Side Channel Analysis

Chester Rebeiro

IIT Madras

Modern ciphers designed with very strong assumptions

- **Kerckhoff's Principle**
  - The system is completely known to the attacker. This includes encryption & decryption algorithms, plaintext
  - only the key is secret

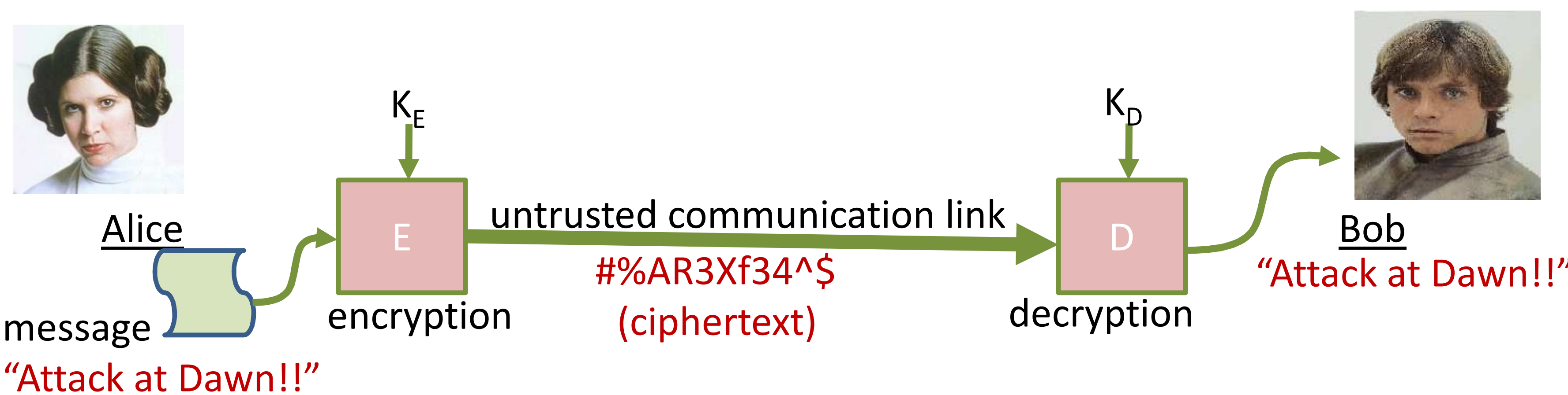
- Why do we make this assumption?
  - Algorithms can be leaked (secrets never remain secret)
  - or reverse engineered

Mallory's task is therefore very difficult….

CR

Security as strong as its weakest link

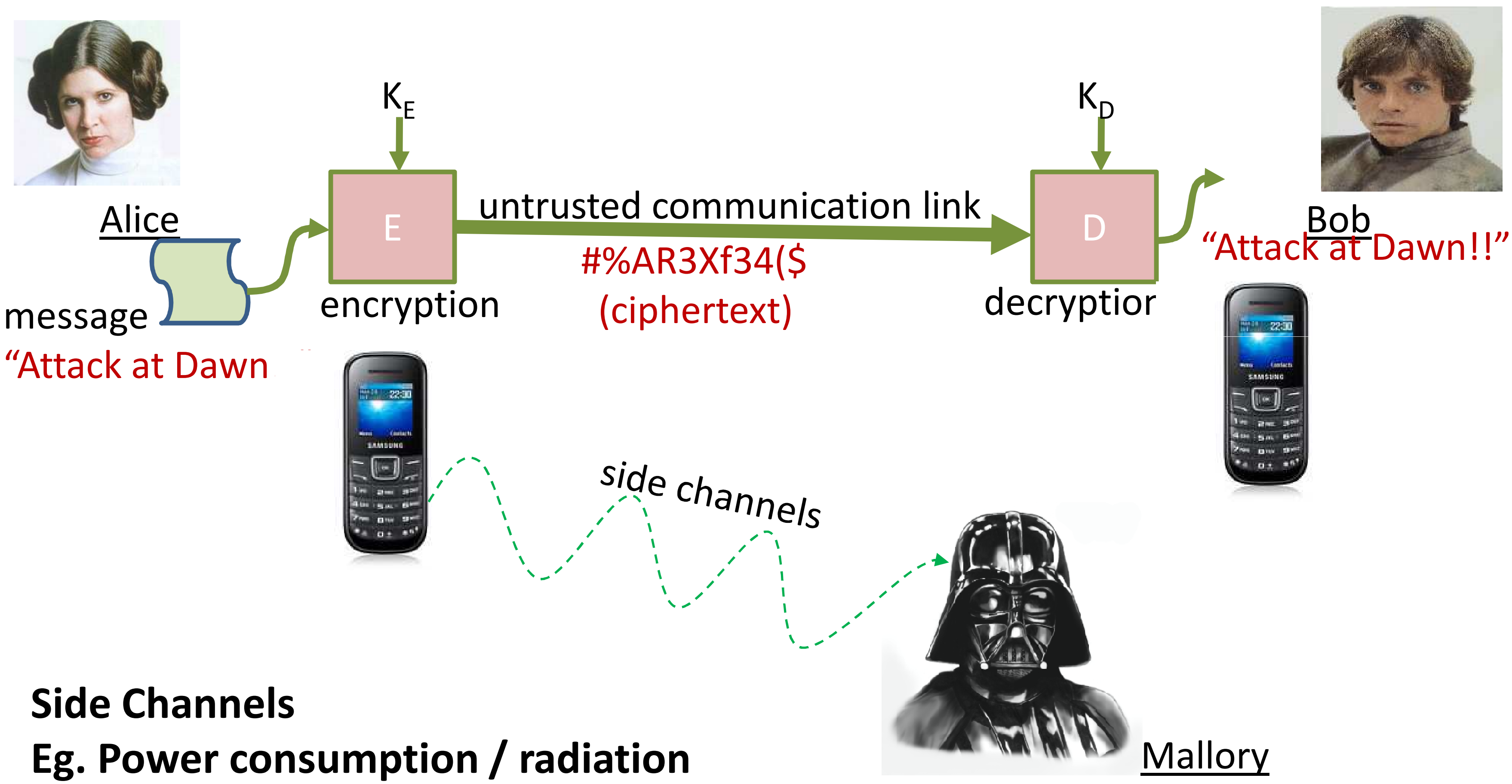
- Mallory just needs to find the weakest link in the system
  ....there is still hope!!!



Side Channels

CR

Side Channel Analysis (the weak links)

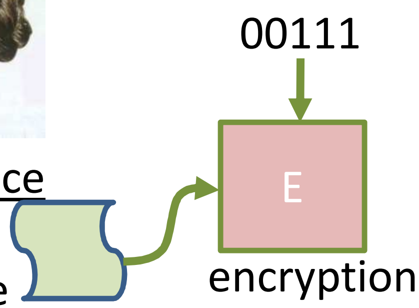


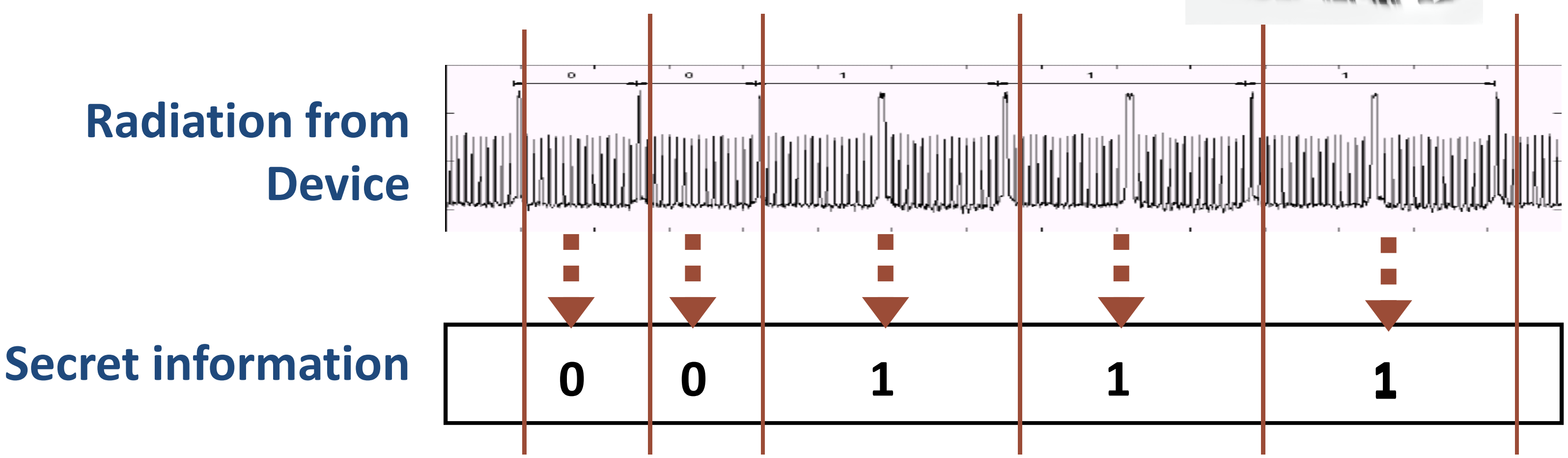
Side Channels
Eg. Power consumption / radiation of device, execution time, etc.

Gets information about the keys by monitoring Side channels of the device

Side Channel Analysis

Mallory measures some
Physical parameter of the device
Like radiation, power consumption or
timing





Types of Side Channel Attacks

	Passive Attacks The device is operated largely or even entirely within its specification	Active Attacks The device, its inputs, and/or its environment are manipulated in order to make the device behave abnormally
Non-Invasive Attacks Device attacked as is, only accessible interfaces exploited, relatively inexpensive	**Side-channel attacks: timing attacks, power + EM attacks, cache trace**	Insert fault in device without depackaging: clock glitches, power glitches, or by changing the temperature
Semi-Invasive Attacks Device is depackaged but no direct electrical contact is made to the chip surface, more expensive	Read out memory of device without probing or using the normal read-out circuits	Induce faults in depackaged devices with e.g. X-rays, electromagnetic fields, or light
Invasive Attacks No limits what is done with the device	Probing depackaged devices but only observe data signals	Depackaged devices are manipulated by probing, laser beams, focused ion beams

CR

source : Elisabeth Oswald, Univ. of Bristol

Timing Attacks

Execution Time

What can you tell from the execution time of this function?

```
unsigned int Divide(unsigned int a, unsigned int b){
  if (b==0)
    return ERROR;
  else
    return a/b;
}
```

Finding N/D

```
while  N ≥ D do
  N := N - D
end
return N
```

- Execution time depends on values of a and b
  - Fastest when b=0
  - Varies depending a / b
- Thus information can be inferred from execution time.
  - Can we get secret information from the timing?

Measuring Time Accurately

- RDTSC : Read Time Stamp Counter
  - 128 bit register that s reset at boot up and increments at every clock cycle

Usage
Flush Pipeline
T1 = rdtsc()
Flush Pipeline
/// invoke function to be timed
T2 = rdtsc()
Flush pipeline

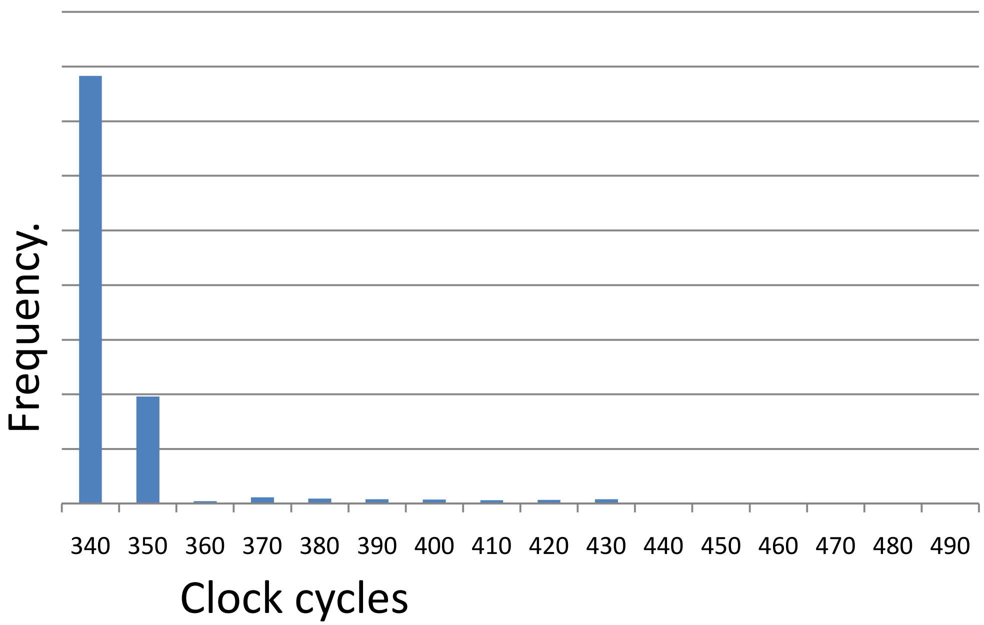
Flush Pipeline and Read TSC

timestamp()

```
cpuid               ; ensure preceding instructions complete
rdtsc               ; read time stamp
cpuid               ; ensure preceding instructions complete
mov  time, eax      ; move counter into variable
load ebx, (ebp)     ; a load from memory
cpuid               ; ensure preceding instructions complete
rdtsc               ; read time stamp again
cpuid               ; ensure preceding instructions complete
sub  eax, time      ; find the difference
```

http://arbidprobramming.blogspot.in/2010/05/measuring-timing-accurately-on-intel.html

DIV: Measuring Execution Time




```
unsigned int Divide(unsigned int a, unsigned int b){
  if (b == 0)
    return ERROR;
  else
    return a/b;
}
```

- For randomly chosen values of a/b
- Note the distribution

Timing Attacks on RSA
(breaking real-world implementations)

Timing Attacks on Implementations of Diffe-Hellman, RSA, DSS, and other systems
http://courses.csail.mit.edu/6.857/2006/handouts/TimingAttacks.pdf

Remote Timing Attacks are Practical
https://crypto.stanford.edu/~dabo/papers/ssl-timing.pdf

Exponentiation with Square and Multiply

$$y = x^c \bmod n$$

- say, x=45=$(101101)_2$

i	c	exp
5	1	y
4	0	y^2
3	1	$y^{4+1}=y^5$
2	1	$y^{10+1}=y^{11}$
1	0	Y^{22}
0	1	$y^{44+1}=y^{45}$

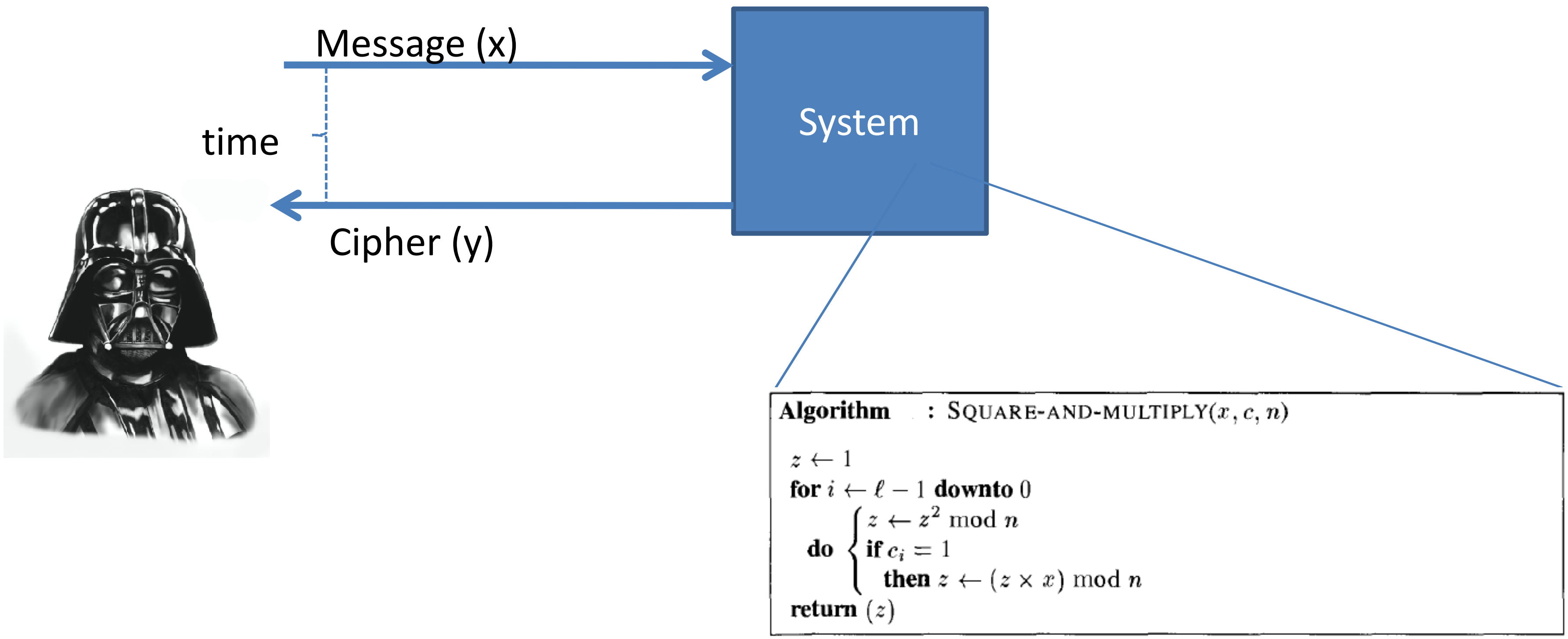
Algorithm : SQUARE-AND-MULTIPLY(x, c, n)

