints



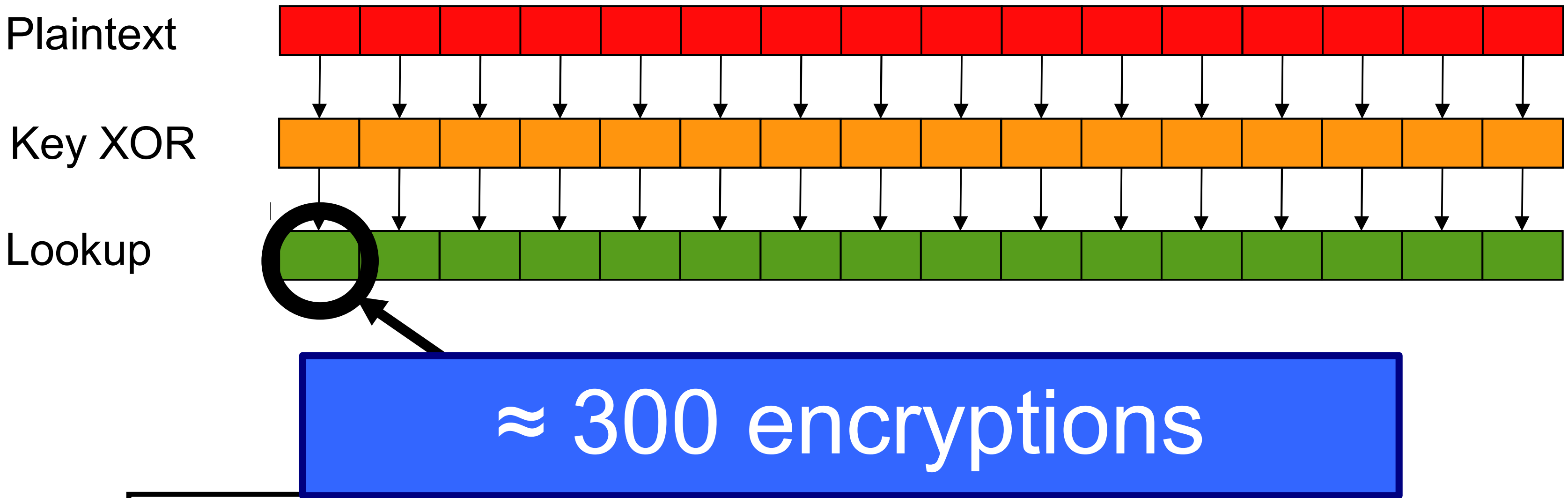
$$P_0 \oplus K_0 \notin \{\text{Untouched lines}\}$$

5) All untouched lines yield constraints



$K_0 \notin \{\text{Untouched lines} \oplus P_0\}$

5) All untouched lines yield constraints

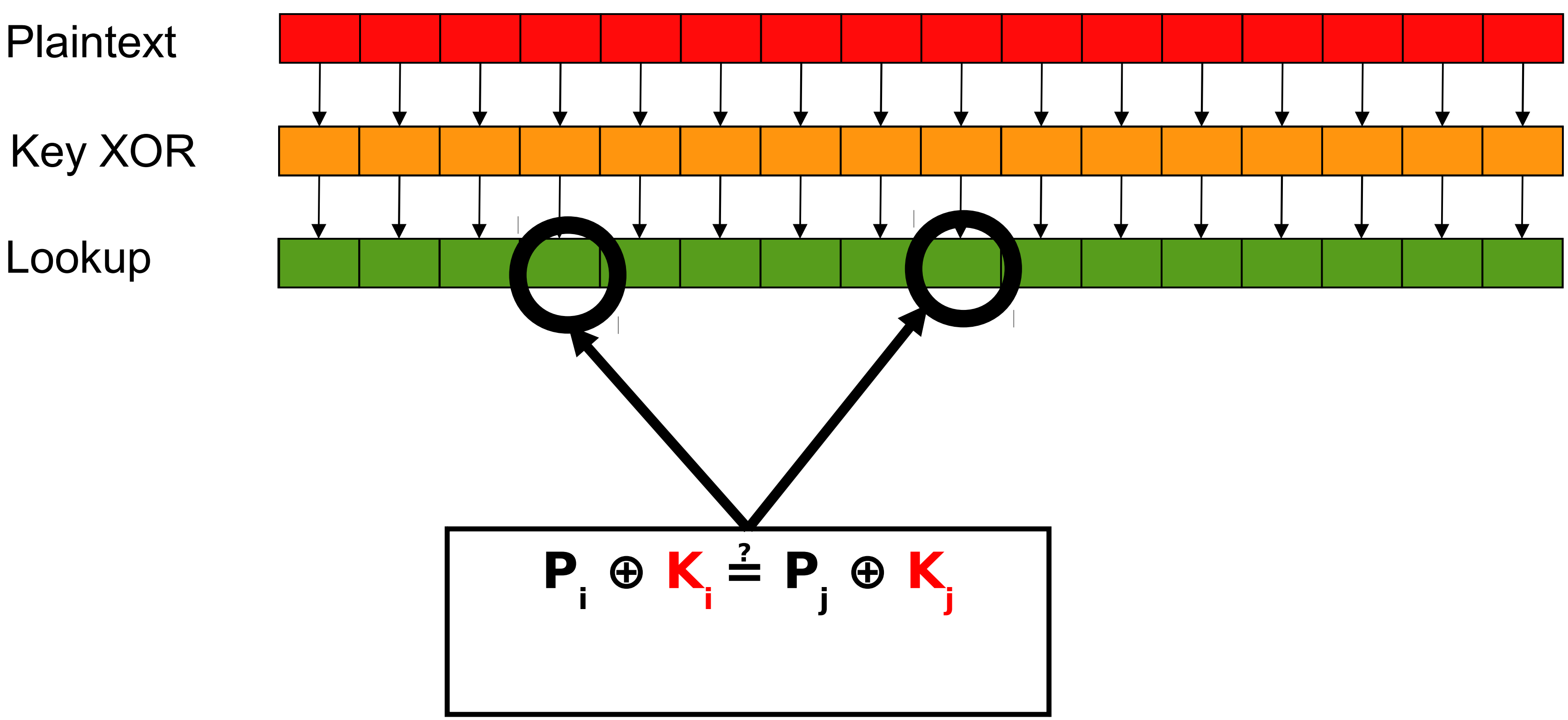


$K_0 \notin \{\text{Untouched lines} \oplus P_0\}$

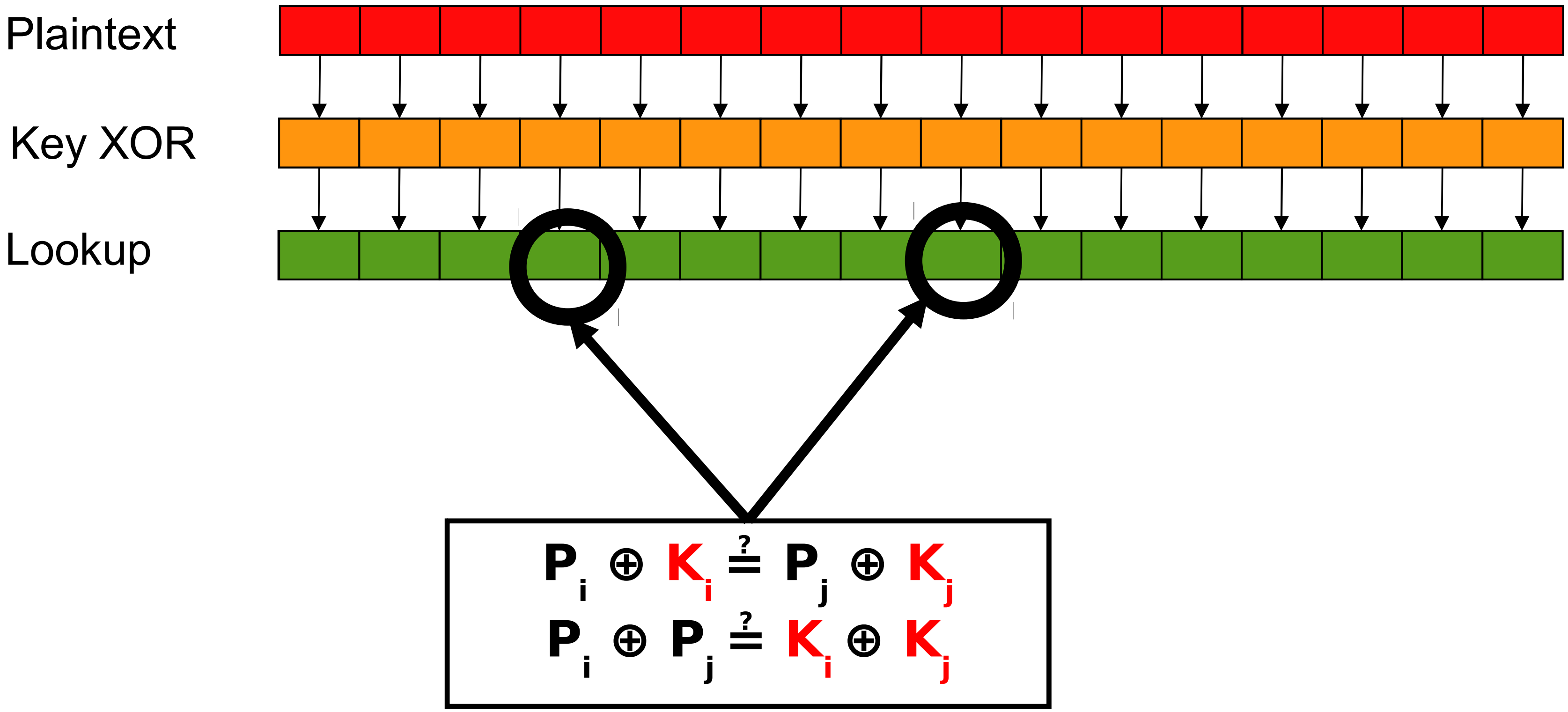
Cache timing attack

(Bonneau and Mironov, 2006)

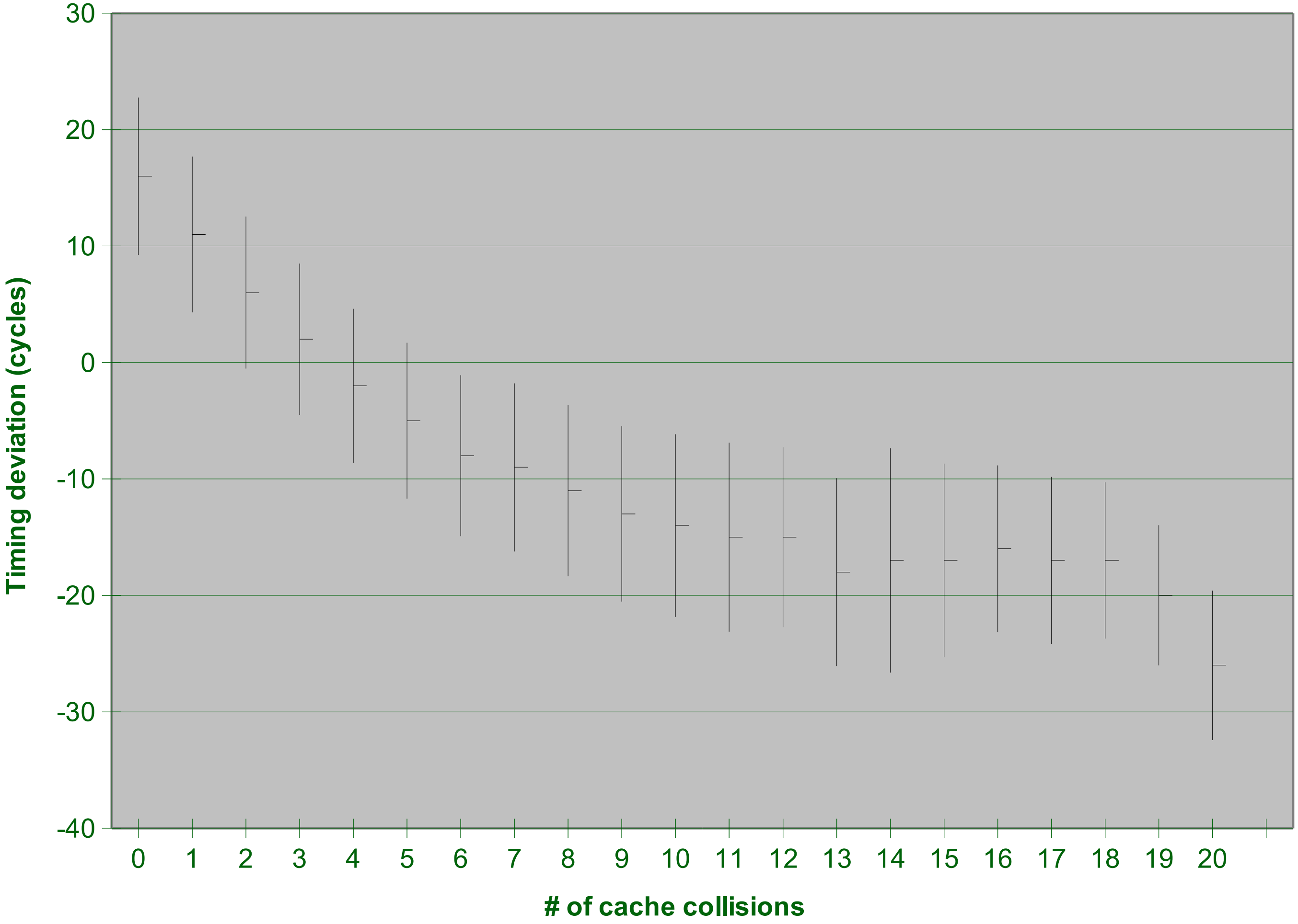
Observation: self-collisions lower encryption time



Observation: self-collisions lower encryption time



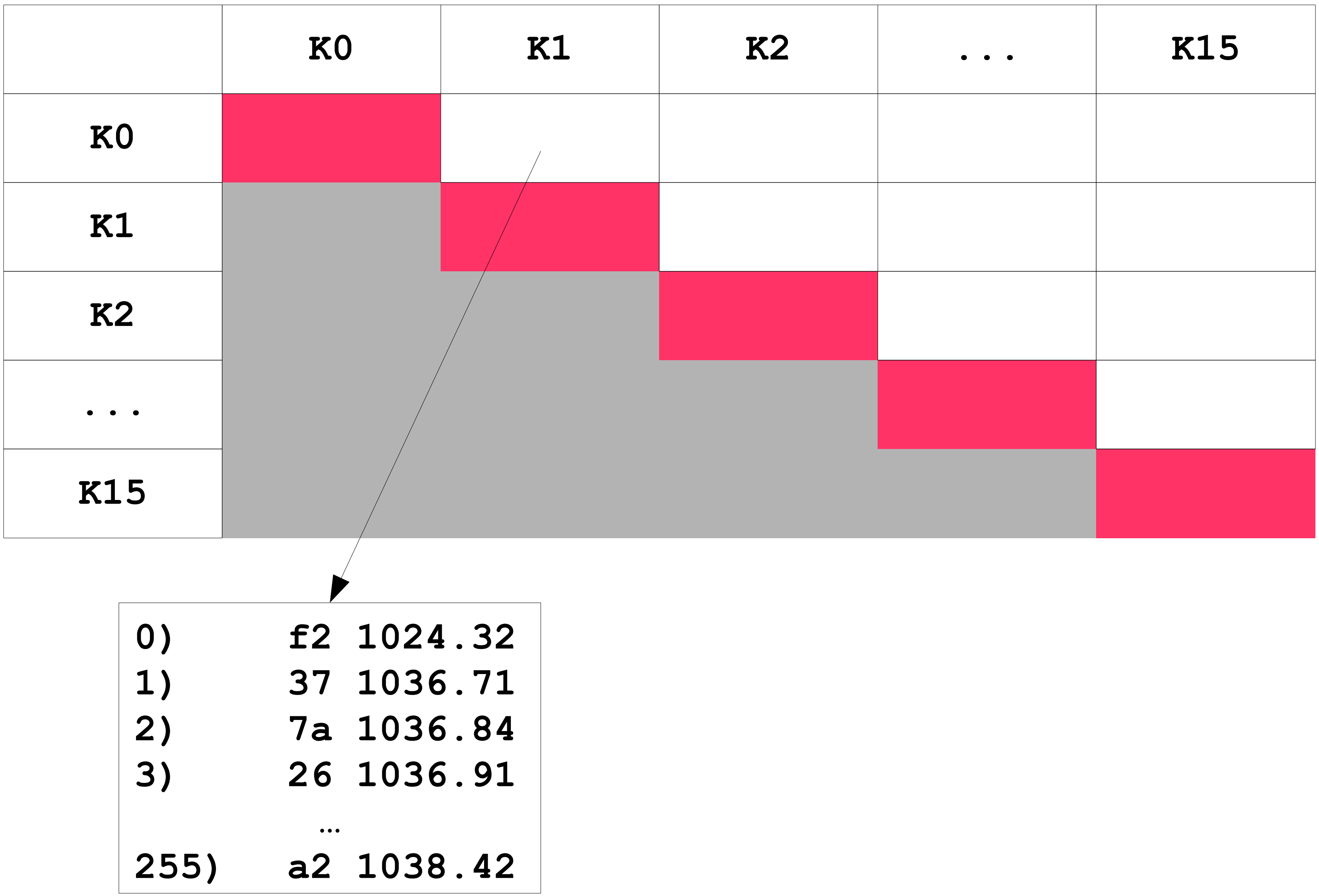
Internal collisions cause most timing variation



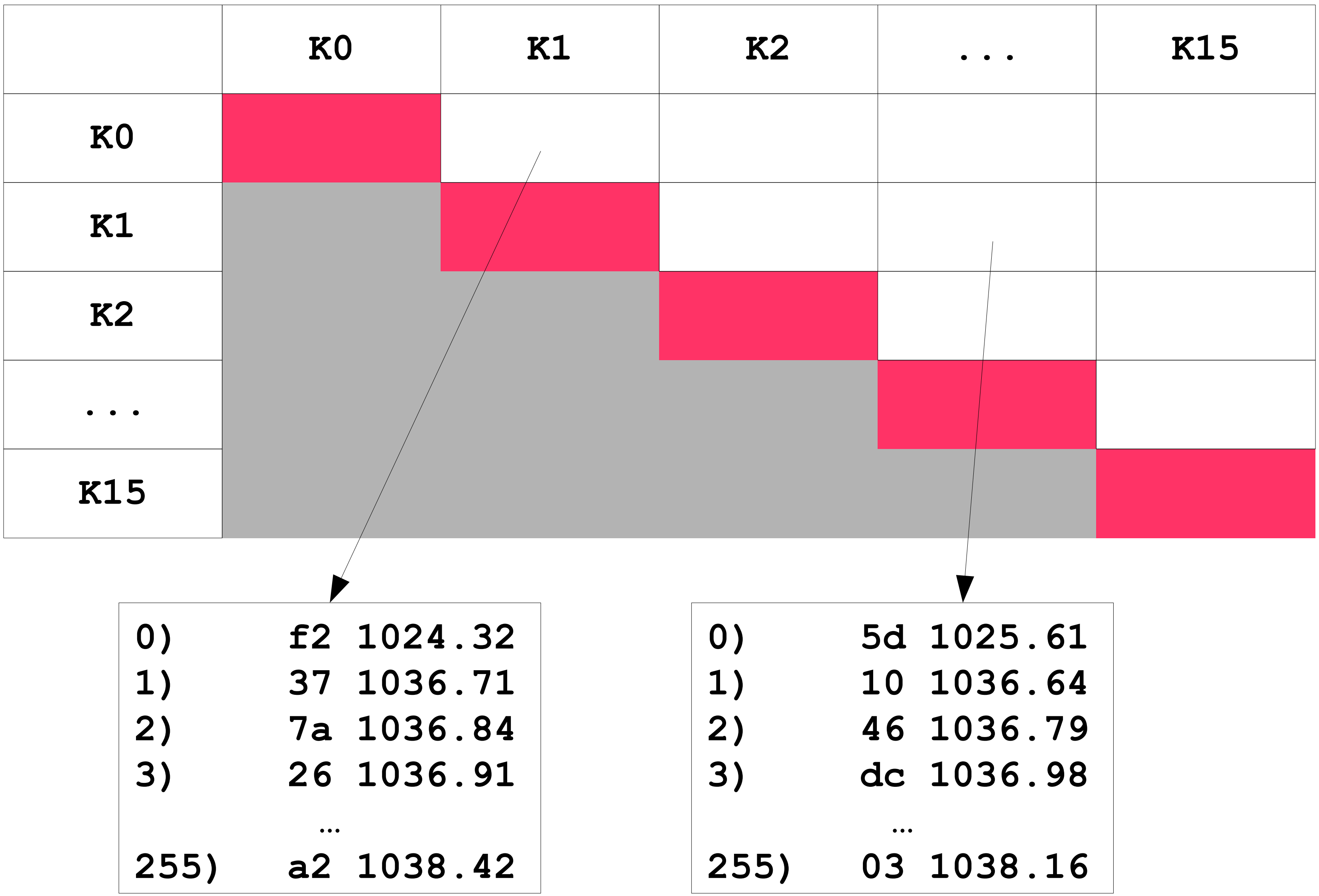
Key byte differences ranked by average time