$z \leftarrow 1$
for $i \leftarrow \ell - 1$ **downto** 0
do {
  $z \leftarrow z^2 \bmod n$
  if $c_i = 1$
    then $z \leftarrow (z \times x) \bmod n$
}
return (z)

CR

The Attack setup

$$y = x^c \bmod n$$



Timing Attacks on Implementations of Diffe-Hellman, RSA, DSS, and other systems
http://courses.csail.mit.edu/6.857/2006/handouts/TimingAttacks.pdf

Kocher's Attack to find the bth bit

Assumption : Attacker knows bits $c_{l-1}, c_{l-2} \cdots c_{b+1}$

Aim : To discover bit c_b

$S1.$ choose a random x

$S2.$ trigger an encryption to get $y \equiv x^c \bmod n$ and execution time t

S3. form $c^{(0)} = (c_{l-1}, c_{l-2} \cdots, c_{b+1}, 0, 0)$ **Guess 0**

trigger an encryption to get $y \equiv x^{c^{(0)}} \bmod n$ and execution time $t^{(0)}$

S4. form $c^{(1)} = (c_{l-1}, c_{l-2} \cdots, c_{b+1}, 1, 0)$ **Guess 1**

trigger an encryption to get $y \equiv x^{c^{(1)}} \bmod n$ and execution time $t^{(1)}$

$S5.$ compute difference in execution time

$$d^{(0)} = t - t_0 \qquad\qquad d^{(1)} = t - t_1$$

$S6.$ Repeat from $S1$ several times

$S7.$ Compute distributions of $D^{(0)}$ from all $d^{(0)}$ and $D^{(1)}$ from all $d^{(1)}$

$S8.$ *If* $\operatorname{var}(D^{(0)}) < \operatorname{var}(D^{(1)})$ *return* '$c_b = 0$'

else *return* '$c_b = 1$'

Adding Distributions

- Consider two random variables G_1 and G_2 with mean and variance (m_1, v_1) and (m_2, v_2)
- $G_1 + G_2$ is a distribution with mean and variance (m_1+m_2, v_1+v_2)
- $G_1 - G_2$ is a distribution with mean and variance $(m_1 - m_2, v_1+v_2)$

Assumption

- During the square and multiply execution,
- The time taken to perform a square or a multiply is independent of all other square and multiply operations

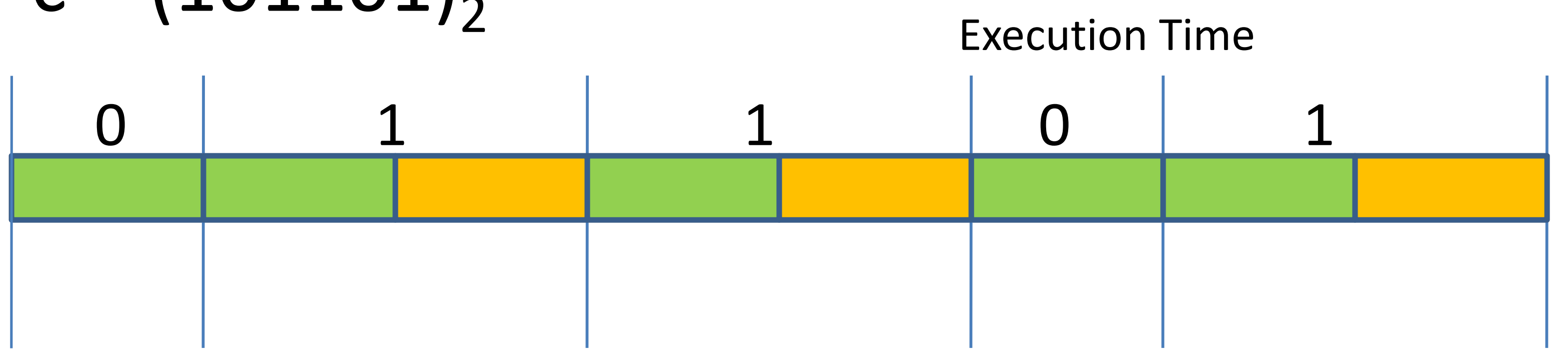
Algorithm : SQUARE-AND-MULTIPLY(x, c, n)

$z \leftarrow 1$
for $i \leftarrow \ell - 1$ **downto** 0
  do $\begin{cases} z \leftarrow z^2 \bmod n \\ \textbf{if } c_i = 1 \\ \quad \textbf{then } z \leftarrow (z \times x) \bmod n \end{cases}$
return (z)

Execution Time of Square and Multiply

- Is a Normal Distribution : T with (m, v)
- Each iteration by itself is a distribution

$c = (101101)_2$



$$T = 3T_{MUL} + 5T_{SQ}$$

$$v = 3v_{MUL} + 5v_{SQ}$$

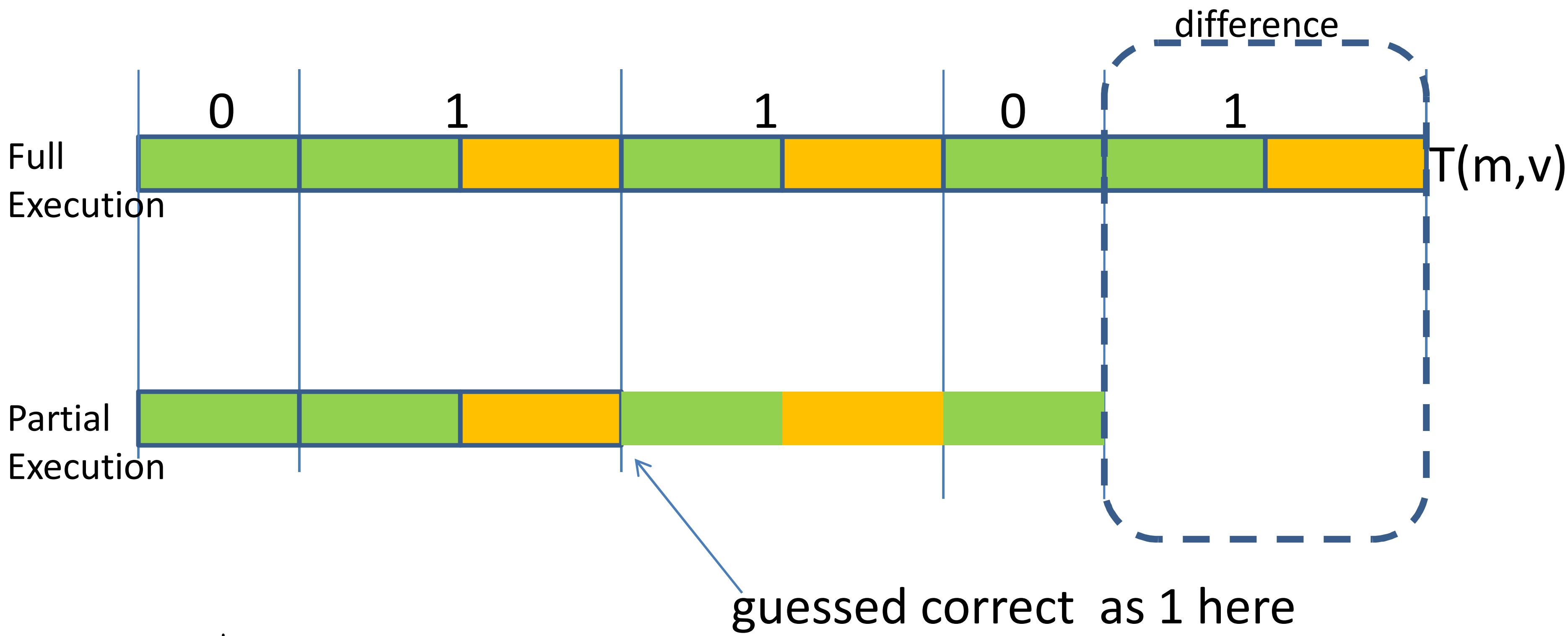
4 cases

- Bit c_b in secret is 1
  - Attacker guessed 1 (correctly)
  - Attacker guessed 0 (wrong)
- Bit c_b in secret is 0
  - Attacker guessed 0 (correctly)
  - Attacker guessed 1 (wrong)

what we will see is that when the attacker guess is wrong, then the variance is higher

Case 1.1, when bit c_b is 1

and attacker guess is correct

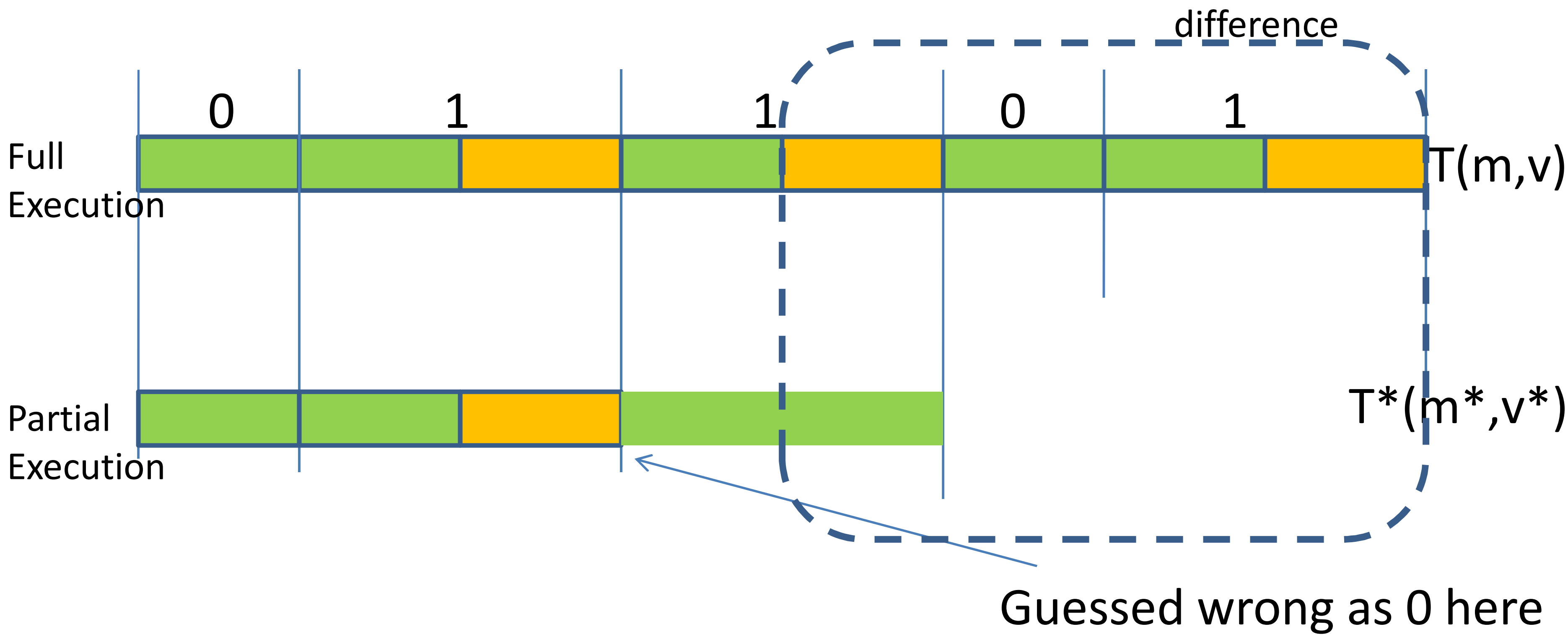


$$v - v^* = 1v_{MUL} + 1v_{SQ}$$

Variance Reduces

Case 1.2, When c_b bit is 1

And attacker guess is wrong

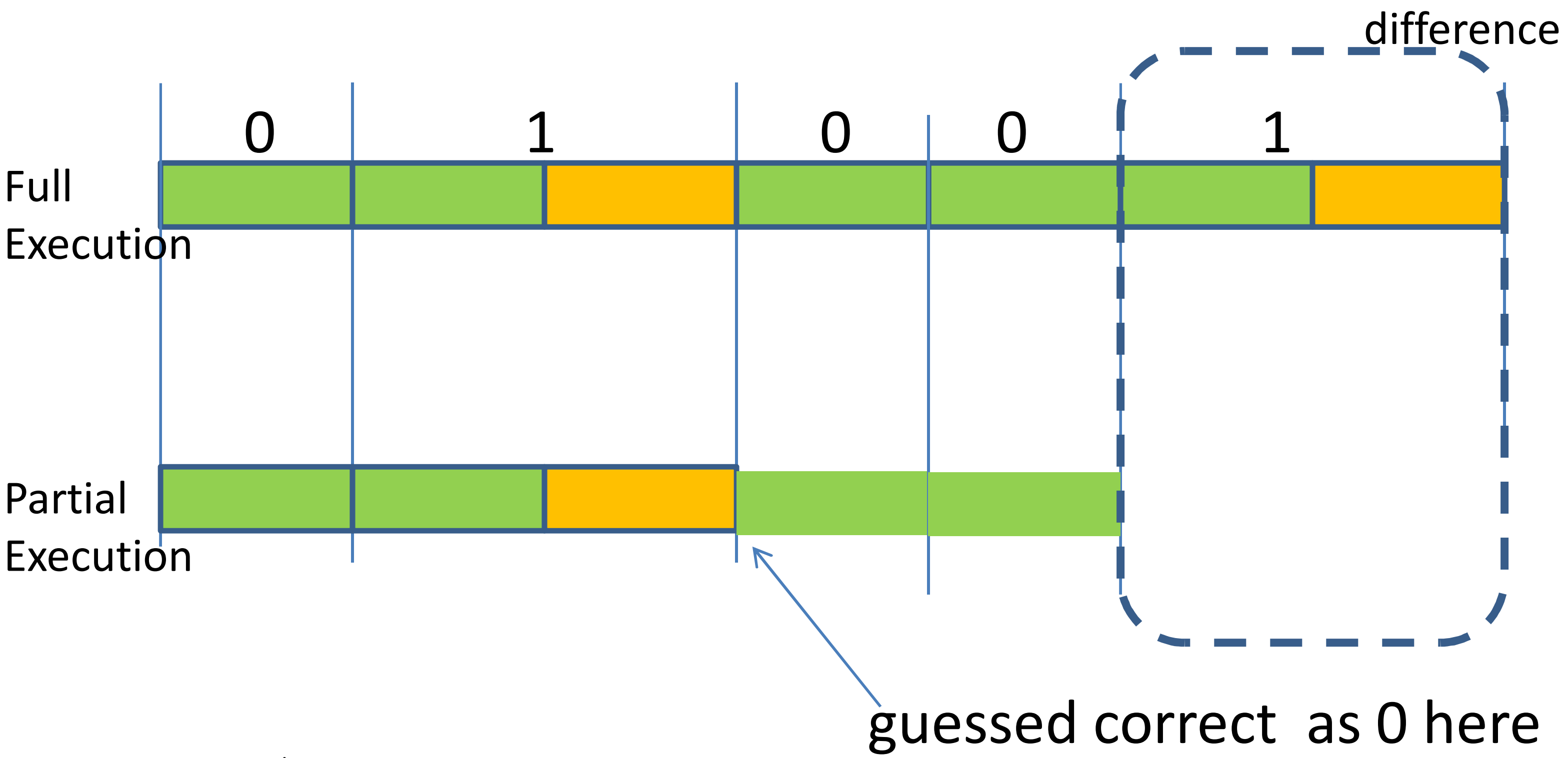


$$v - v^* = 2v_{MUL} + 3v_{SQ} > (v_{MUL} + v_{SQ})$$

Variance Increases

Case 2.1, when c_b is 0

And attacker guess is correct



$$v - v^* = 1v_{MUL} + 1v_{SQ}$$

Variance Less

Case 2.2, When c_b is 0

$$v - v^* = 2v_{MUL} + 3v_{SQ}$$

Variance increases

The Iterative Attack

- We start with the MSB and target one bit at a time till we reach the LSB

What happens if there is an error in a bit?