	K0	K1	K2	...	K15
K0					
K1					
K2					
...					
K15					

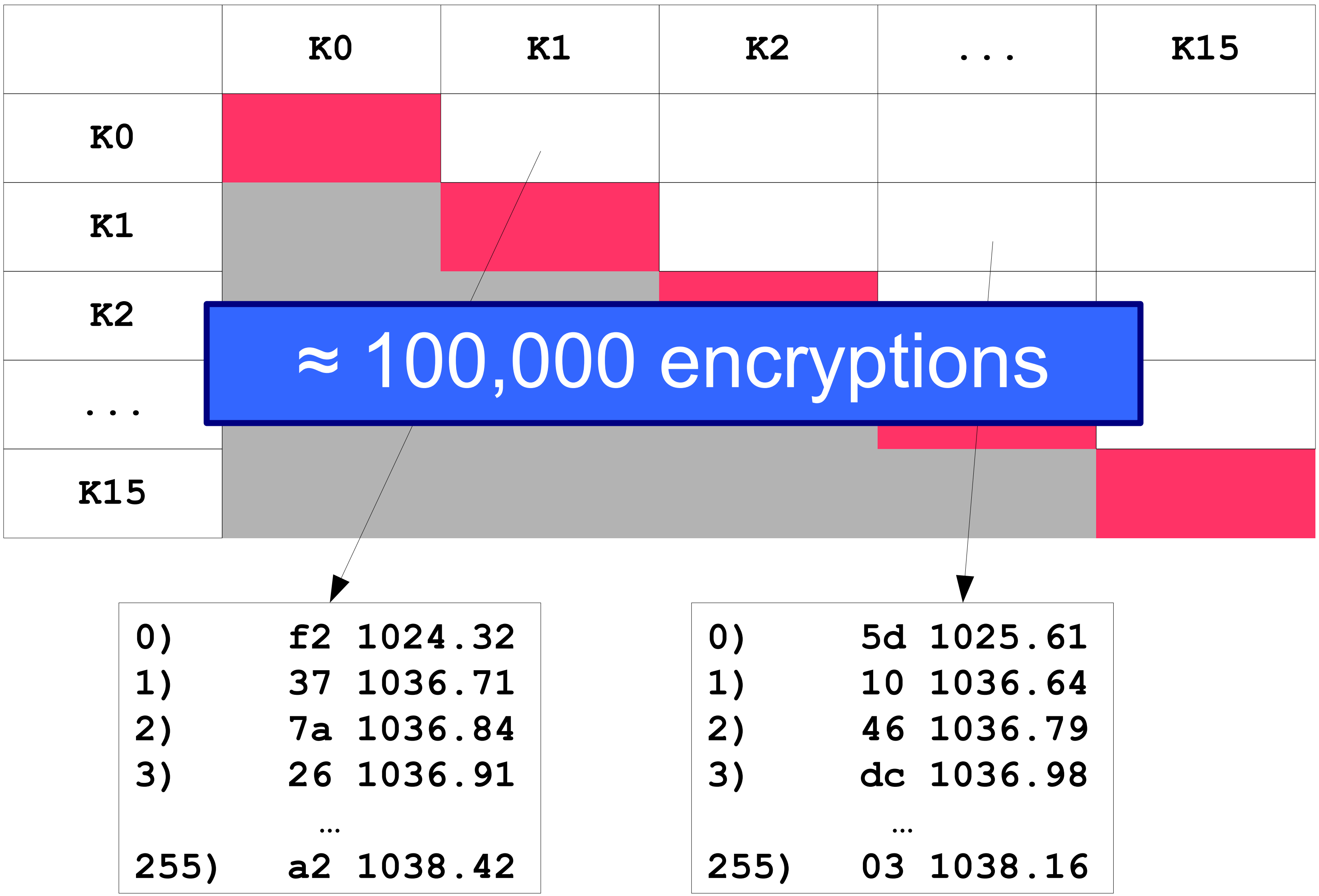
Key byte differences ranked by average time



Key byte differences ranked by average time



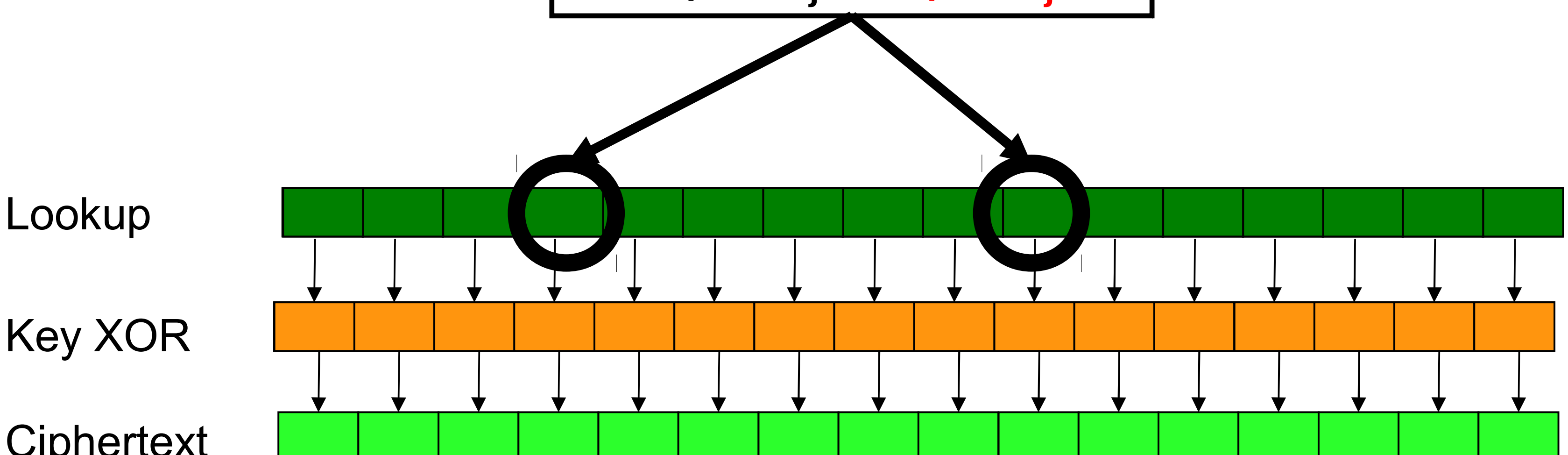
Key byte differences ranked by average time



Final round is much better to attack

$$C_i \oplus K_i = S[X]$$
$$C_j \oplus K_j = S[Y]$$

$$X = Y \Rightarrow$$
$$C_i \oplus K_i = C_j \oplus K_j$$
$$C_i \oplus C_j = K_i \oplus K_j$$



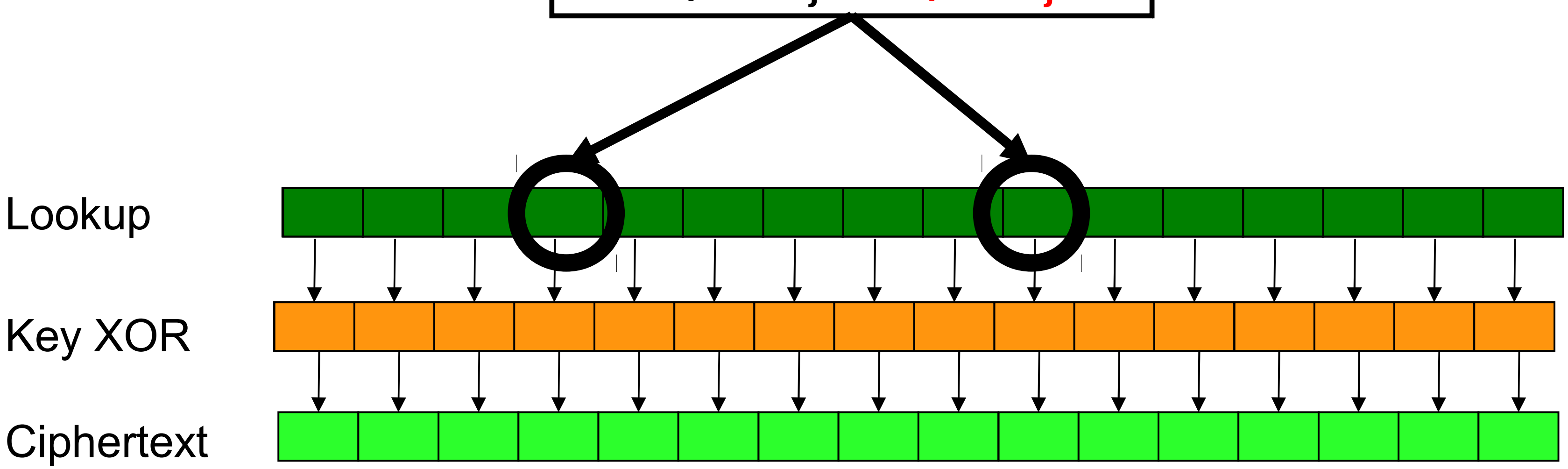
Final round is much better to attack

$C_i \oplus K_i = S[X]$

$C_j \oplus K_j = S[Y]$

≈ 32,000 encryptions

$C_i \oplus C_j = K_i \oplus K_j$



Hardware countermeasures on the way

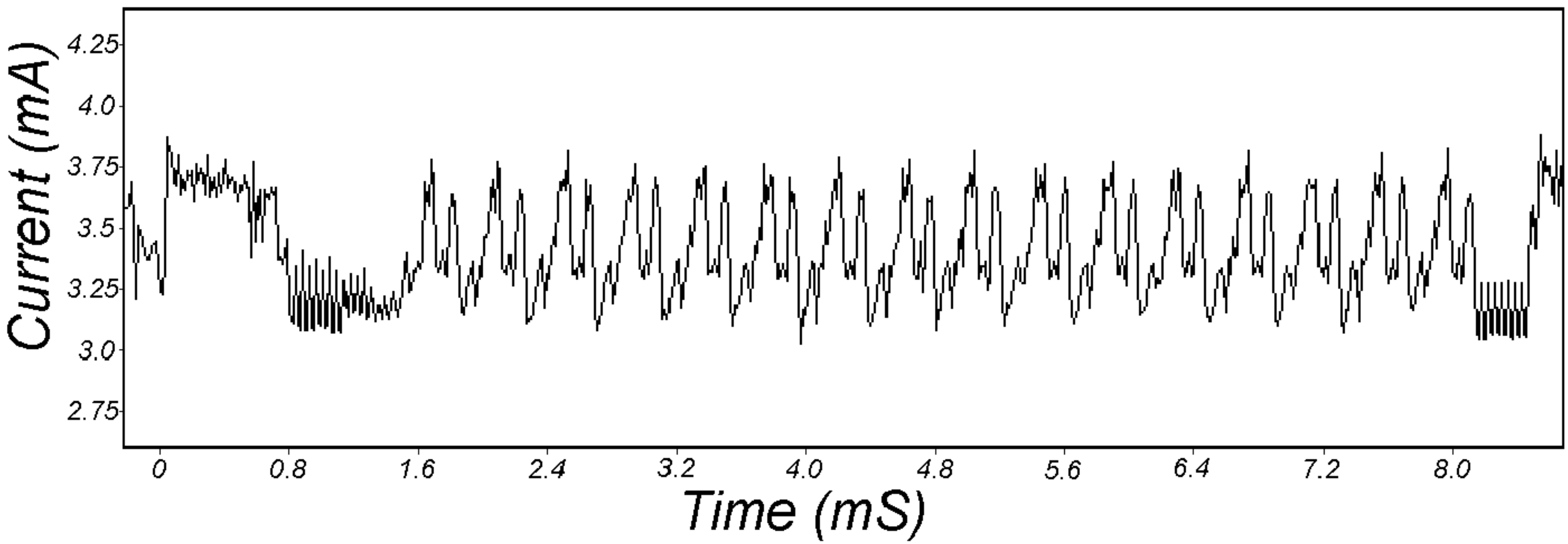
```
/*
AES-128 encryption sequence.
The data block is in xmm15.
Registers xmm0-xmm10 hold the round keys(from 0 to 10 in
this order).
In the end, xmm15 holds the encryption result.
*/
      pxor xmm15, xmm0            // Input whitening
      aesenc xmm15, xmm1          // Round 1
      aesenc xmm15, xmm2          // Round 2
      aesenc xmm15, xmm3          // Round 3
      aesenc xmm15, xmm4          // Round 4
      aesenc xmm15, xmm5          // Round 5
      aesenc xmm15, xmm6          // Round 6
      aesenc xmm15, xmm7          // Round 7
      aesenc xmm15, xmm8          // Round 8
      aesenc xmm15, xmm9          // Round 9
      aesenclast xmm15, xmm10     // Round 10
```