Naïve Countermeasures don't always work

All operations constant time

Easier said than done!

Practically infeasible

Highly dependent on system architecture

Naïve Countermeasures don't always work

Adding noise to timing measurements

- Such as, by random delays

These reduce the Signal-to-noise ratio.

Can be circumvented by taking making more number of measurements

If the SNR reduces by a factor of n, then number of measurements increase by a factor of n^2

CR

Prevention by Blinding

$$choose \quad r \ randomly \ and \ keep \ it \ \sec ret$$

$$compute \quad r^c \bmod n \qquad and \quad r^{-c} \equiv r^c \bmod n$$

$$y' \equiv (x \cdot r)^c \bmod n$$

$$y \equiv y' \cdot r^{-c} \bmod n$$

The blind ‘r’ should be changed before each decryption.
One way is to choose r and compute r^2.
For the next encryption compute r^2 and $(r^{-1})^2$

Why does it work?
Since ‘r’ is secret, attackers have no useful knowledge about the input to the modular exponentiatior.

RSA Decryption in Practice (OpenSSL crypto-lib uses CRT)

1. $x_1 \equiv y^{a_1} \bmod p$
2. $x_2 \equiv y^{a_2} \bmod q$

$$\begin{bmatrix} x_1 \equiv y^{a_1} \bmod p \\ x_2 \equiv y^{a_2} \bmod q \end{bmatrix} \langle = \rangle \quad x \equiv y^{a} \bmod n$$

where

$a_1 \equiv a \bmod \phi(p)$

$a_2 \equiv a \bmod \phi(q).$

Derive x from x_1 and x_2

compute $q' \equiv q^{-1} \bmod p$

3. $h = q'(x_2 - x_1) \bmod p$

$x = x_1 + h \cdot q$

xis the message
y is the ciphertext
a is the secret key
n = pq

Garner's formula.

$$x = (x_1 \cdot p \cdot p^{-1} \bmod q + x_2 \cdot q \cdot q^{-1} \bmod p) \bmod n$$

from EEA, $p \cdot p^{-1} \bmod q + q \cdot q^{-1} \bmod p = 1$

$$p \cdot p^{-1} \bmod q = 1 - q \cdot q^{-1} \bmod p$$

$$x = x_1 + (x_2 - x_1) q \cdot q^{-1} \bmod p$$

Crypto libraries like the OpenSSL implement multiplication using the Montgomery multiplication

Preventing Kocher's Attack with the Montgomery Ladder

- $s = y^c \bmod n$

  say, $c = 45 = (101101)_2$

i	c_i	R0	R1
		1	y
0	1	y	Y^2
1	0	Y^2	Y^3
2	1	Y^5	Y^6
3	1	Y^{11}	Y^{12}
4	0	Y^{22}	Y^{23}
5	1	Y^{45}	Y^{46}

```
Input: c, y
Output: yᶜ mod n

exp(x,y){
  R0 = 1
  R1 = y
  for i=0 to n-1 do
      if xi = 0 then
              R1 = R0 * R1 mod N
              R0 = R0 * R0 mod N
      else
              R0 = R0 * R1 mod N
              R1 = R1 * R1 mod N
  return R0
}
```

$c_b=0$ and $c_b=1$ take the same time

Modular multiplications done with Montgomery multiplier

Montgomery Multiplication

- Montgomery multiplication changes **mod q** operations to **mod 2^k**
  - This is much faster (since mod 2^k is achieved taking the last k bits)
- Computing **c ≡ a*b mod q** using Montgomery multiplication
  1. For the given q, select **$R=2^k$** such **(R > q)** and **gcd(R,q) = 1**
  2. Using Extended Euclidean Algorithm find two integers to compute R^{-1} and q' such that **$R.R^{-1} - q.q' = 1$**
  3. Convert multiplicands to their Montgomery domain:

     A ≡ aR mod q   **B ≡ bR mod q**
  4. Compute abR mod N using the following steps

```
S = A * B
S = S + (S * q' mod R) * q / R
If (S > q)
   S = S – q
return S
```

Requires 3 integer multiplications

  5. Perform **$S*R^{-1}$ mod q** to obtain **ab mod q**

CR

http://www.hackersdelight.org/MontgomeryMultiplication.pdf

Montgomery Multiplier in the Montgomery Ladder

```
Input: c, y
Output: y^c mod N

exp(c,y){
  R0 = 1 * R mod N
  R1 = y * R mod N
  for i=0 to n-1 do
     if ci = 0 then
            R1 = R0 * R1
            R0 = R0 * R0
     else
            R0 = R0 * R1
            R1 = R1 * R1

   return (R0 * R^-1)
}
```

Convert to Montgomery domain.

Multiplications in Montgomery domain. Note. Each result is also in Montgomery domain.

Return to Original domain

The final 'if' in Montgomery Multiplication

- Observation

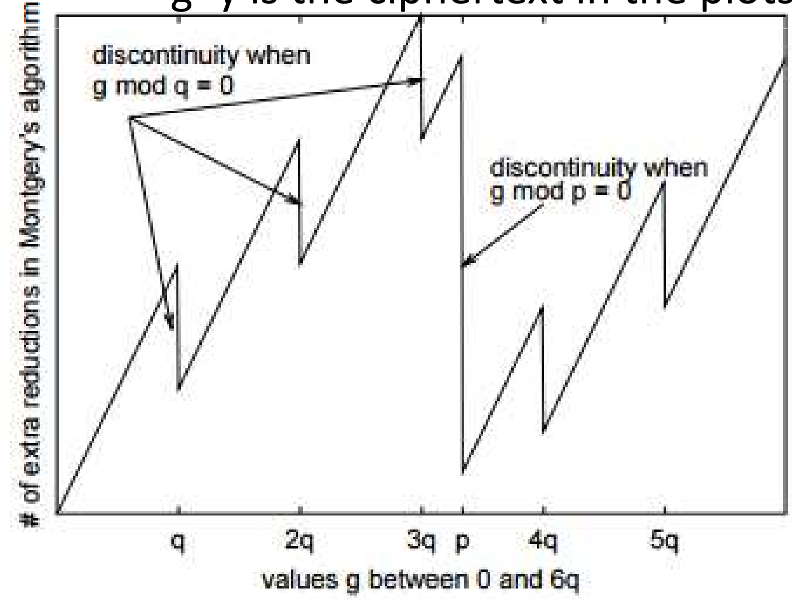
$$\Pr[\text{ExtraReduction}] = \frac{y \bmod q}{2R}$$

Extra reduction step

S = (A * B) R^{-1} mod q
If (S > q) then S = S - q

  - Consider y to be an integer increasing in value
  - As y approaches q, Pr[ExtraReduction] increases
  - When y is a multiple of q, Pr[ExtraReduction] drops
  - Extra reductions causes execution time to increase

g=y is the ciphertext in the plots



Another timing variation due to Integer multiplications

- 30-40% of OpenSSL RSA decryption execution time is spent on integer multiplication
- If multiplicands have the same number of words n, OpenSSL uses Karatsuba multiplication $O(n^{\log_2 3})$
- If integers have unequal number of words n and m, OpenSSL uses normal multiplication $O(nm)$

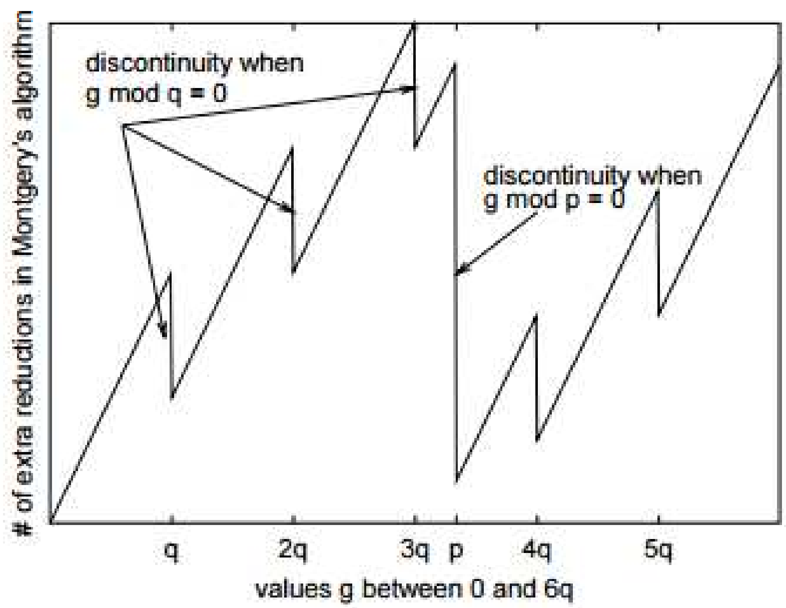
these further cause timing variations…

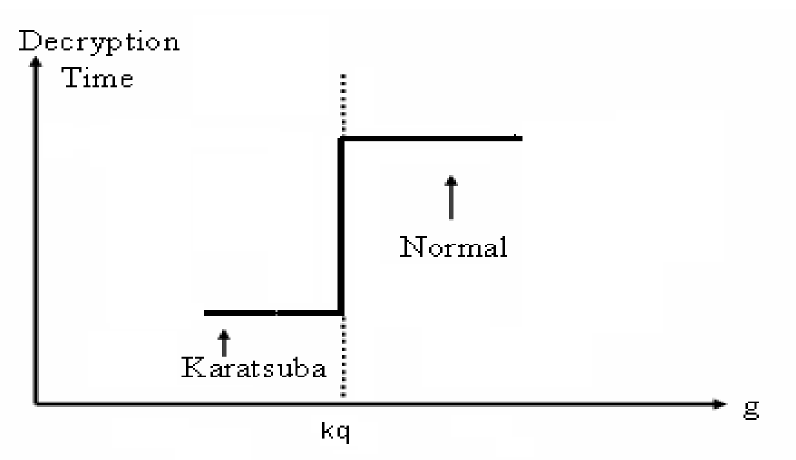
CR

Summary of Timing Variations

	y < q	y > q
Montgomery Effect	Longer	Shorter
Multiplication Effect	Shorter	Longer

Opposite effects, but one will always dominate





Retrieving a bit of q

Assume the attacker has the top i-1 bits of q,

High level attack to get the i^{th} bit of q

1. $Set\ \ y_0 = (q_{l-1}, q_{l-2}, q_{l-3}, \cdots q_{l-i-1}, 0,0,0,\cdots)$

   $Set\ \ y_1 = (q_{l-1}, q_{l-2}, q_{l-3}, \cdots q_{l-i-1}, 1,0,0,\cdots)$

   $note\ that$

   $if\ \ q_i = 0, \quad y_0 \le q < y_1$

   $if\ \ q_i = 1, \quad y_0 < y_1 \le q$

2. $Sample\ \ decryption\ time\ \ for\ \ y_0\ \ and\ \ y_1$

   $t_0\ :\ \ DecryptionTime(y_0)$

   $t_1\ :\ \ DecryptionTime(y_1)$

3. $If\ \ |t_1 - t_0|\ \ \text{is large} \ \ \rightarrow\ q_i = 0 \quad (corresponds\ to\ \ y_0 \le q < y_1)$

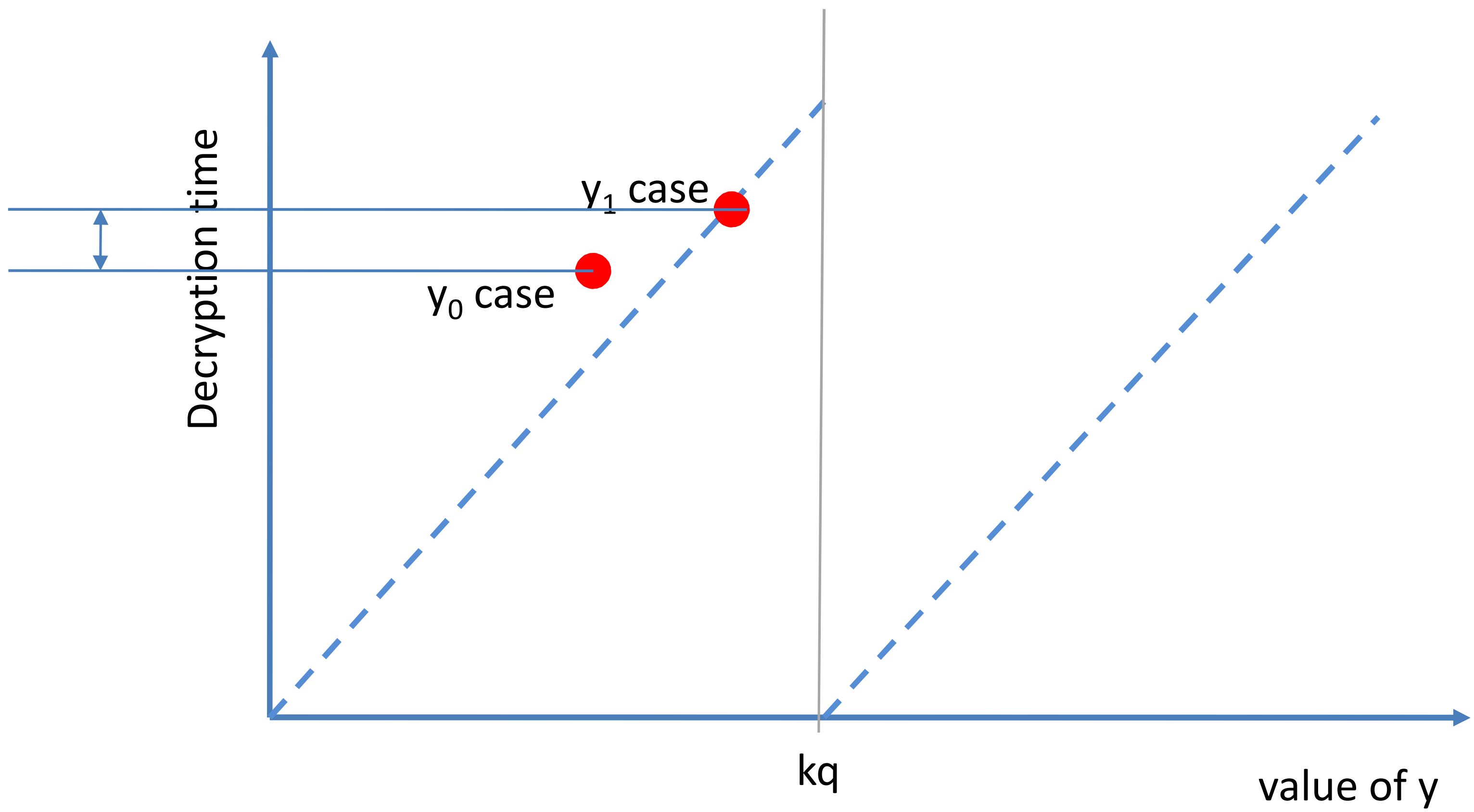
   $\qquad else\ \ q_i = 1 \qquad (corresponds\ to\ y_0 < y_1 \le q)$

CR

What's happening here?

Assume Montgomery multiplier dominates over Integer multiplication

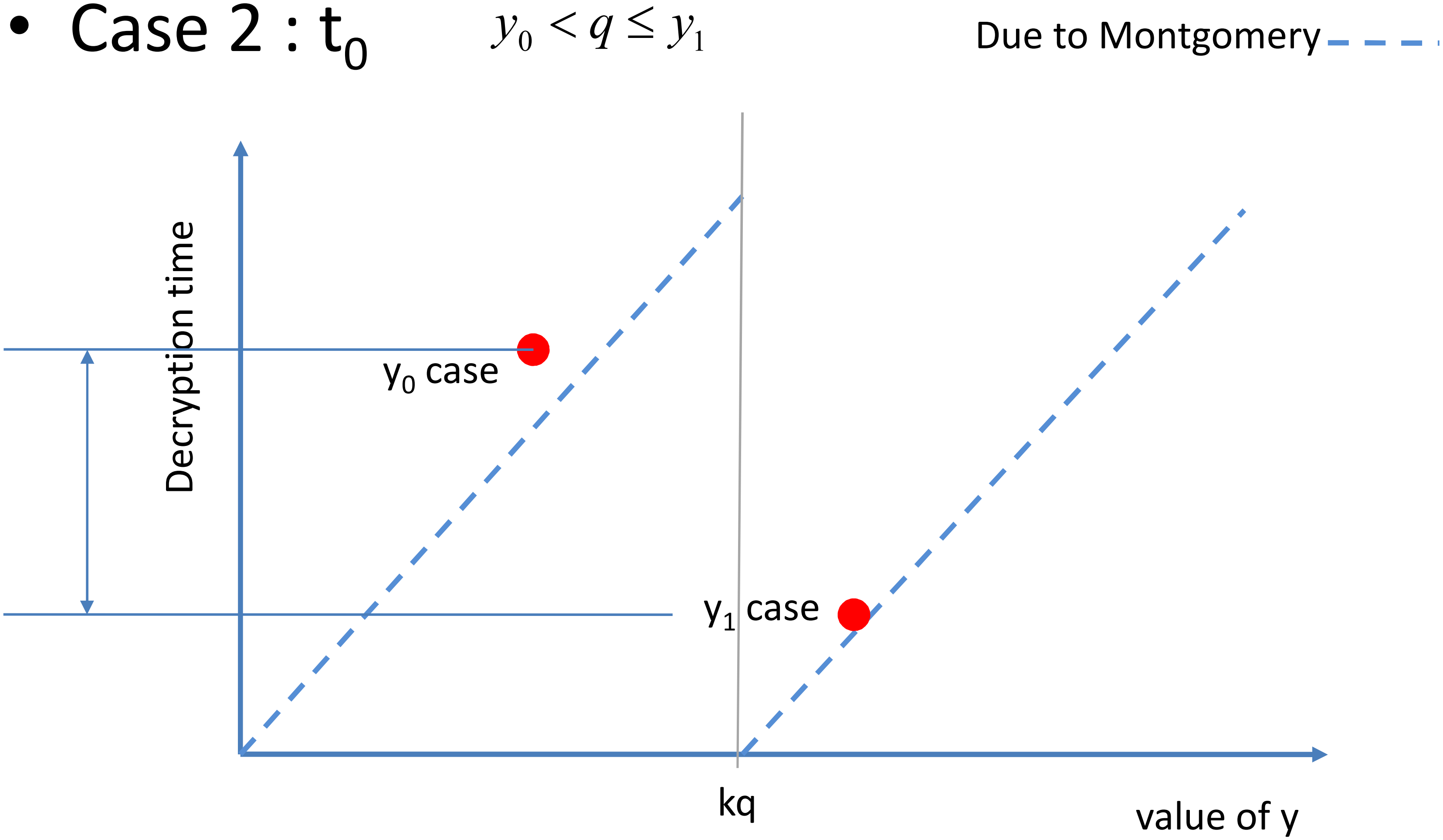
- Case 1 : t_1 $y_0 < y_1 \le q$



What's happening here?

Assume Montgomery multiplier dominates over Integer multiplication

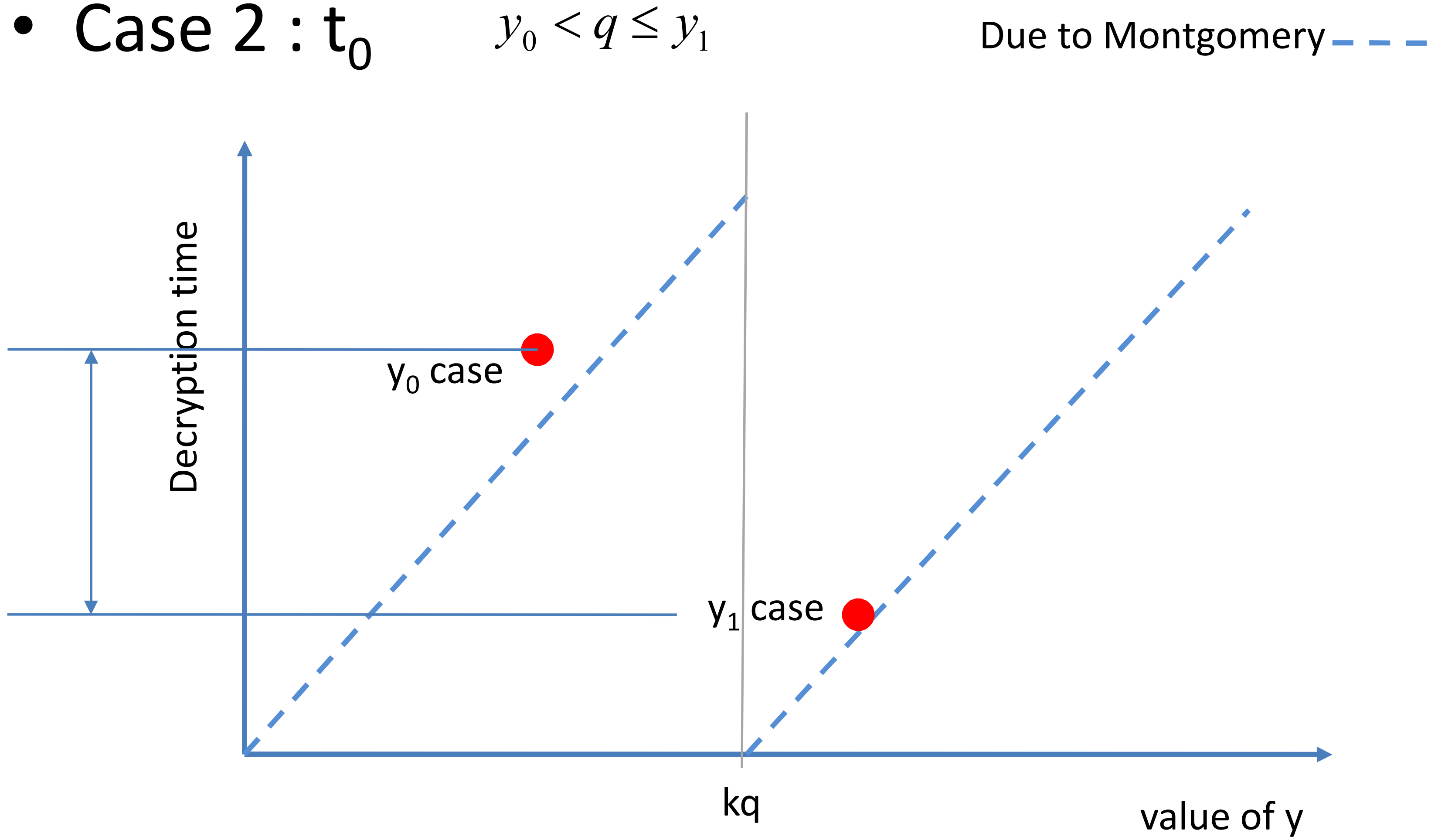
- Case 2 : t_0 $y_0 < q \le y_1$



What's happening here?

Assume Montgomery multiplier dominates over Integer multiplication

- Case 2 : t_0 $y_0 < q \le y_1$

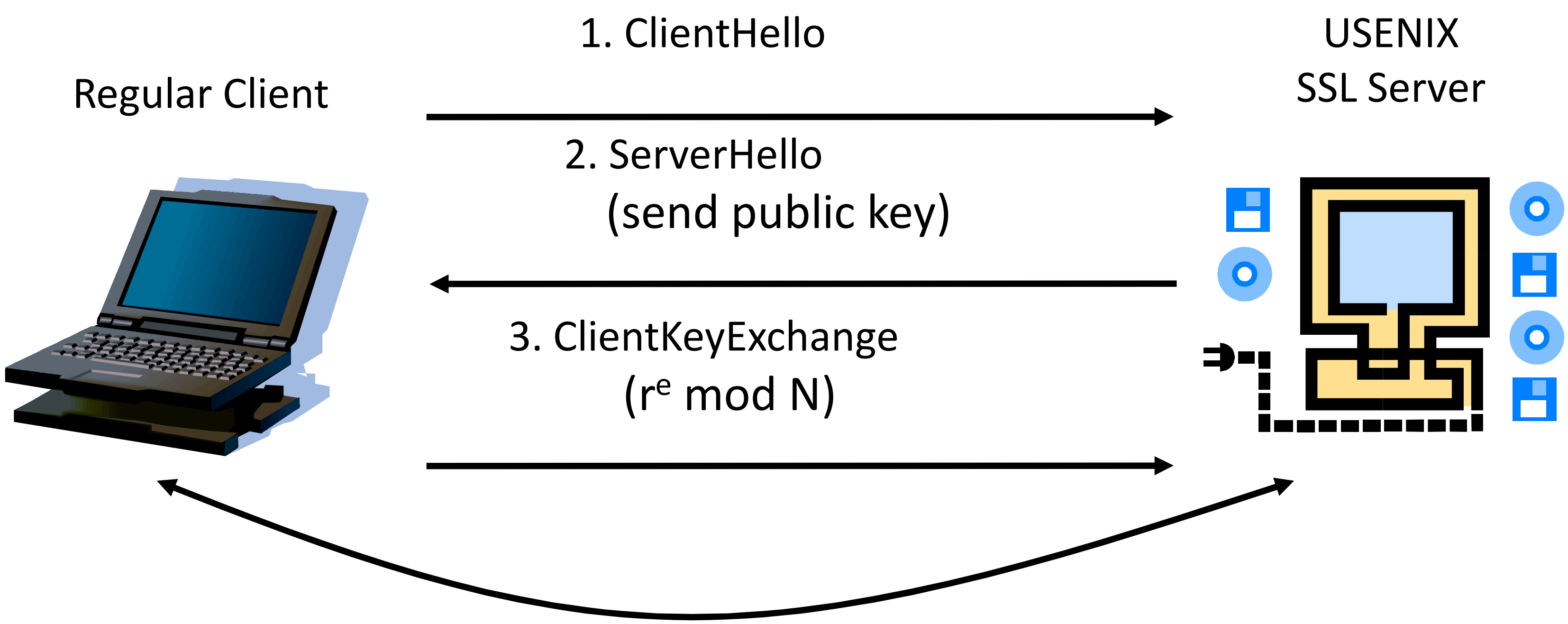


What happens when integer multiplier dominates or Montgomery multiplier?

How does this work with SSL?

How do we get the server to decrypt our y?

Normal SSL Session Startup



Result: Encrypted with computed shared master secret

Attacking Session Startup

Attack Client

1. ClientHello

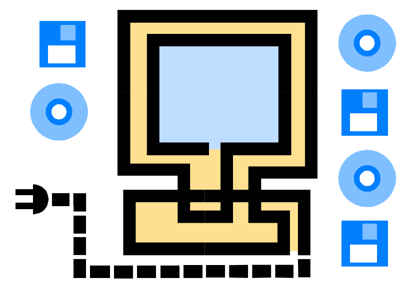
2. ServerHello
   (send public key)

3. Record time t_{start}
   Send guess y_0 or y_1

4. Alert

5. Record time t_{end}
   Compute $t_{start} - t_{end}$

USENIX
SSL Server



Timing Attacks on Block Ciphers

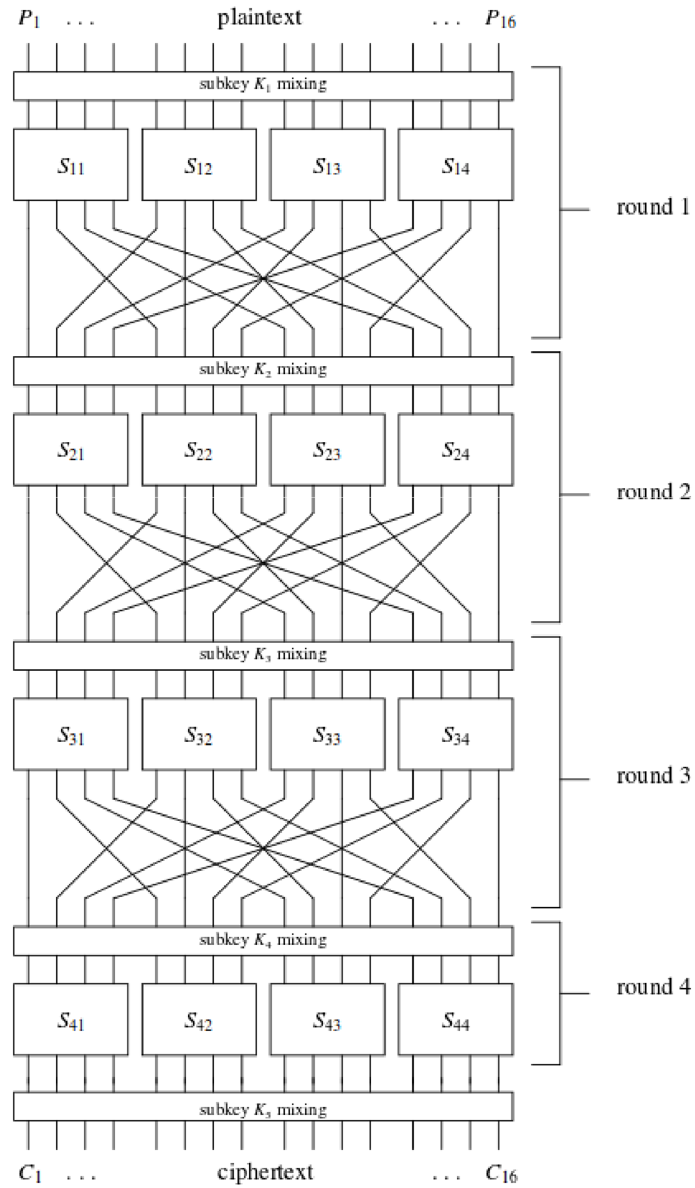
Cache Attacks and Countermeasures: the Case of AES
https://eprint.iacr.org/2005/271.pdf

Cache Timing Attacks on AES
https://cr.yp.to/antiforgery/cachetiming-20050414.pdf

Block Cipher Constructions

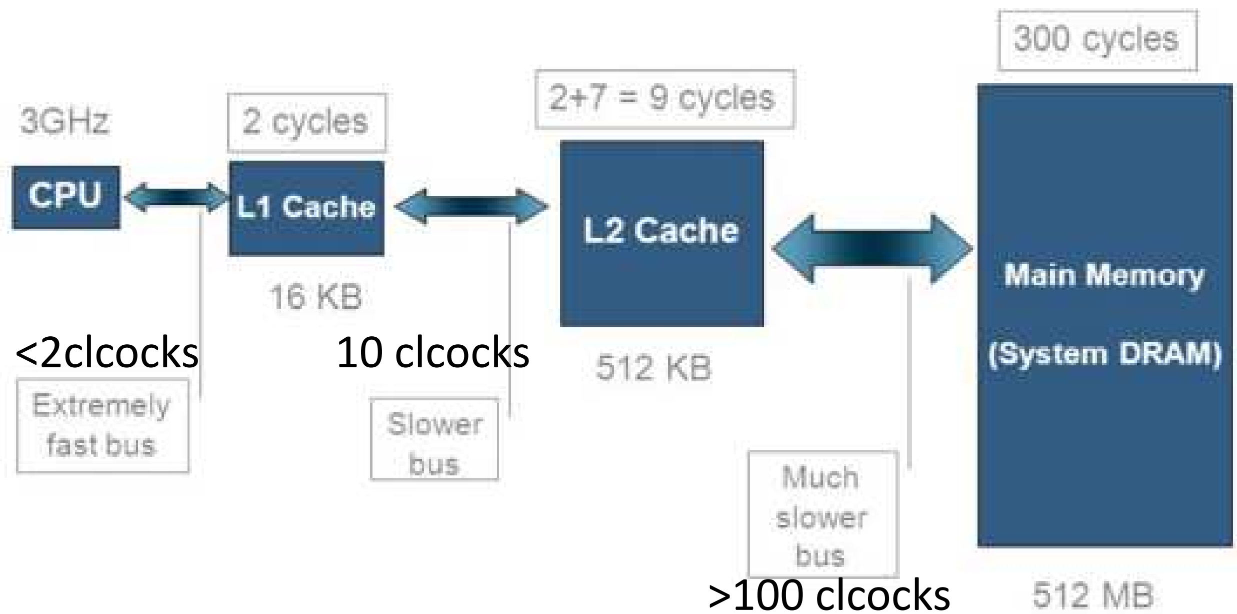
- Sboxes typically implemented with look up tables

- If block cipher is implemented in a system with cache memory, then the look up tables present could lead to timing attacks



Memory Hierarchies in Systems

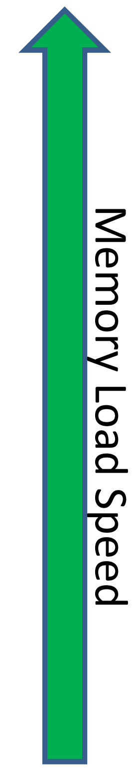
- Von-Neumann bottleneck
  - Due to high speed of processors and relatively low speed of RAM
- Goal of Memory Hierarchy
  - Low latency, high bandwidth, high capacity, low cost