Courtesy of Intel

Differential power analysis

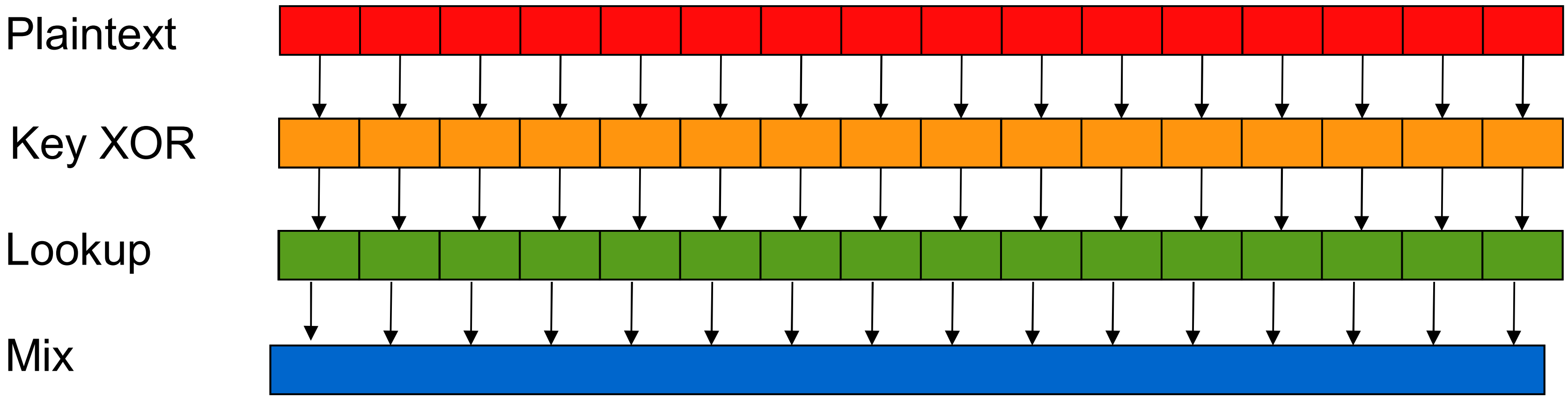
(Kocher et. al, 1999)

Simple power analysis ineffective

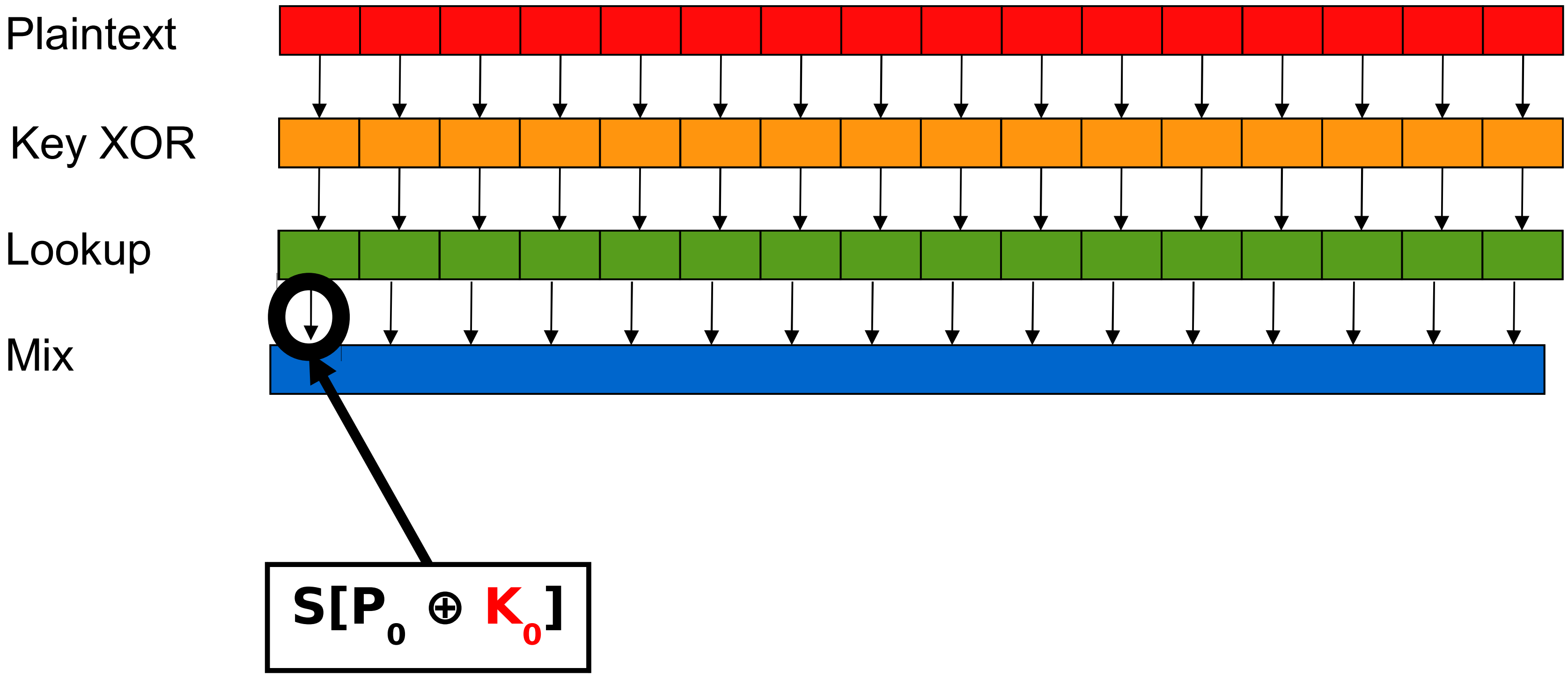


Trace courtesy of Cryptography Research, Inc.

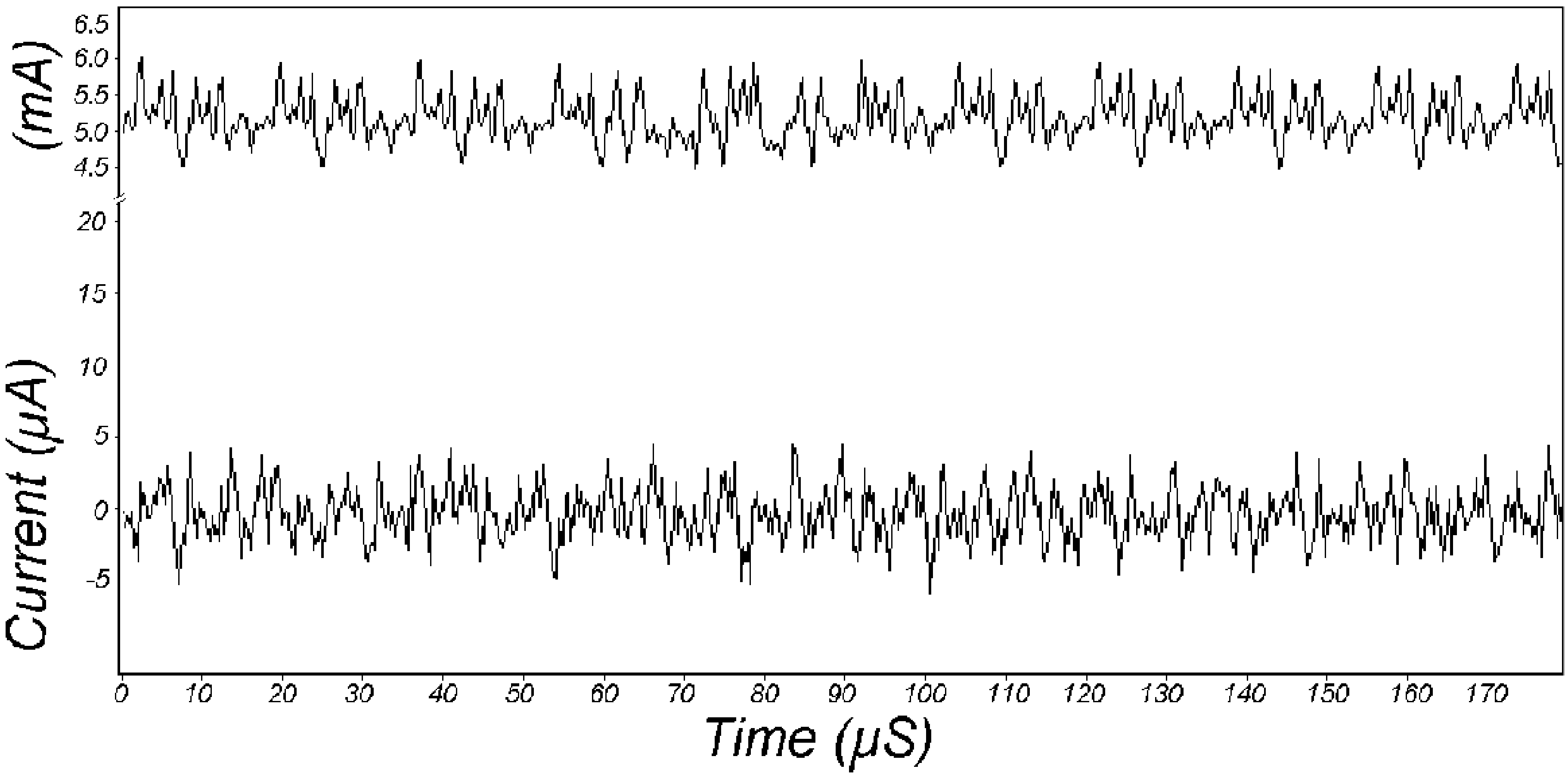
Hardware implementations don't use cache