Cache Memories

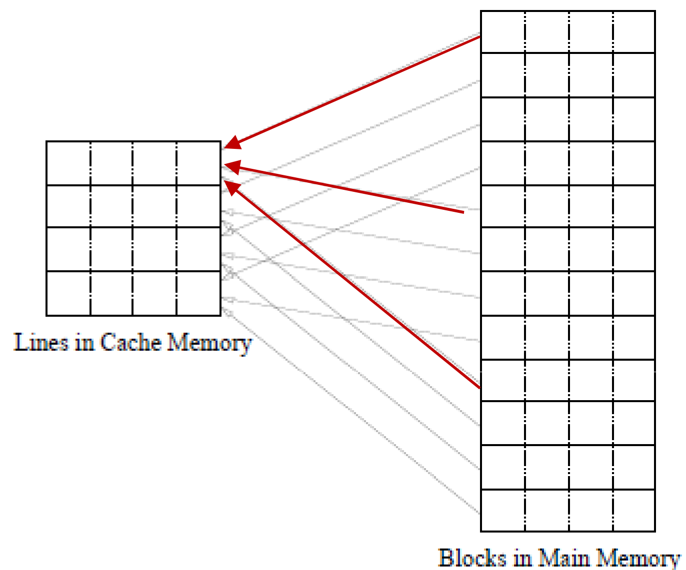
```
Memory Load Instruction{

        If data present in L1 cache  (L1 cache hit){
                then return data from L1 cache
        } else if data not present in L1 cache (L1 cache miss){
                if data present in L2 cache (L2 cache hit){
                        return data from L2 cache and fill L1 cache
                }
                else if data present in L3 cache (L3 cache hit){
                        return data from L3 cache and fill L1 and L2 caches
                }
                else{
                        read data from RAM and fill in all caches
                }
        }
```



Address Mapping of Cache Memories

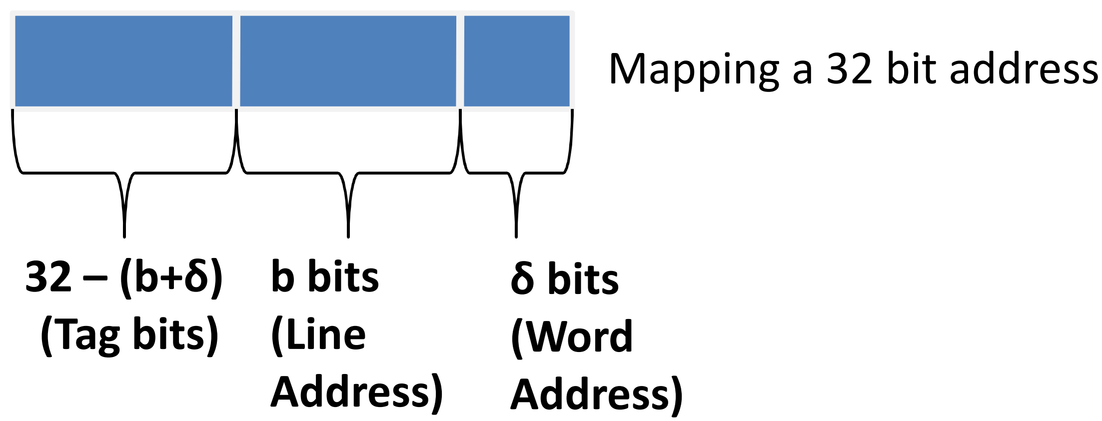
- Memory divided into blocks
  One block typically 64 bytes
- Cache memory divided into lines. Line size = block size.
- There is a mapping from blocks in memory to lines in the cache
  – Example direct mapped cache.
    - If the cahe size contains 4 lines, then every 4-th block gets mapped to the same cache line



CR

Address Mapping of Cache Memories

- Cache Details:
  - Let the number of words in a cache line be $\mathbf{2^{\delta}}$
  - Let the number of lines in the cache be 2^b
  - The number of words in the cache is therefore $2^{b+\delta}$
- How to compute the mapping?

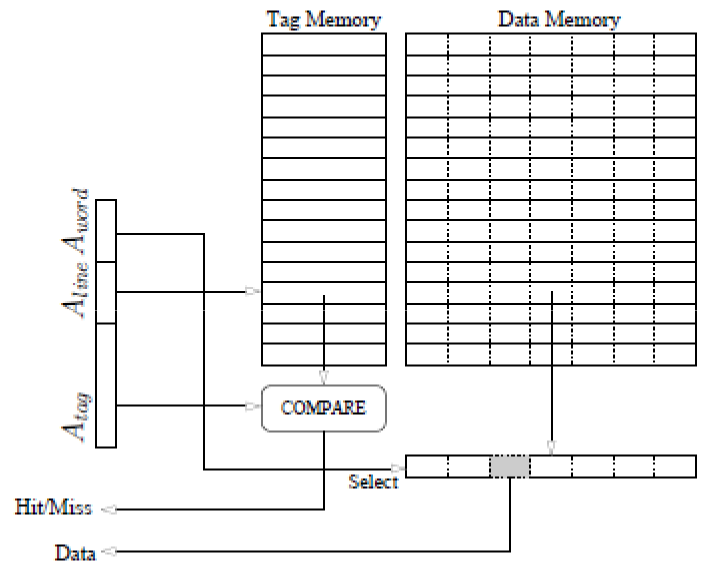


CR

Organization of a Direct Mapped Cache

```
const unsigned char T0[256] = { 0x63, 0x7C, 0x77, 0x7B, ... };
```

T0 address is 0x804af60



Mapping of Table T0 to a Direct-Mapped Cache of size $4KB$ ($2^\delta = 64$ and $2^b = 64$)

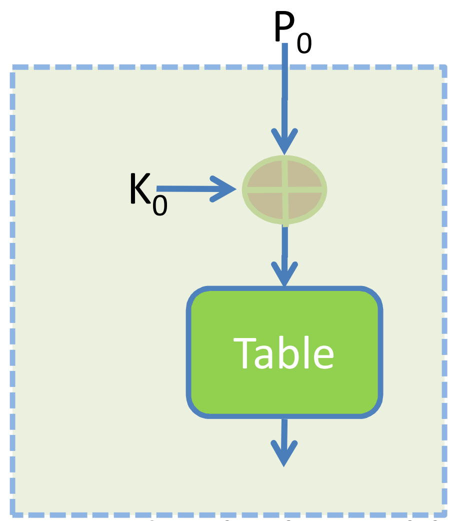
Elements	Address	line	Tag
T0[0] to T0[63]	0x804af40 to 0x804af7f	61	0x804a
T0[64] to T0[127]	0x804af80 to 0x804afbf	62	0x804a
T0[128] to T0[191]	0x804afc0 to 0x804b0ff	63	0x804a
T0[192] to T0[255]	0x804b000 to 0x804b03f	0	0x804b

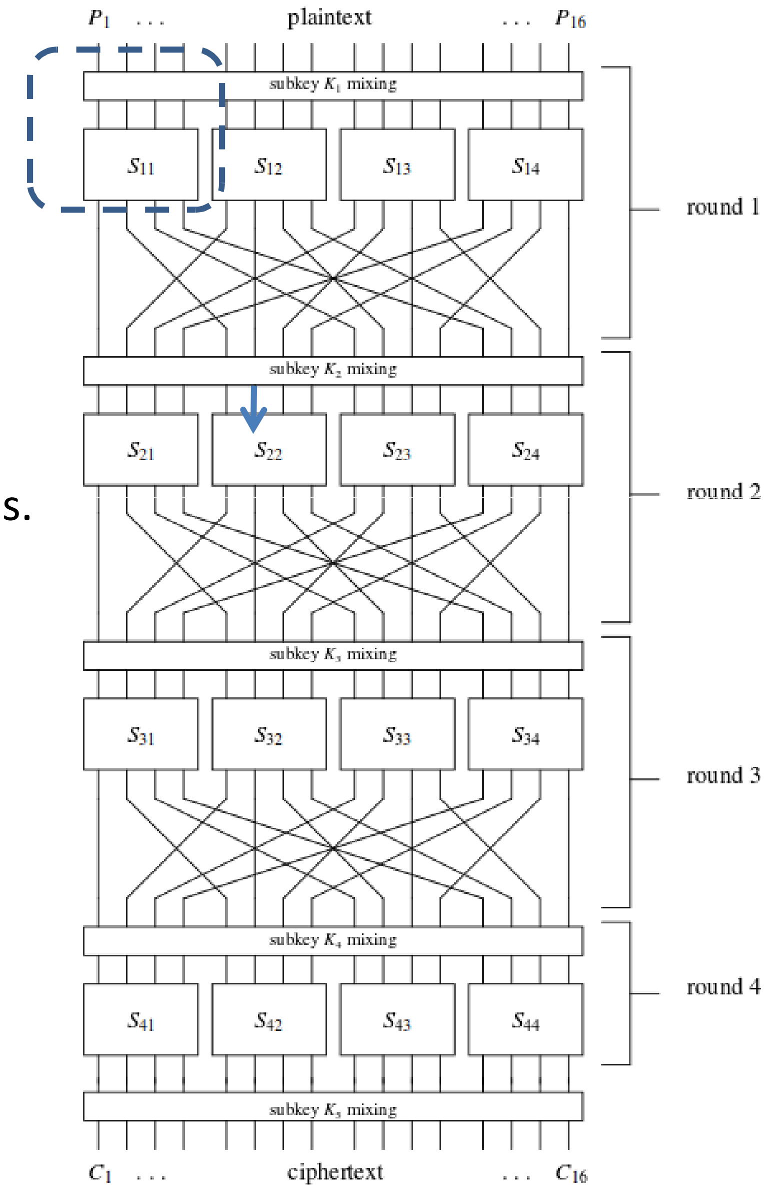
Access Driven Attacks

- Assumptions
  - The attacker shares the same hardware as the victim. For instance, cloud infrastructure.
  - The attacker manipulates the system in such a way as to track execution patterns of a victim process
  - These execution patterns are used to infer sensitive data about the victim

CR

S-boxes and Cache Memories



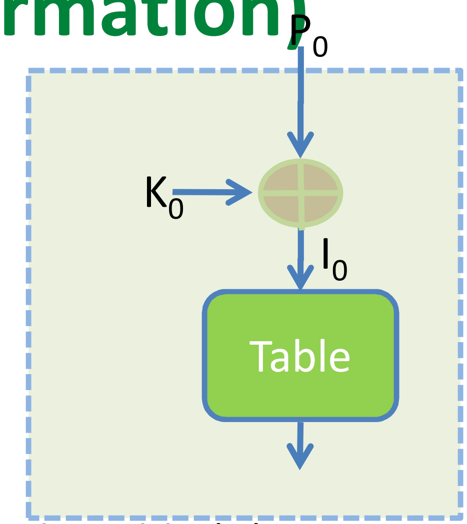


S-boxes generally implemented as lookup tables. Arrays stored in memory.

When accessed, a part of the table gets loaded into the cache memory.

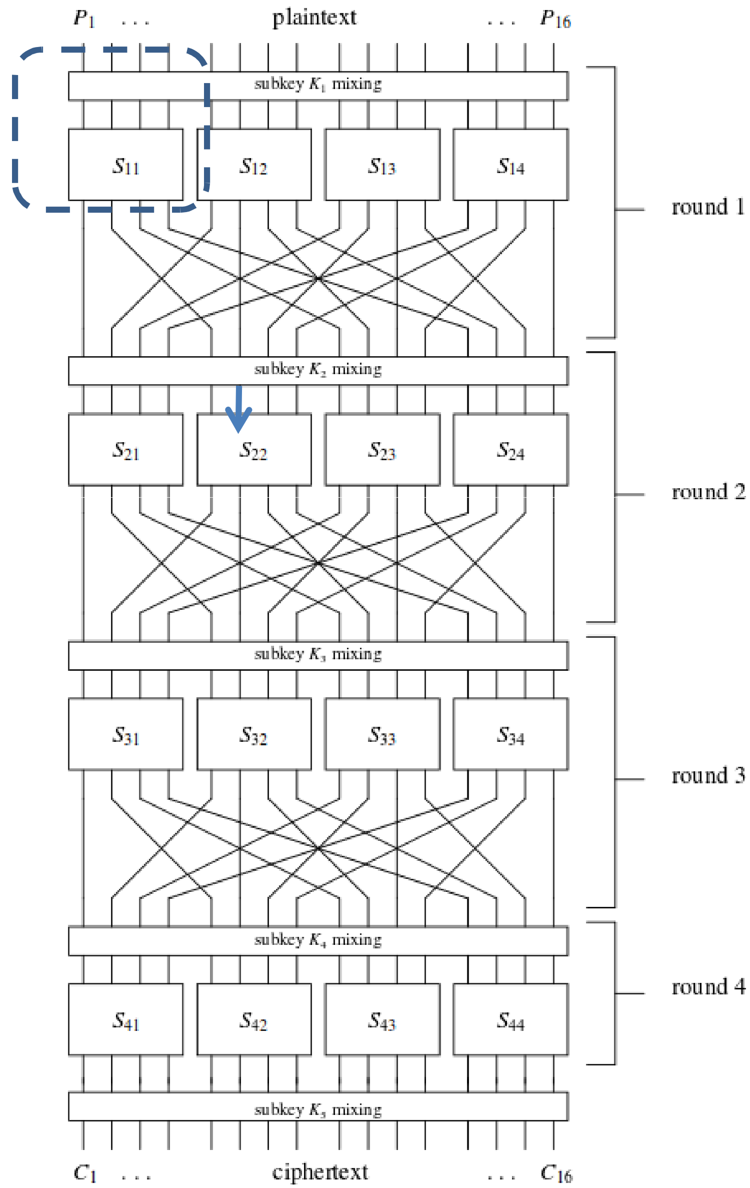
Subsequent accesses to the part of the table results in cache hits (unless evicted).

S-boxes and Cache Memories (getting information)

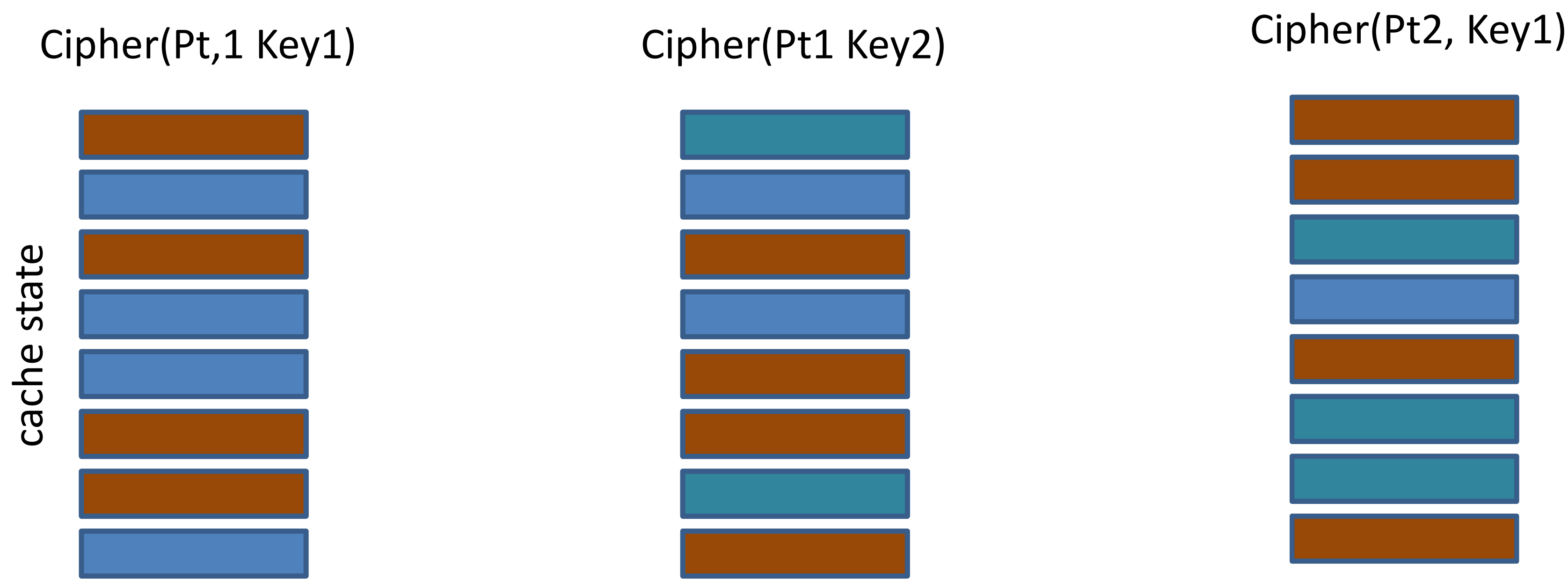


If I know the index into the table (I_0)
and I know P_0 then $\mathbf{P_0}$ **xor** $\mathbf{K_0 = I_0}$
Thus, $\mathbf{P_0}$ **xor** $\mathbf{I_0 = K_0}$

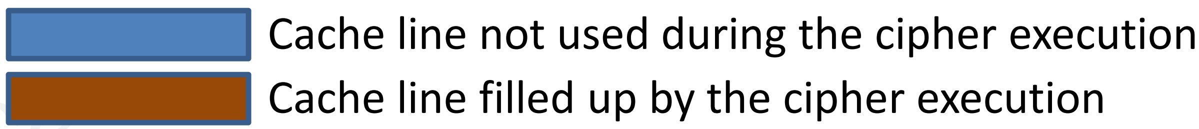
We will see how few bits of I0 can be recovered from monitoring the execution time of the cipher



Cache State when a cipher is executed

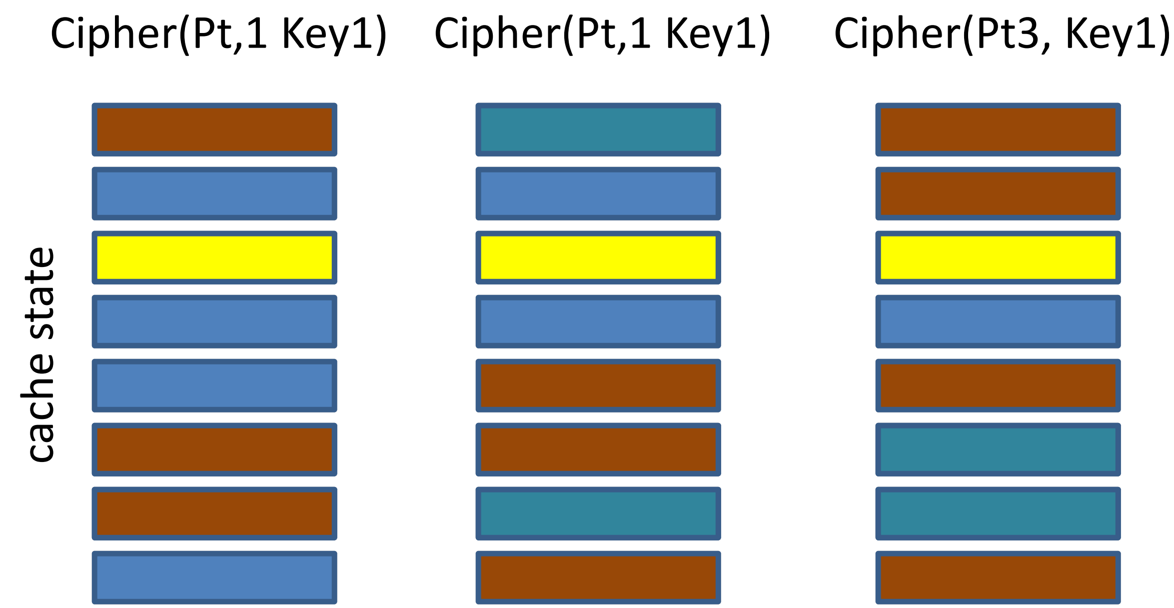


Changing plaintext or key will alter how the cache memory is used



Cache State when a cipher is executed

Pt1, Pt2, Pt3 are same in one byte. All other bytes may be different



When plaintexts have one byte which is same, then there exists one cache line that is filled in every encryption

- Cache line filled in every encryption
- Cache line not used by the cipher execution
- Cache line filled up by the cipher execution

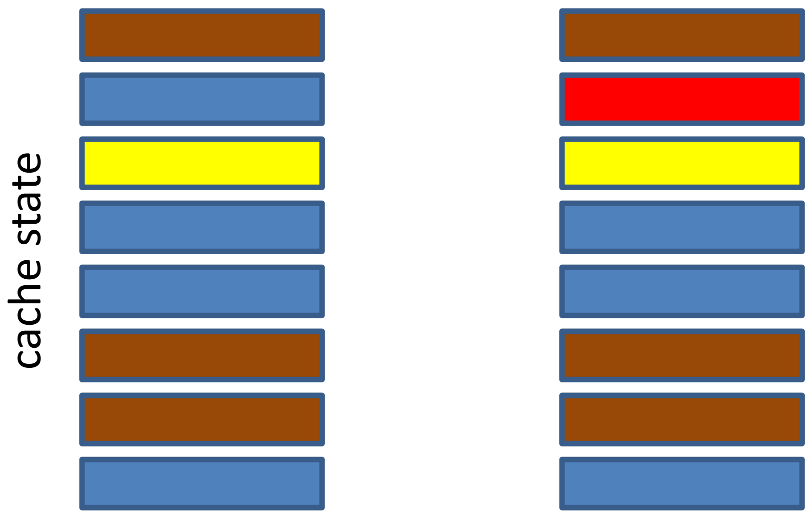
Evict+Time Attack

Repeat multiple times

1. P is a randomly chosen plain text (with one byte say P0 fixed)
2. Invoke encryption of P
3. Evict a random line in the cache (say line L)
4. Invoke encryption of P (again) and **time encryption**

- Note that encryption of P occurs twice. So the second encryption **will predominantly result in cache hits.**
- If line L is used during the encryption, a cache miss arises… leading to an increase in execution time of 2^{nd} encryption
- If line L is not used during the encryption, no additional cache miss arises …. There may not be a significant increase in the execution time of 2^{nd} encryption

What's Happening here?



Evicted line,
Picked randomly is
Shown in red

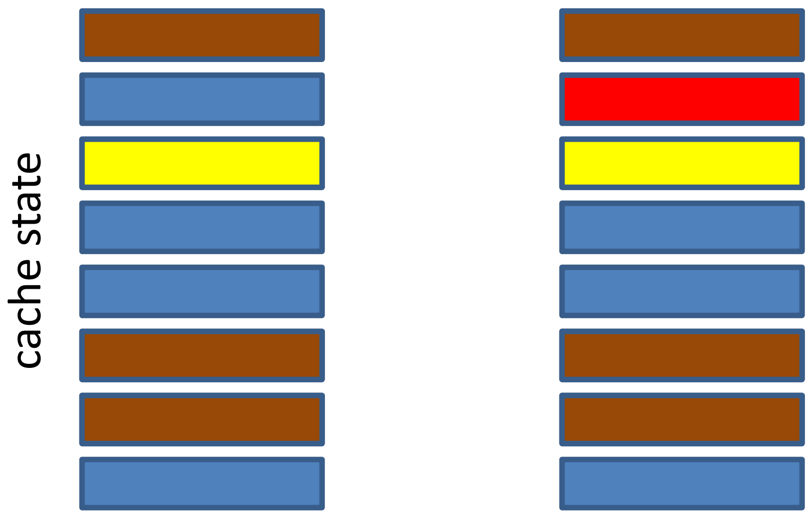
Three scenarios arise

1. Evicted line L (Red) collides with the yellow

2. Evicted line (Red) collides with the brown. But this is unlikely to happen for every encryption, since P changes

3. Evicted line (Red) does not collide with Yellow or Brown. This is also unlikely to happen in every encryption, since P changes.

What's Happening here?



Three scenarios arise

1. Evicted line L (Red) collides with the yellow

2. Evicted line (Red) collides with the brown. But this is unlikely to happen for every encryption, since P changes

3. Evicted line (Red) does not collide with Yellow or Brown. This is also unlikely to happen in every encryption, since P changes.

What can we infer?

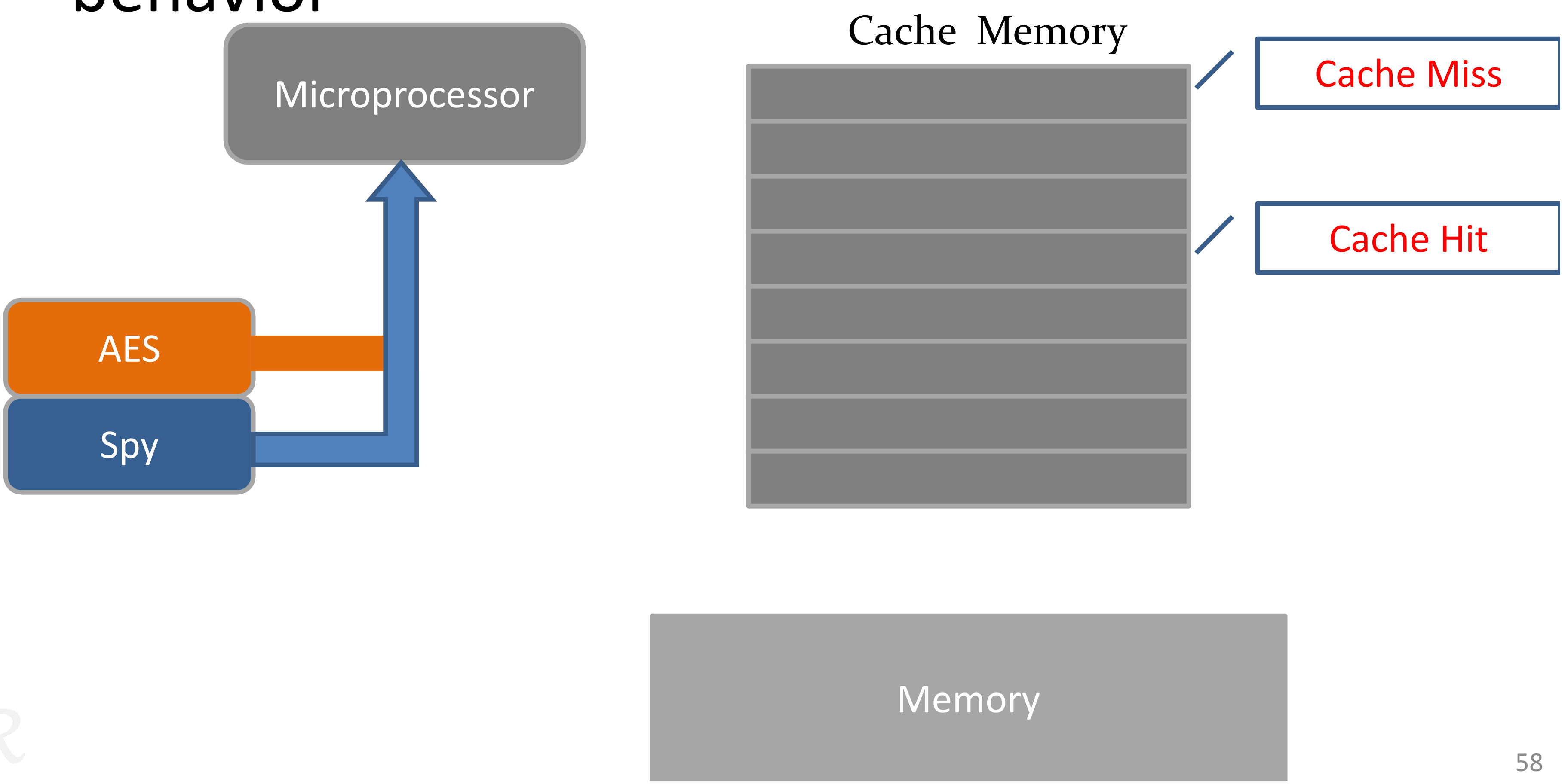
In case 1, there is always an additional Cache miss during the second encryption.

In case 2 or 3, an additional cache miss may or Occur

Thus avg time in case 1 > avg time in case 2 or 3

Prime+Probe

- Uses a spy program to determine cache behavior



CR

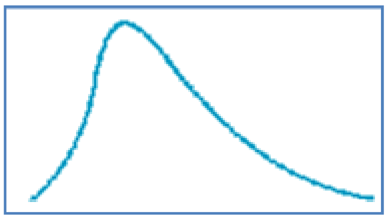
Limitations

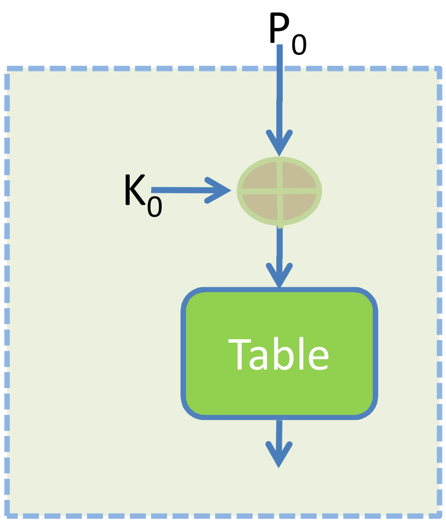
- Number of bits recovered is restricted by the cache line size.

- Solved to certain extent by targeting cache hits in the second round of the block cipher

Bernstein’s Profiled Time Driven Cache Attacks

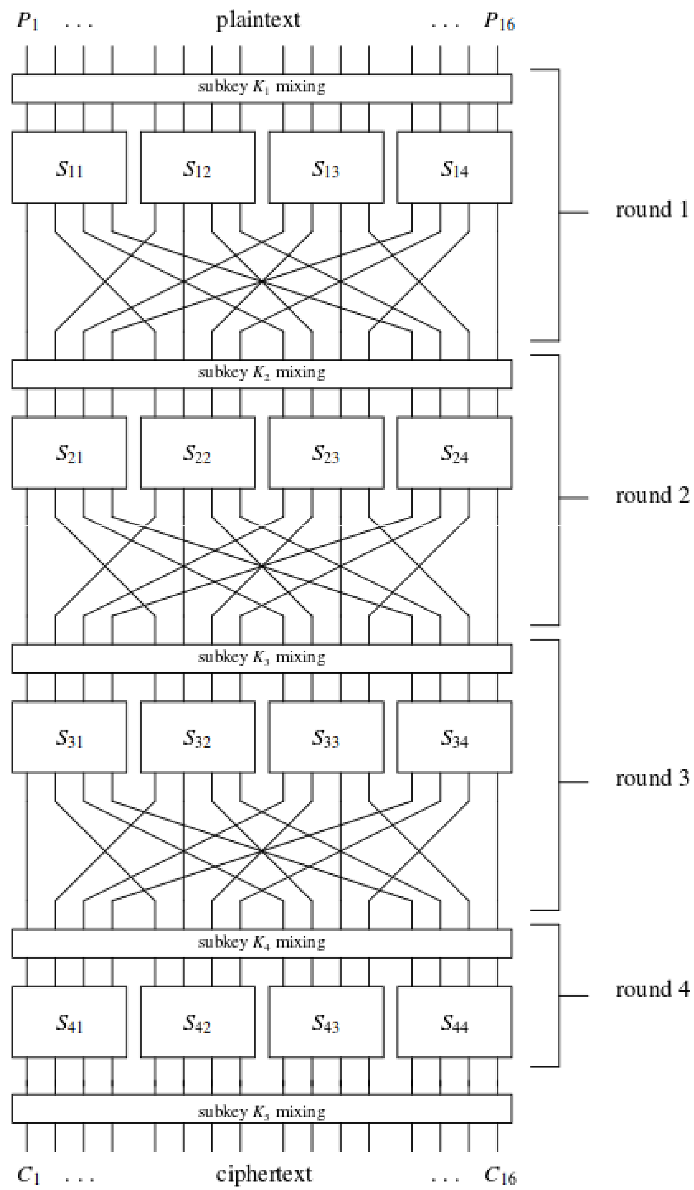
Time Profiles



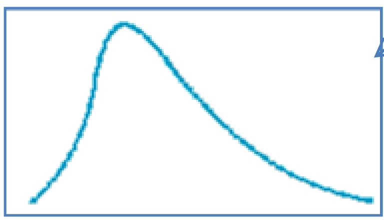


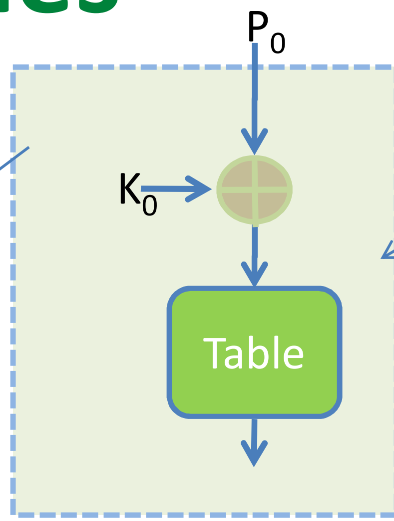
The table is accessed at location P0 ^ K0.