Hardware implementations don't use cache

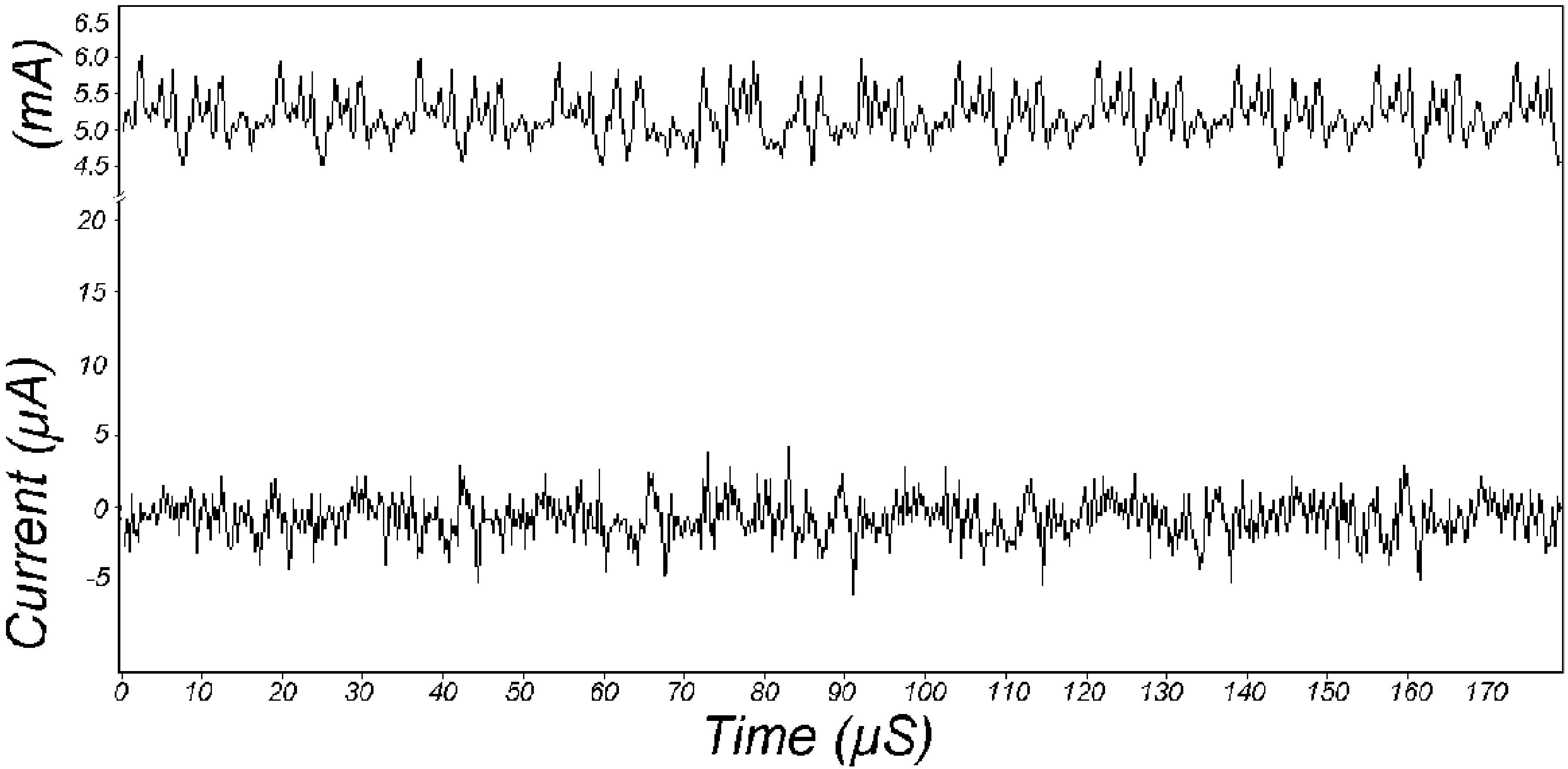


Partition traces by some predicted intermediate bit



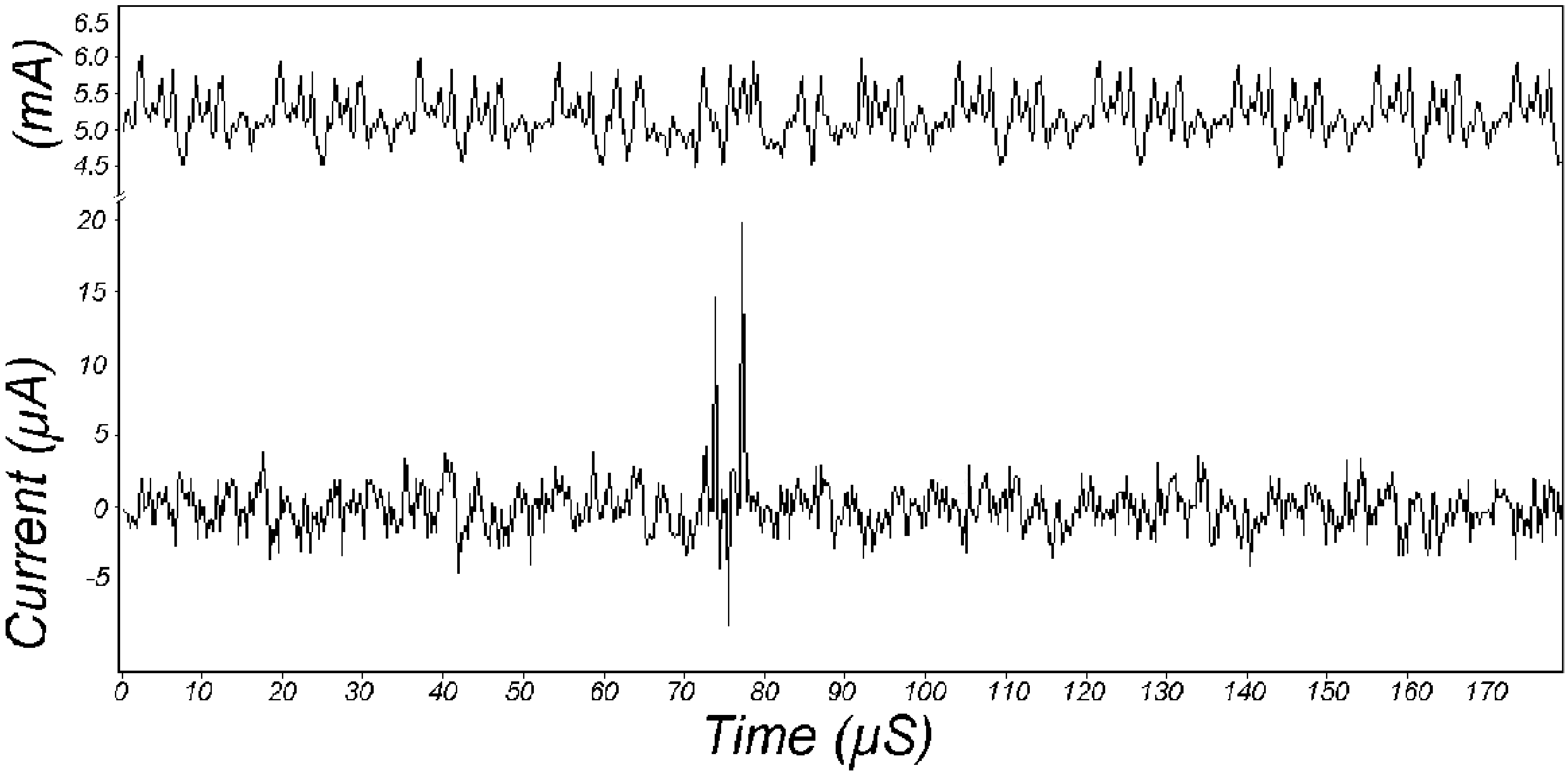
Guessing K_0 = 00, traces where high bit of $S[P_0 \oplus K_0]$ is set

Partition traces by some predicted intermediate bit



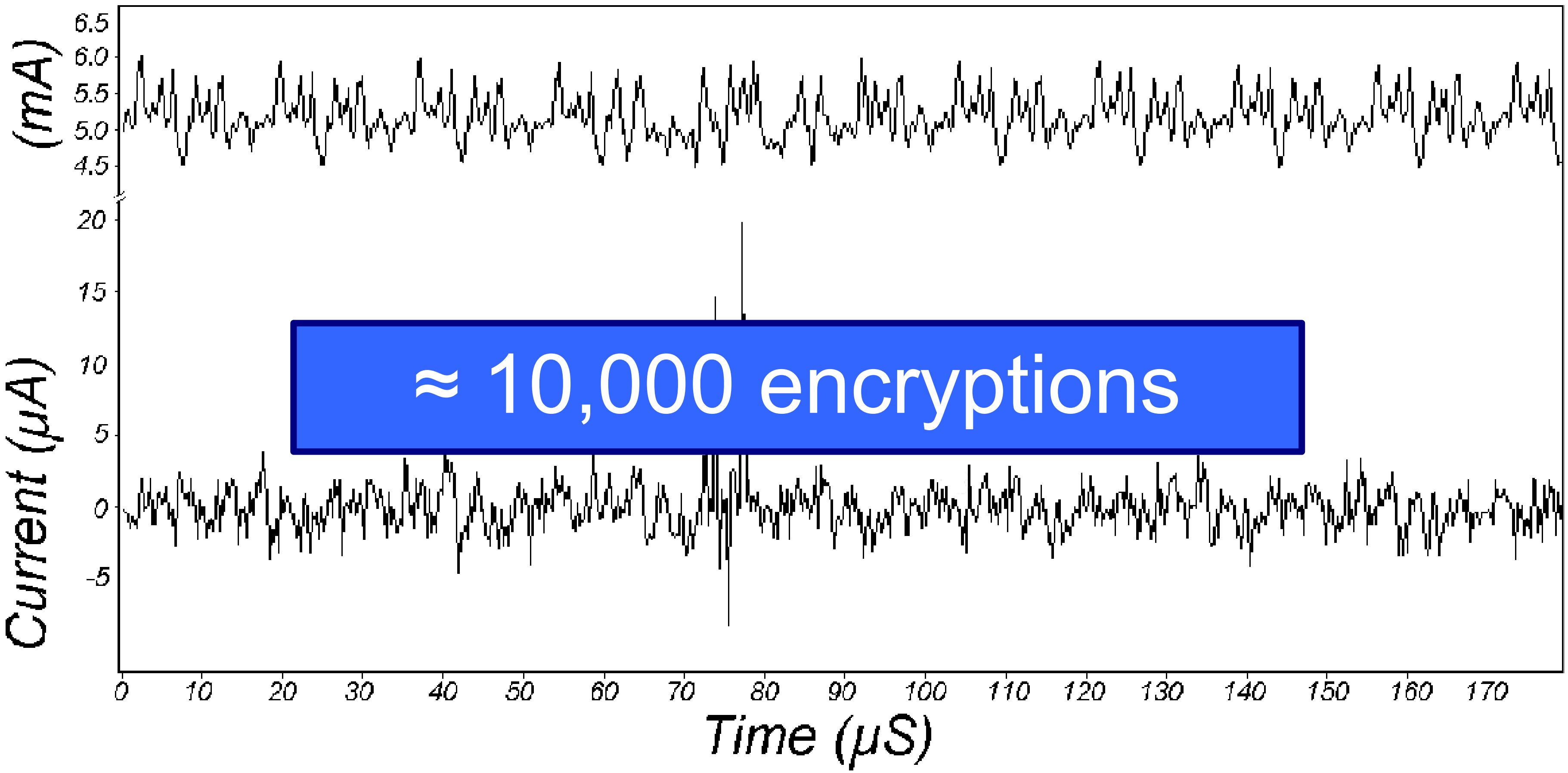
Guessing K_0 = 01, traces where high bit of $S[P_0 \oplus K_0]$ is set

Partition traces by some predicted intermediate bit



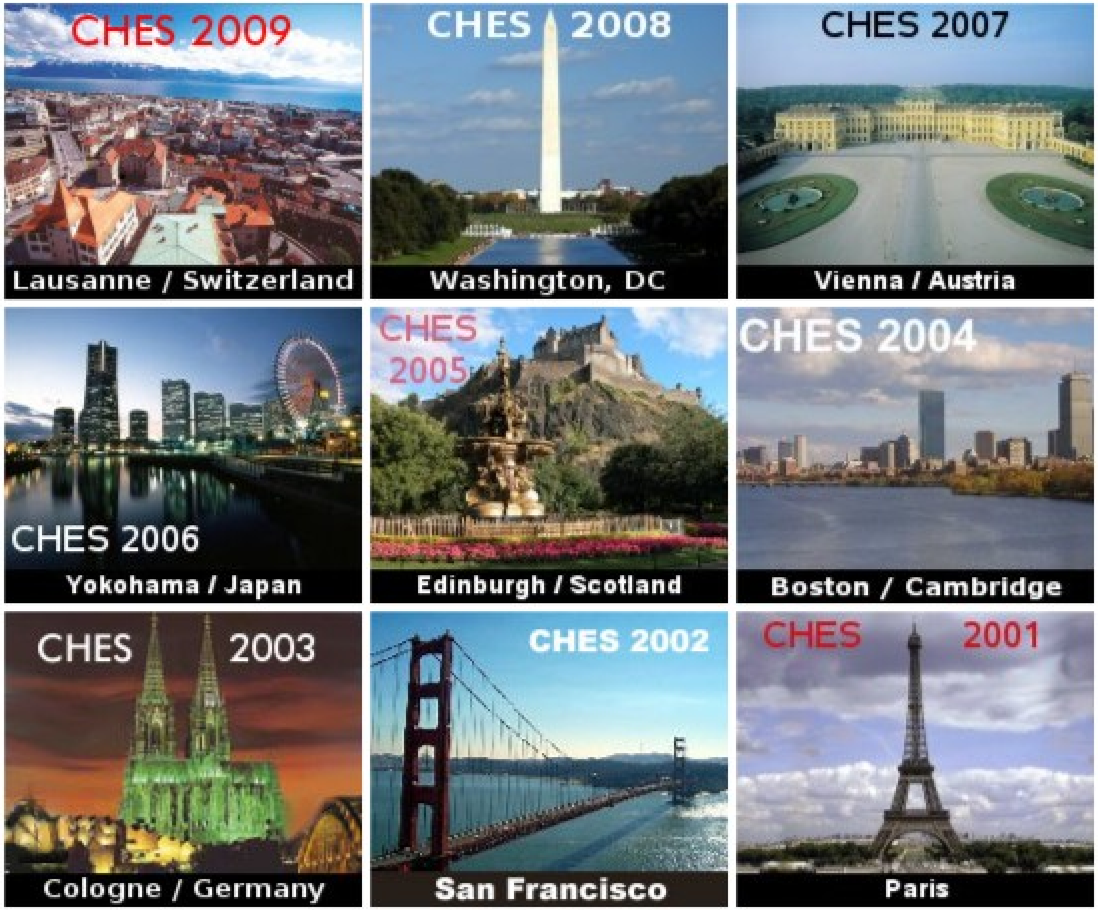
Guessing K_0 = 02, traces where high bit of $S[P_0 \oplus K_0]$ is set

Partition traces by some predicted intermediate bit



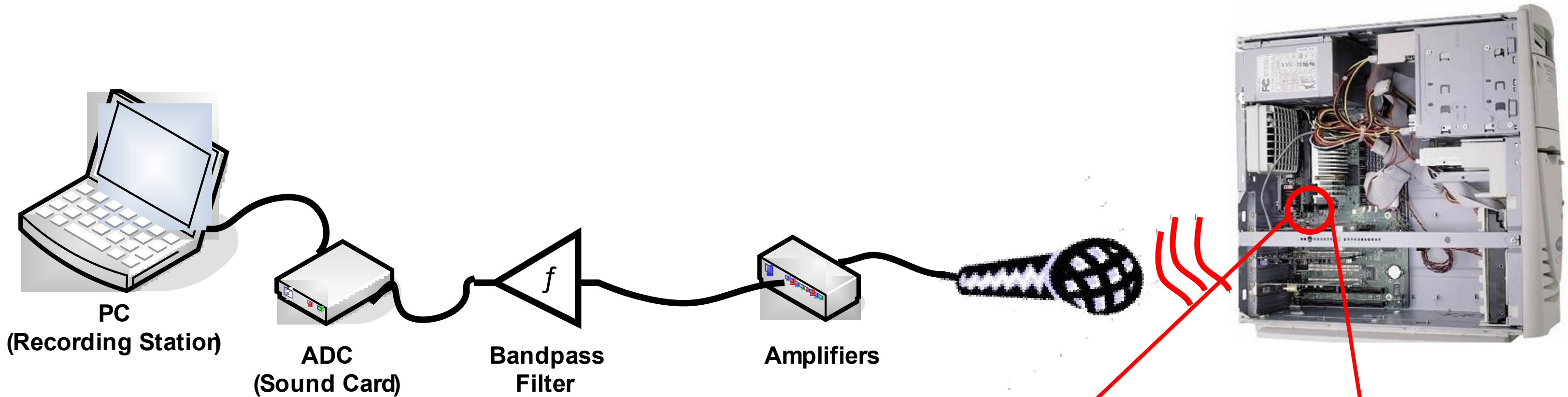
Guessing K_0 = 02, traces where high bit of $S[P_0 \oplus K_0]$ is set

Perfect countermeasures are very difficult

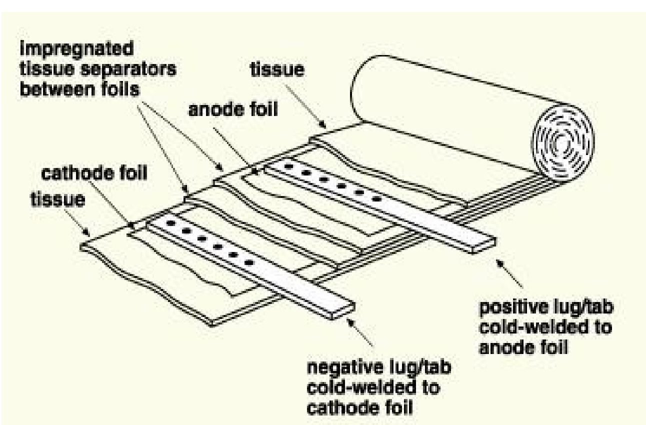


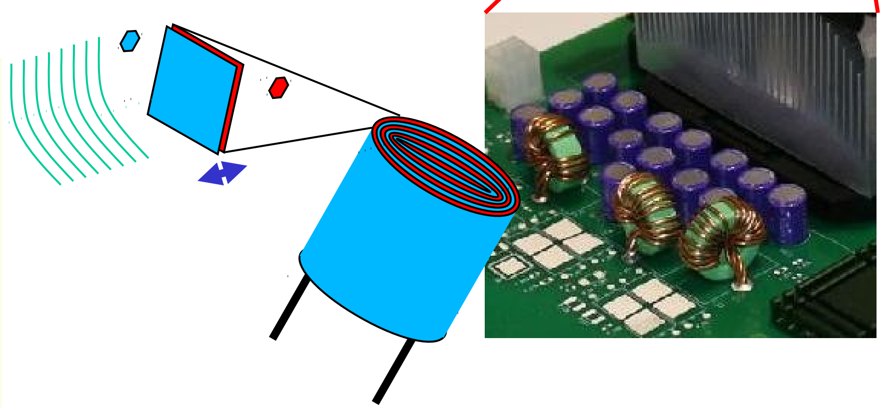
Even further down the rabbit hole...