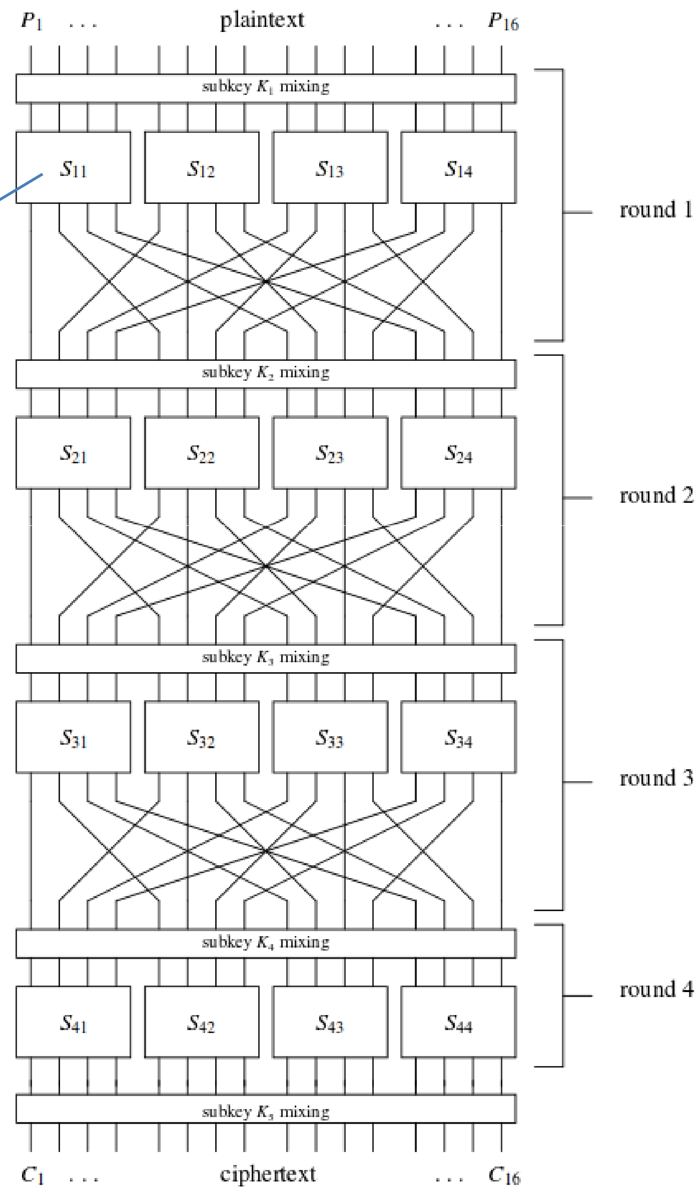
Each value of (P0 ^ K0) results in a unique timing distribution



Time Profiles

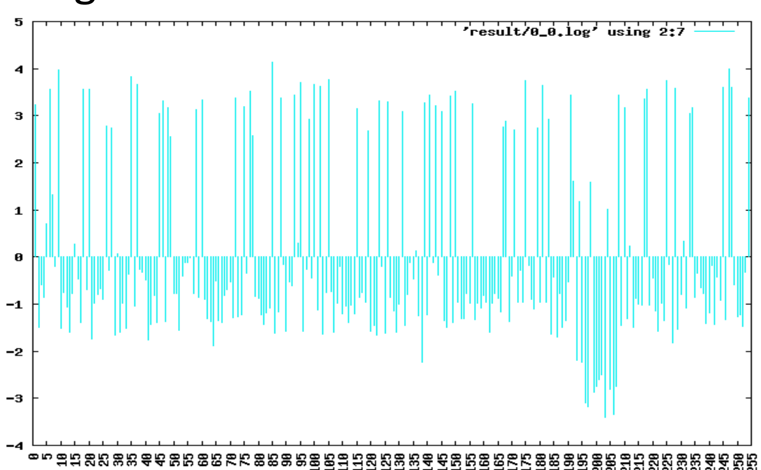






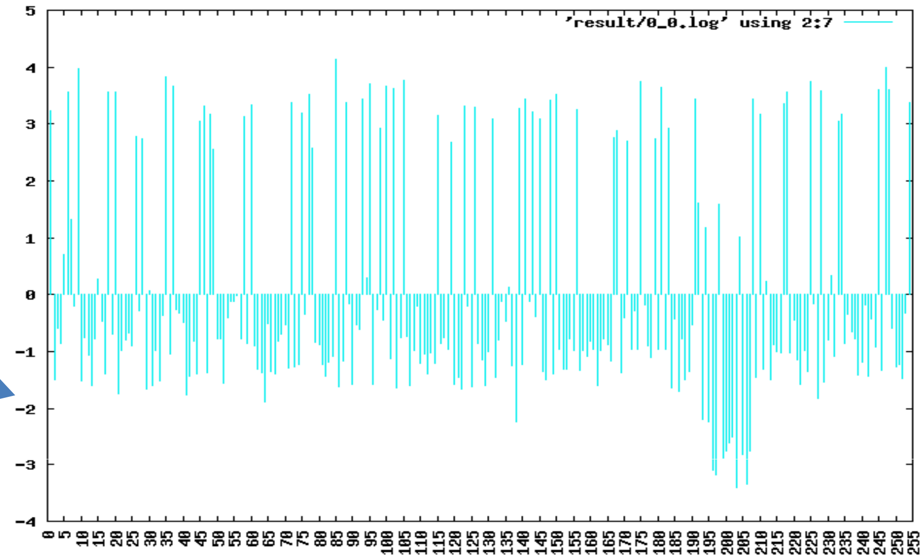
The table is accessed at location P0 ^ K0.

Each value of (P0 ^ K0) results in a unique timing distribution

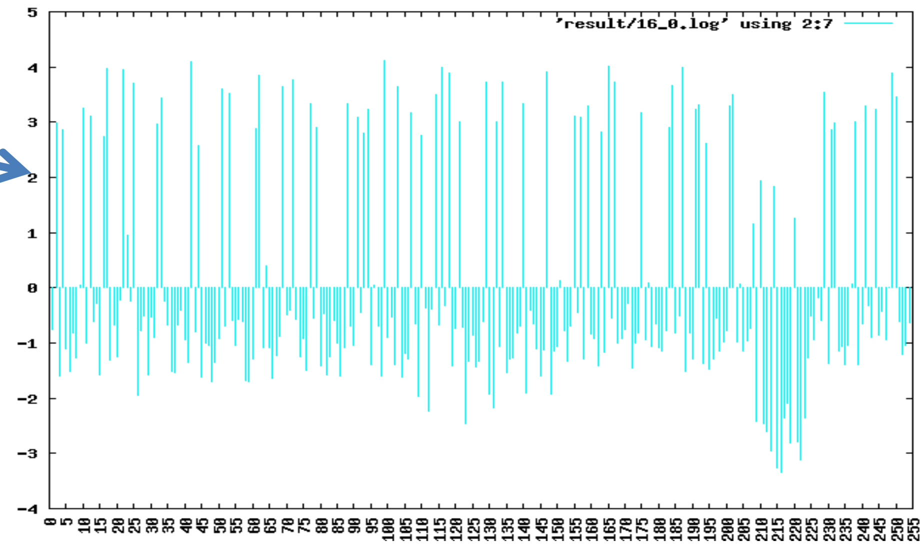


Bernstein's Cache Timing Attack

Put key to all ZEROs and perform experiment



Repeat experiment with unknown key



Correlate the two results

Results for the Block Cipher AES

key	Correct key	Ten most likely keys for each byte									
$k_0^{(0)}$	11	4e	47	41	4a	46	4c	48	45	4f	44
$k_1^{(0)}$	22	05	22	c2	*2f*	ca	33	e1	06	*23*	c9
$k_2^{(0)}$	33	33	*38*	*3b*	*3a*	*34*	*37*	*39*	0c	*3f*	a7
$k_3^{(0)}$	44	83	89	8a	81	*41*	8b	84	*46*	*4b*	*4a*
$k_4^{(0)}$	55	d1	de	d9	a8	d0	d3	aa	a5	a0	a1
$k_5^{(0)}$	66	8f	52	c3	7a	2b	50	1a	23	f6	4a
$k_6^{(0)}$	77	79	*73*	*78*	*74*	77	*7e*	*7f*	*75*	8d	8e
$k_7^{(0)}$	88	*8e*	*87*	*8f*	*80*	*8a*	*86*	*89*	*8d*	*8b*	88
$k_8^{(0)}$	99	99	39	83	*90*	ba	1e	7a	af	70	13
$k_9^{(0)}$	aa	b4	e2	7b	e8	b1	c8	53	7a	79	bb
$k_{10}^{(0)}$	bb	65	57	5f	*b2*	24	*b6*	60	25	5e	80
$k_{11}^{(0)}$	cc	*c6*	*c2*	*ce*	*ca*	*cb*	cc	*c1*	*c0*	14	*cf*
$k_{12}^{(0)}$	dd	53	5b	50	52	49	58	5d	51	*d1*	48
$k_{13}^{(0)}$	ee	7c	*e0*	4e	98	94	*eb*	*e5*	d7	b3	3b
$k_{14}^{(0)}$	ff	ea	*fd*	*fb*	3a	e1	a4	e9	03	*f1*	ff
$k_{15}^{(0)}$	00	*05*	*01*	*06*	*02*	*04*	*08*	*03*	*0a*	*00*	*0c*

CR

Results for the Block Cipher CLEFIA

key	Correct key	Ten most likely keys for each byte									
$RK0_0$	f4	f4	e2	c1	eb	52	18	e1	d7	14	44
$RK0_1$	d0	d0	52	f0	df	46	51	d8	44	f2	d7
$RK0_3$	6a	6a	5f	94	92	e8	48	6c	75	a9	b6
$RK1_0$	ca	ca	a7	5b	40	54	52	bf	58	51	53
$RK1_1$	7b	7b	46	db	d1	c6	c4	52	56	8f	79
$RK1_2$	91	91	13	5a	8c	f2	14	64	a8	f6	36
$RK1_3$	60	60	ab	07	68	c5	ec	9c	78	e9	16
$RK2_0 \oplus WK0_0$	fe	fe	f8	00	06	ec	14	11	1c	f6	1b
$RK2_1 \oplus WK0_1$	57	57	51	62	a7	5a	f7	64	24	e1	9f
$RK2_2 \oplus WK0_2$	3c	3c	ea	c5	eb	3d	8c	be	92	11	ec
$RK2_3 \oplus WK0_3$	80	80	51	02	58	57	3c	d8	89	10	74
$RK3_0 \oplus WK1_0$	6b	6b	7b	42	90	6f	a3	56	d6	3d	a9
$RK3_1 \oplus WK1_1$	40	40	4a	b1	88	fd	92	16	2b	05	13
$RK3_2 \oplus WK1_2$	16	16	05	94	fd	45	6b	b9	15	f8	6e
$RK3_3 \oplus WK1_3$	36	36	f2	42	a8	ad	86	80	c5	1b	34
$RK4_0$	7e	7e	e0	fe	e8	01	11	ff	07	1c	12
$RK4_1$	32	32	2f	34	26	38	31	35	3f	3e	29
$RK4_2$	50	50	5d	00	a0	81	f0	65	82	b0	03
$RK4_3$	e1	e1	0e	37	dc	63	cc	c8	e5	89	77
$RK5_0$	eb	eb	9b	da	85	1e	f8	3e	fe	4c	99
$RK5_1$	11	11	24	e9	ef	33	93	cd	0e	d2	17
$RK5_2$	47	47	37	92	f8	99	8c	bb	34	b2	52
$RK5_3$	35	35	b7	38	7f	e7	5f	31	e8	8b	ed

Countermeasures for Timing Attacks

- Requirements for a successful Side Channel Attack
  - Perturbations :
    - When the cipher executes, some entity in the system must be disturbed (perturbed)
  - Manifestations:
    - These perturbations should be manifested through some channel (for instance a power glitch)
  - Oberservable:
    - The manifestations should be observable / measurable in spite of all the noise
- Preventing any one of these requirements can counter side channel attacks.

CR

Preventing Cache Timing Attacks

- Adding noise during the encryption ….
- Constant time implementations ….. difficult
- Non-cached memory access
- Specialized cache designs
  – Partitioned cache
  – Random permutation cache
- Specialized Instructions
- Prefetching
- Fuzzing Clocks
  – Virtual time stamp counters

Fault Attacks

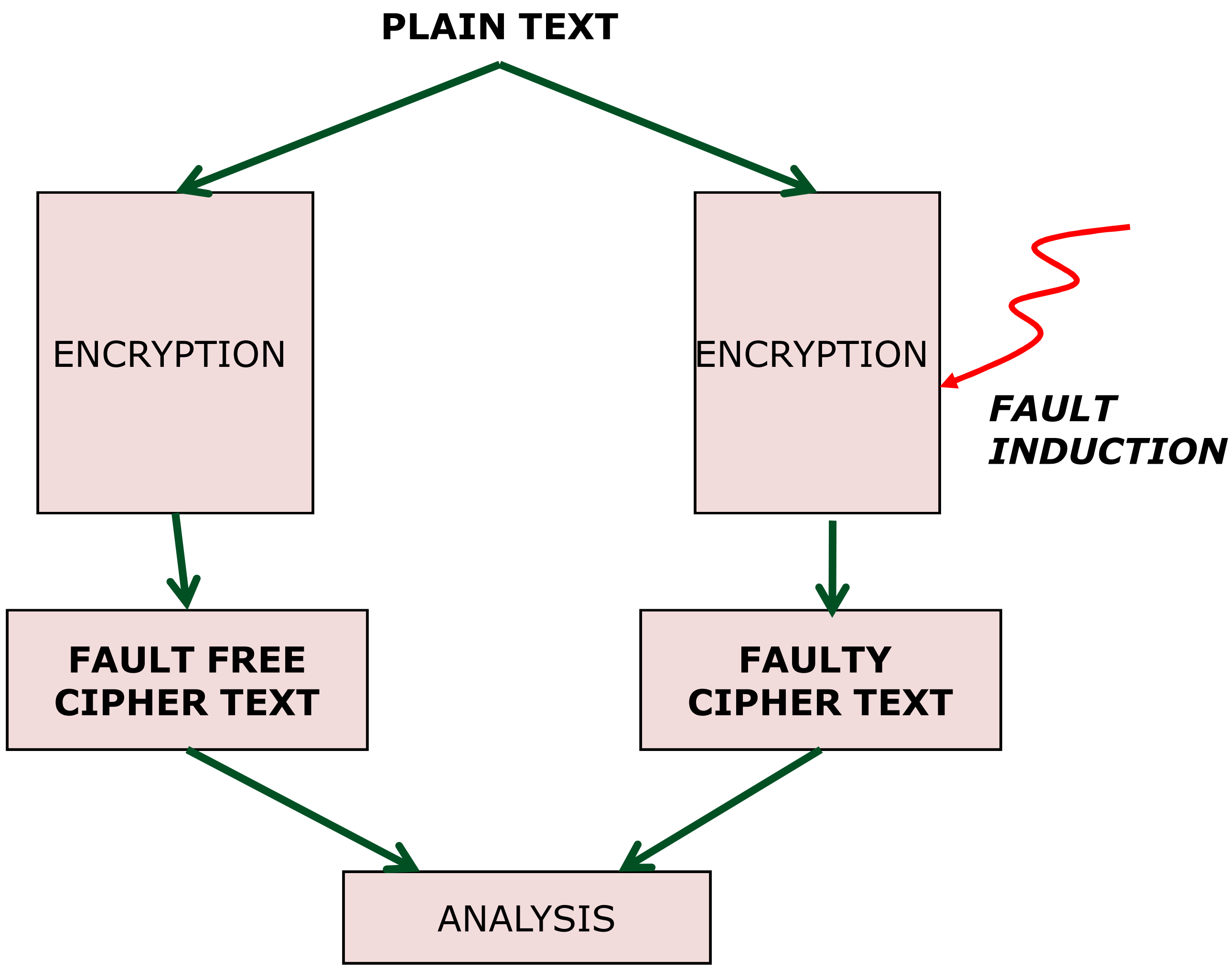
"Differential Fault Analysis of the Advanced Encryption Standard using a Single Fault", Michael Tunstall, Debdeep Mukhopadhyay, and Subidh Ali
https://eprint.iacr.org/2009/575.pdf

Fault Attacks

- Active Attacks based on induction of faults
- First conceived in 1996 by Boneh, Demillo and Lipton
- E. Biham developed Differential Fault Analysis (DFA) attacker DES
- Optical fault induction attacks : Ross Anderson, Cambridge University – CHES 2002