CPU Noise



Collects sounds which emanates from CPU

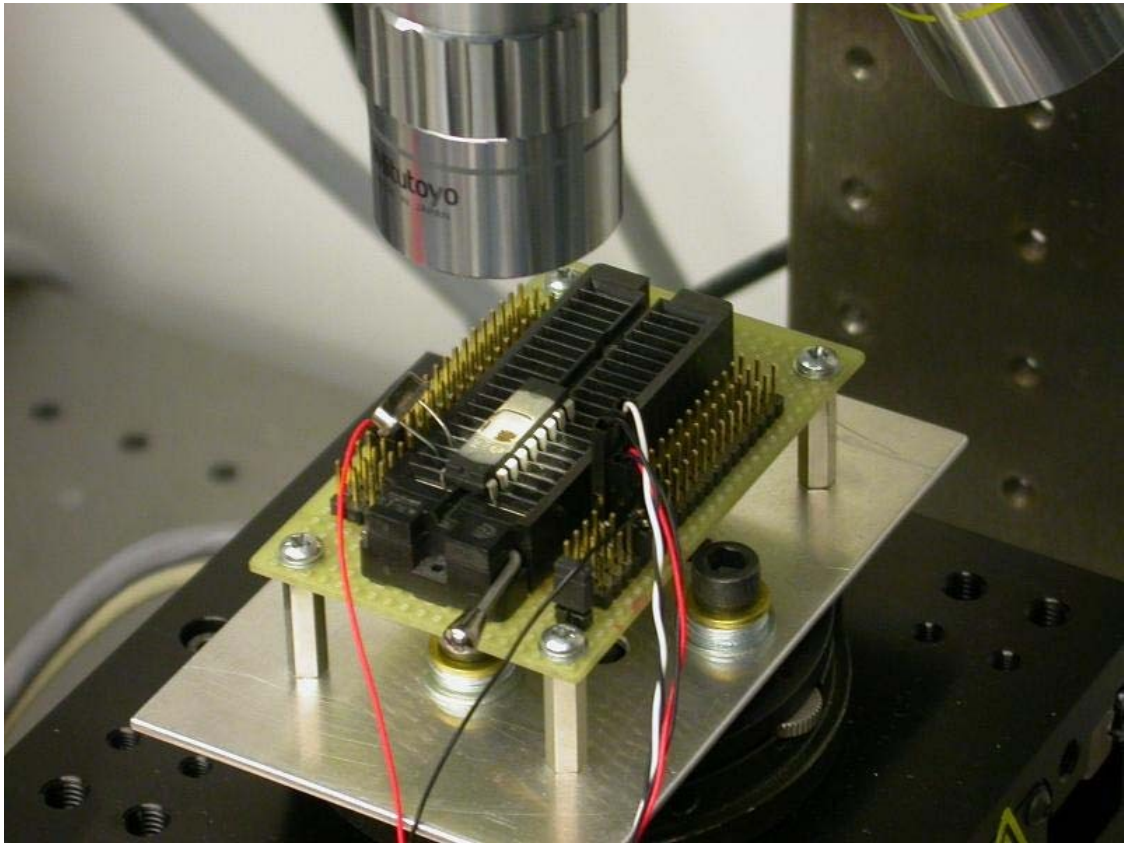


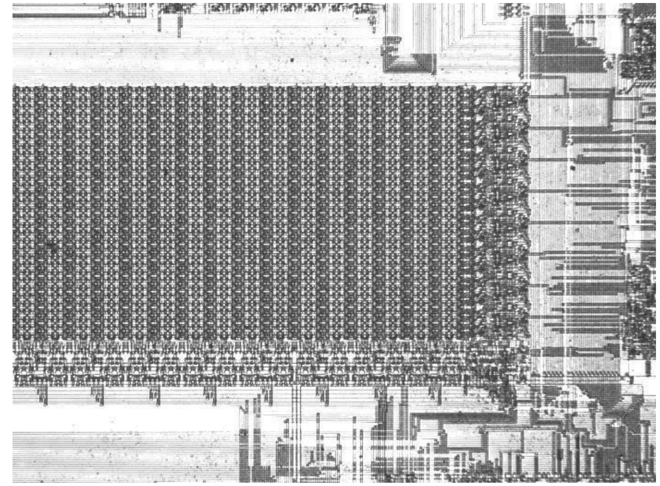


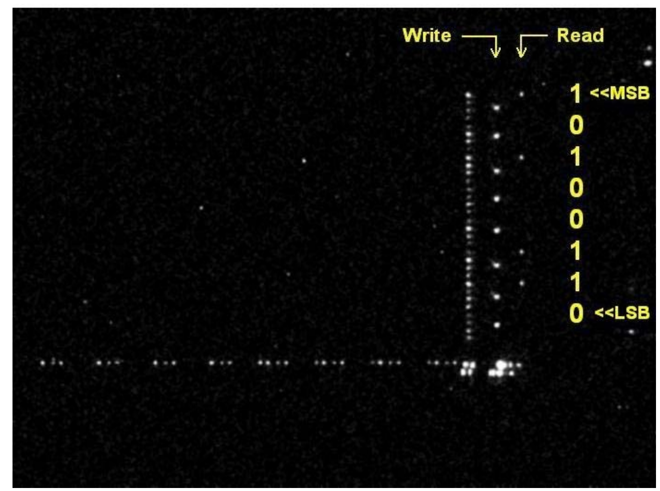
Capacitor emits sound from its foils due
to fast pulse charging n discharging

Adi Purwono, 2008

Photon emissions



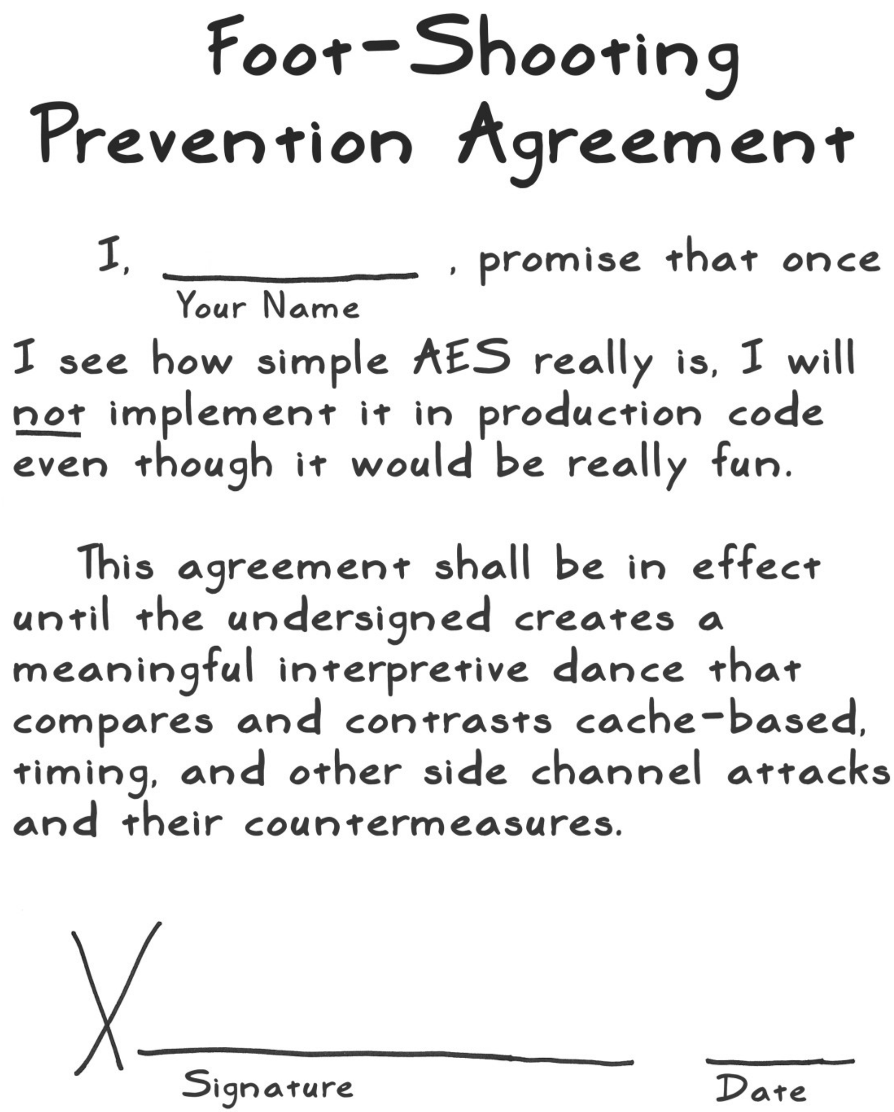




Sergei Skorobogatov, 2009

Lessons

Don't design or implement your own crypto



Foot-Shooting Prevention Agreement

I, ____________, promise that once
Your Name
I see how simple AES really is, I will <u>not</u> implement it in production code even though it would be really fun.

This agreement shall be in effect until the undersigned creates a meaningful interpretive dance that compares and contrasts cache-based, timing, and other side channel attacks and their countermeasures.

X ____________________ ________
Signature Date

Jeff Moser

Do break whatever you can

Algorithms:

- Elliptic curve DSS/DH
- Pairing-based algorithms
- AES-GCM authentication
- SHA-3 candidates

Side-channels:

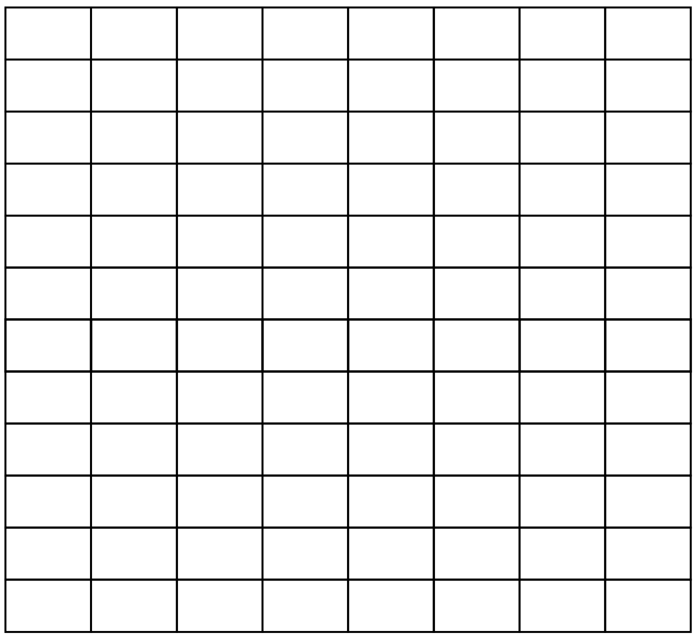
- Motherboard sensors
- CPU debug registers

Killer target:

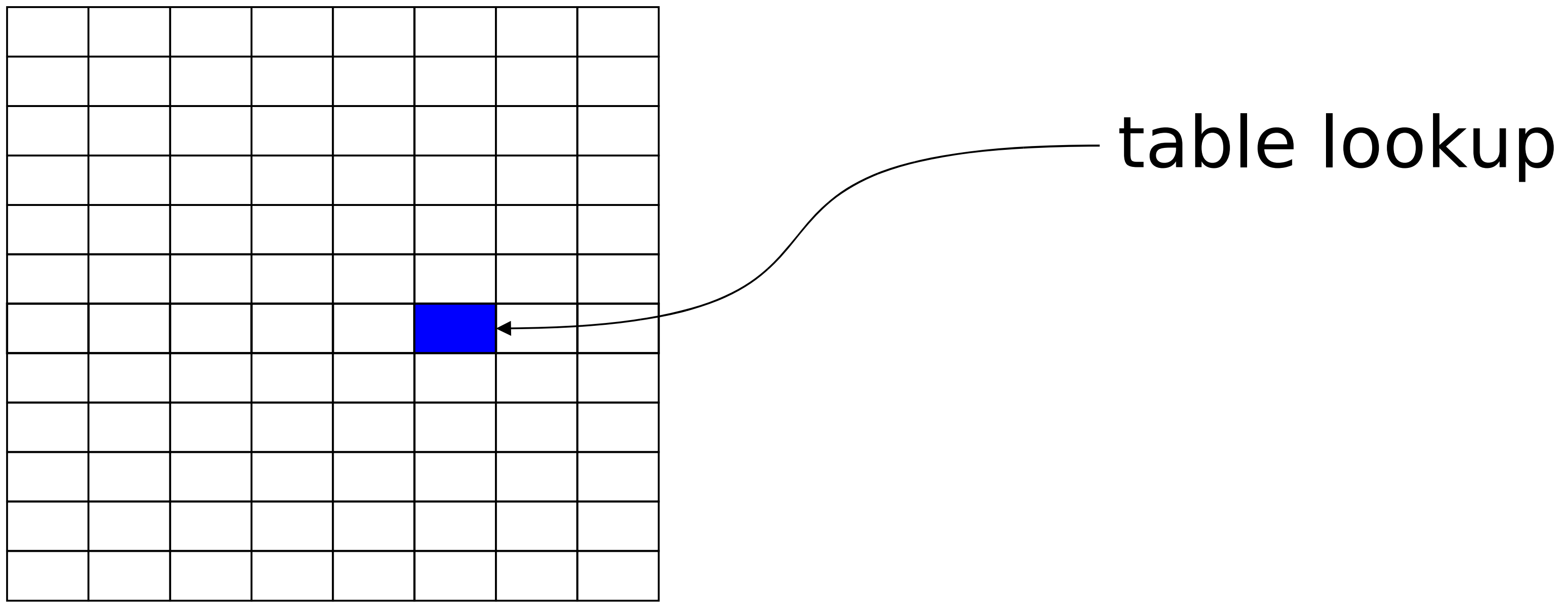
- Cross-VM key compromise

Thank you

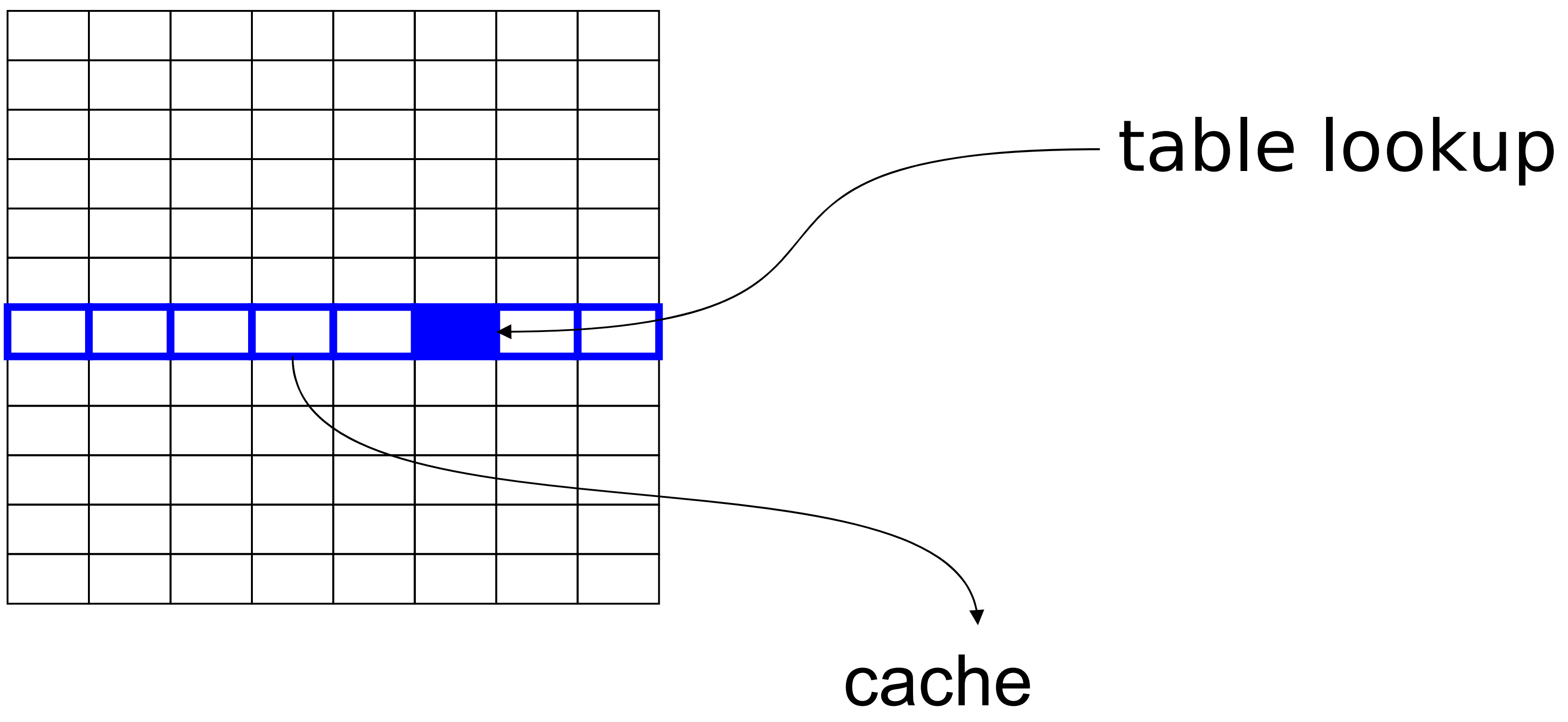
Complication: cache lines



Complication: cache lines



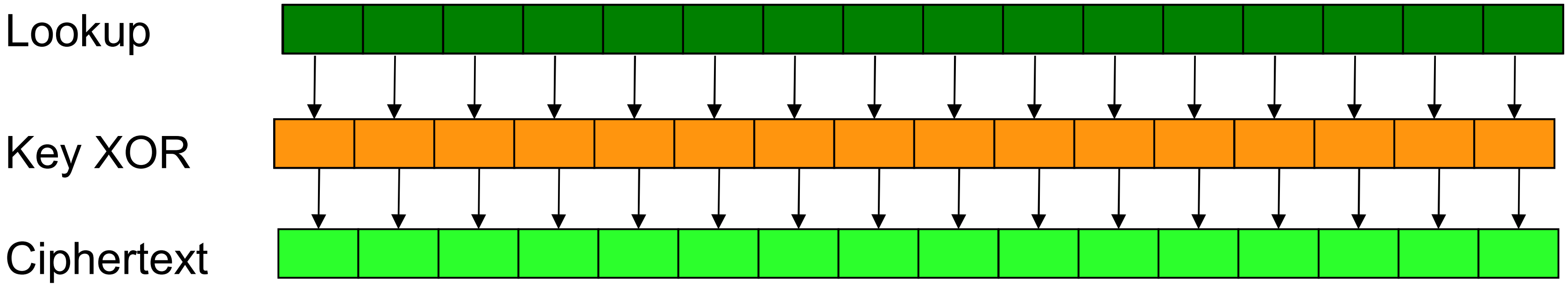
Complication: cache lines



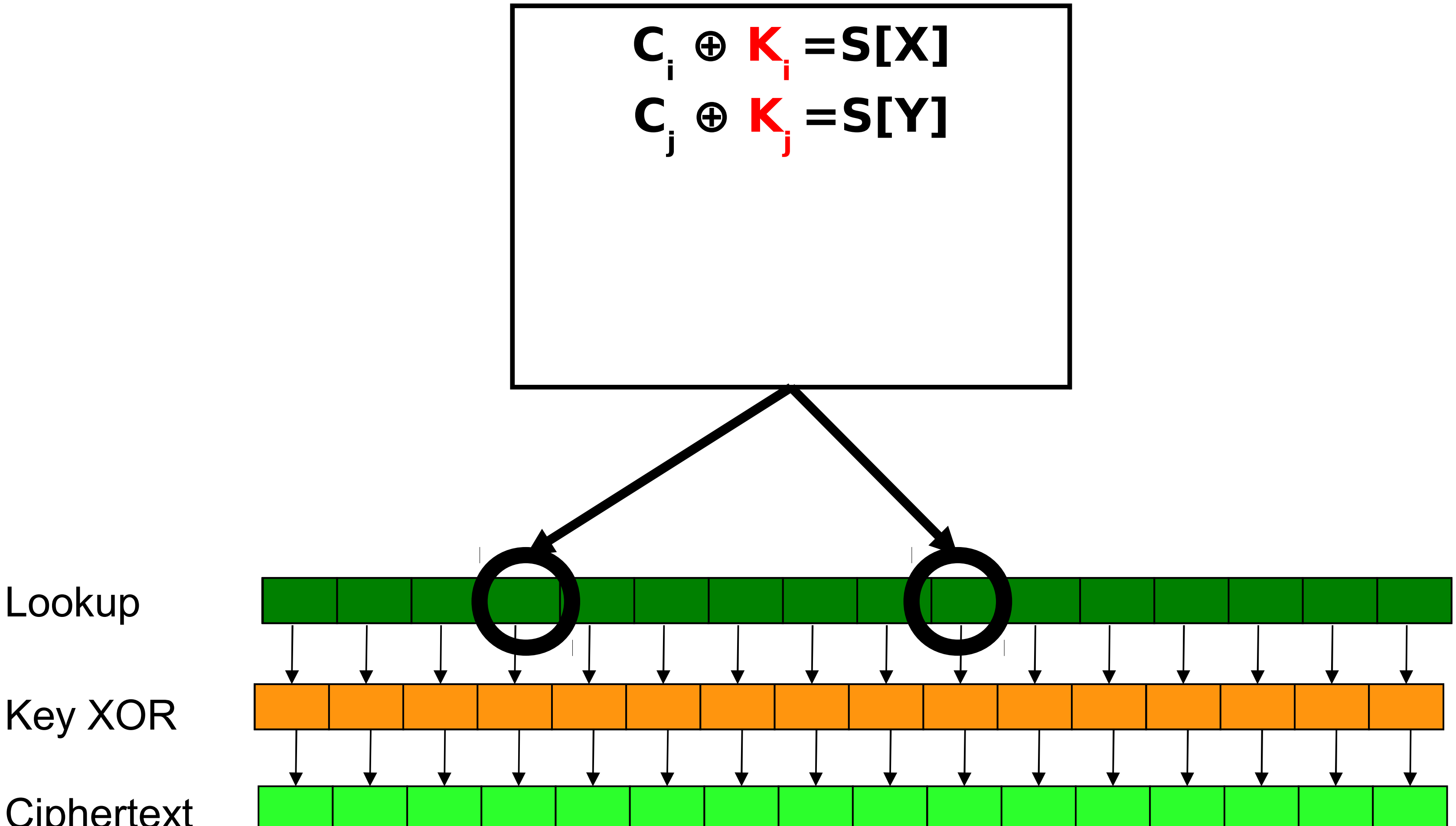
Complication: Table families

```
t0 =   Te0[(s0 >> 24)        ] ^
       Te1[(s1 >> 16) & 0xff] ^
       Te2[(s2 >>  8) & 0xff] ^
       Te3[(s3      ) & 0xff] ^
       rk[0];
t1 =   Te0[(s1 >> 24)        ] ^
       Te1[(s2 >> 16) & 0xff] ^
       Te2[(s3 >>  8) & 0xff] ^
       Te3[(s0      ) & 0xff] ^
       rk[1];
t2 =   Te0[(s2 >> 24)        ] ^
       Te1[(s3 >> 16) & 0xff] ^
       Te2[(s0 >>  8) & 0xff] ^
       Te3[(s1      ) & 0xff] ^
       rk[2];
t3 =   Te0[(s3 >> 24)        ] ^
       Te1[(s0 >> 16) & 0xff] ^
       Te2[(s1 >>  8) & 0xff] ^
       Te3[(s2      ) & 0xff] ^
       rk[3];
```

Final round is much better to attack



Final round is much better to attack



Final round is much better to attack

$$C_i \oplus K_i = S[X]$$

$$C_j \oplus K_j = S[Y]$$

$$X = Y \Rightarrow$$

$$C_i \oplus K_i = C_j \oplus K_j$$

$$C_i \oplus C_j = K_i \oplus K_j$$