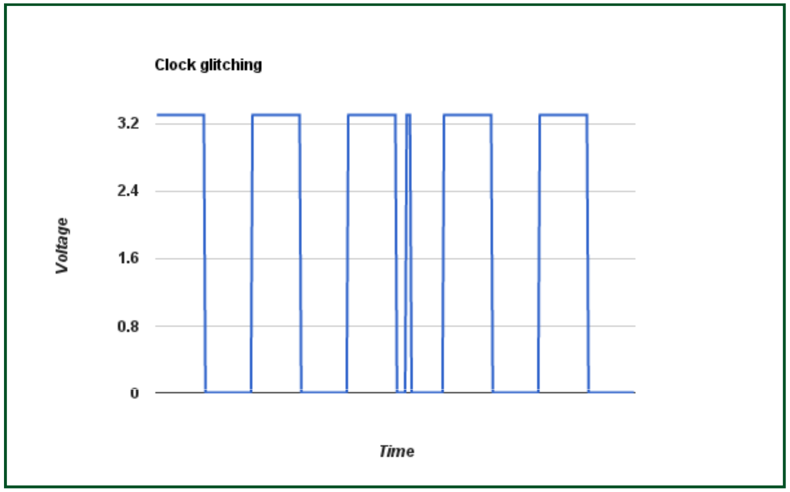
CR

Illustration of a Fault Attack

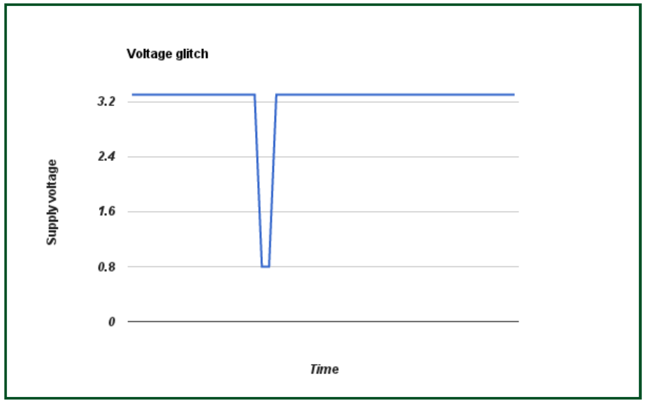


How to achieve fault injection

Laser



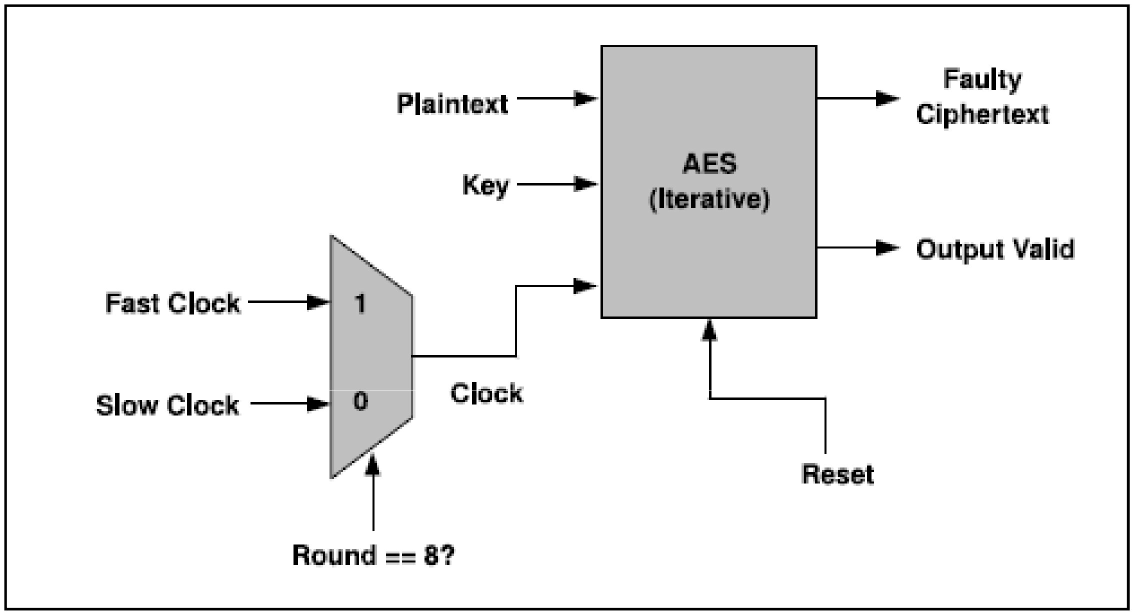
Clock Glitching



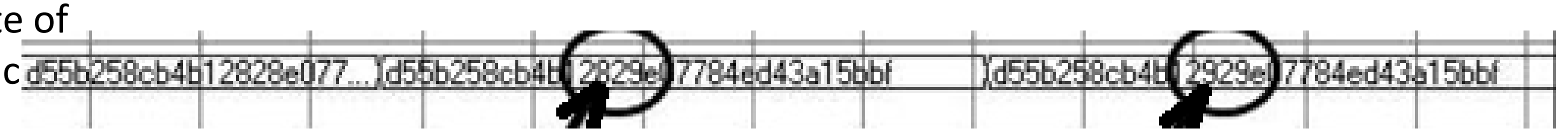
Power Glitching

Temperature???

Fault Injection Using Clock Glitches



An Internal state of
The AES on logic
scope



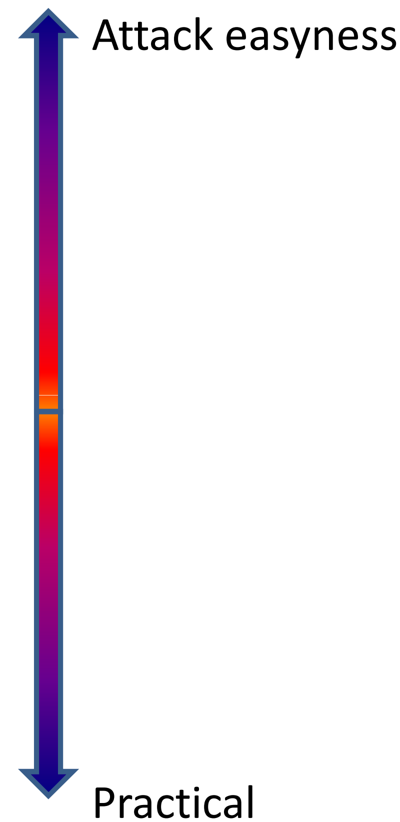
Fault Models

- Bit model : When fault is injected, exactly one bit in the state is altered

  eg. 8823124345 → 8833124345
- Byte model : exactly one byte in the state is altered

  eg. 8823124345 → 8836124345
- Multiple byte model : faults affect more than one byte

  eg. 8823124345 → 8836124333



Fault injection is difficult…. The attacker would want to reduce the number of faults to be injected

Fault Attack on RSA

RSA decryption has the following operation

$$x = y^a \bmod n$$

where a is the private key y the ciphertext and x the plain text

Suppose, the attacker can inject a fault in the i^{th} bit of a.
Thus she would get two ciphertexts:

The fault free ciphertext $x = y^a \bmod n$
The faulty ciphertext $\tilde{x} = y^{\tilde{a}} \bmod n$

CR

Fault Attack on RSA

$$a \text{ and } \tilde{a} \text{ differ by exactly 1 bit; the } i^{th} \text{ bit. Thus}$$

$$a - \tilde{a} = \begin{cases} 2^i & \text{if } a_i = 1 \\ -2^i & \text{if } a_i = 0 \end{cases}$$

Now consider the ratio

$$\frac{x}{\tilde{x}} = \frac{y^a}{y^{\tilde{a}}} \bmod n = y^{a-\tilde{a}} \bmod n$$

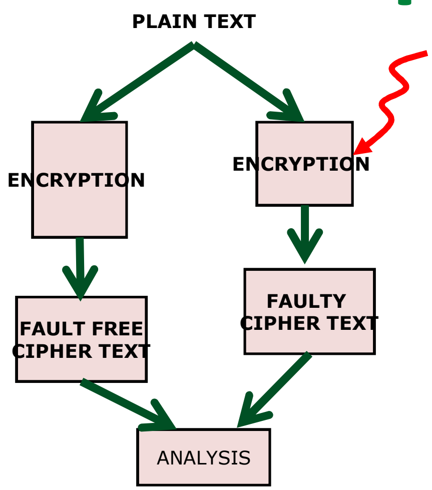
$$Thus,$$

$$\frac{x}{\tilde{x}} = \begin{cases} y^{2^i} & \text{if } a_i = 1 \\ y^{-2^i} & \text{if } a_i = 0 \end{cases}$$

The attacker thus gets 1 bit of a_i. Similar faults on other bits will reveal more information about the private key a_i

CR

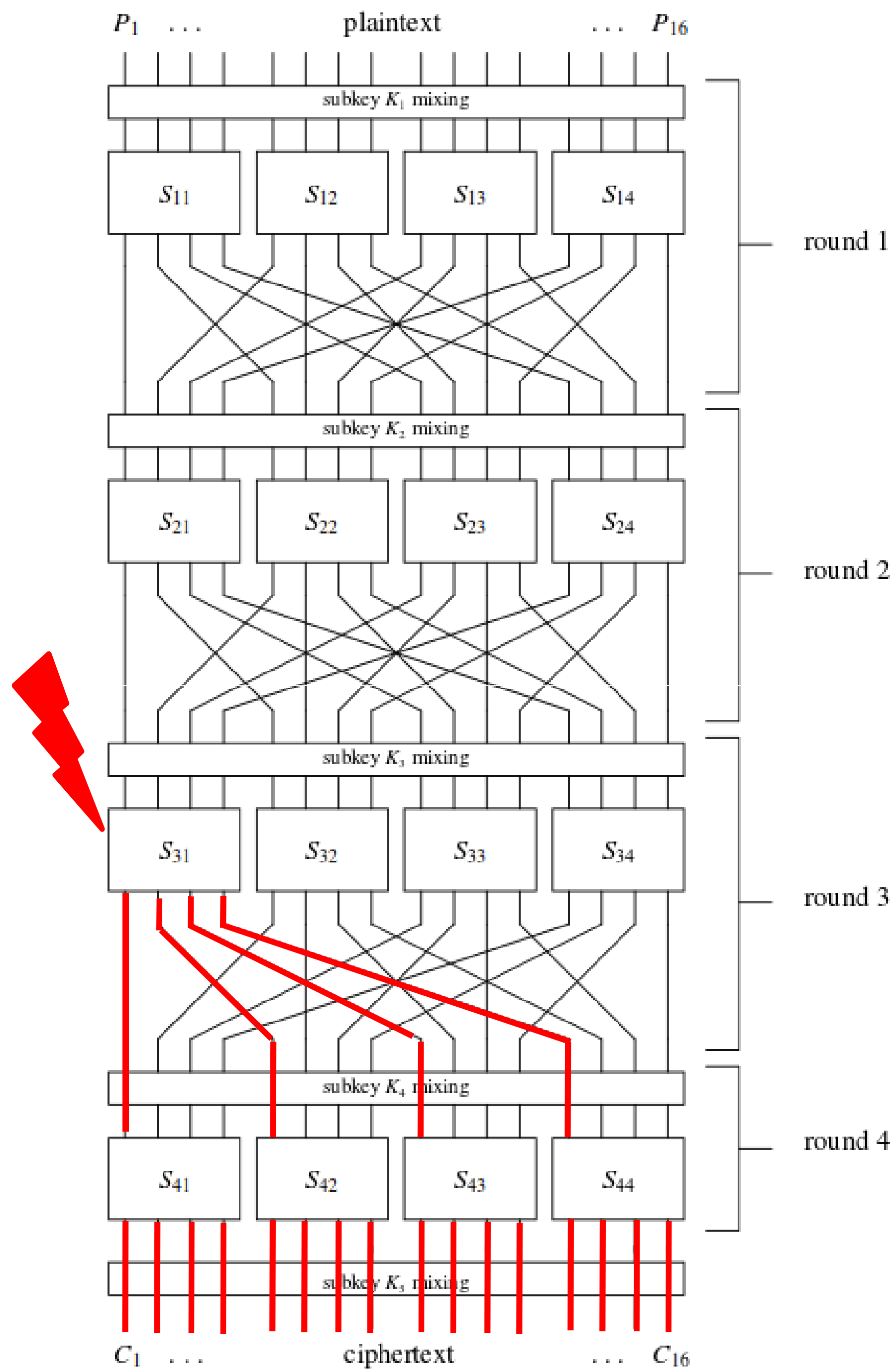
What a fault does to a block cipher?



- A fault (generally at the s-box input) creates a difference wrt the fault free encryption
- This difference is propagated and diffused to multiple output bytes of the cipher
- The attacker thus has 2 cipertexts :

  (1) the fault free ciphertext (C)

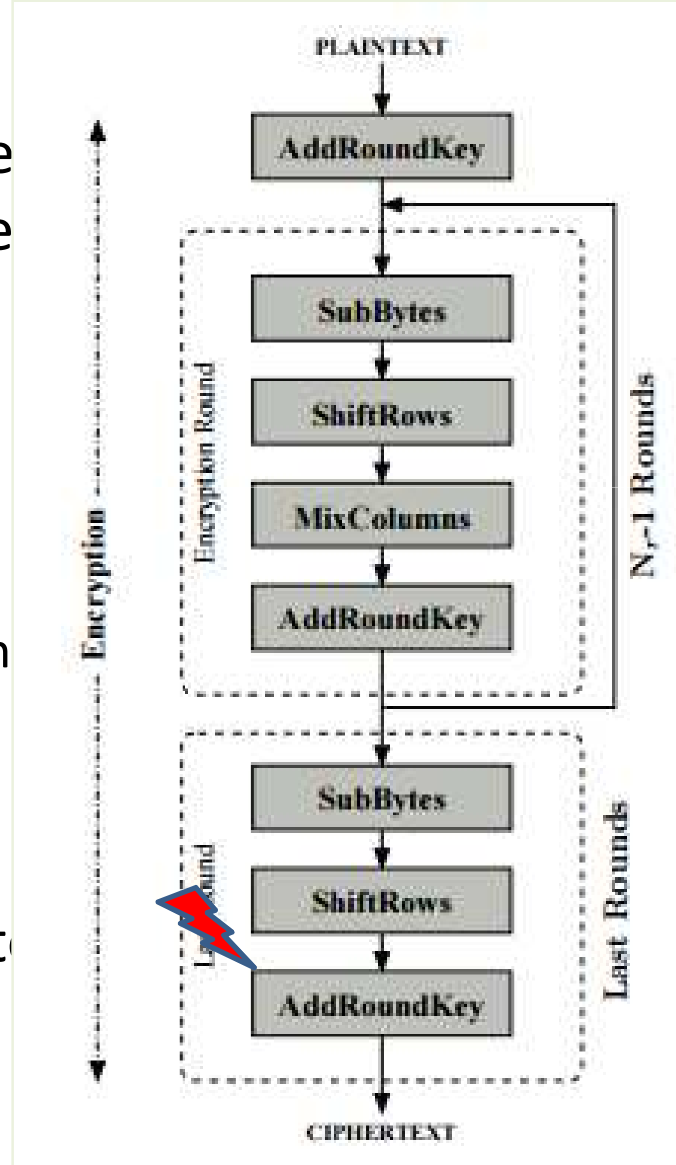
  (2) the faulty ciphertext (C*)



A Simple Fault Attack on AES

- Let's assume that the attacker has the capability of resetting a particular line during the AES round key addition.

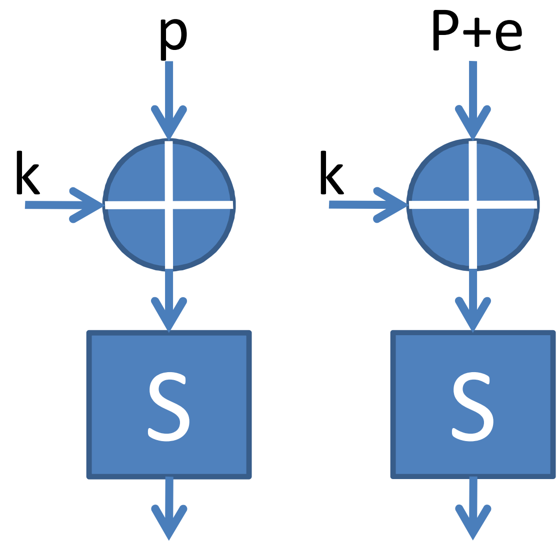
  (i.e. exactly one bit is reset)
- Attack Procedure
  1. Put plaintext to 0s and get ciphertext C
  2. Put plaintext to 0s. Inject fault in the ith bit as shown. Get the ciphertext C*
  3. If C=C*, we infer $K_i = 1$
     If C≠C*, we infer $K_i = 0$
- This techniques requires 128 faults to be injected.
  – difficult,,,, can we do better?



CR

Differential Fault Attack on AES

- Differential characteristics of the AES s-box



DFA on last round of AES (using a single bit fault)

$C_0 + C_0^{*} = S(p) + S(p+f)$

Since it is a single bit fault,

f can take on one of 7 different values:

(00000001), (00000010), (000001000),

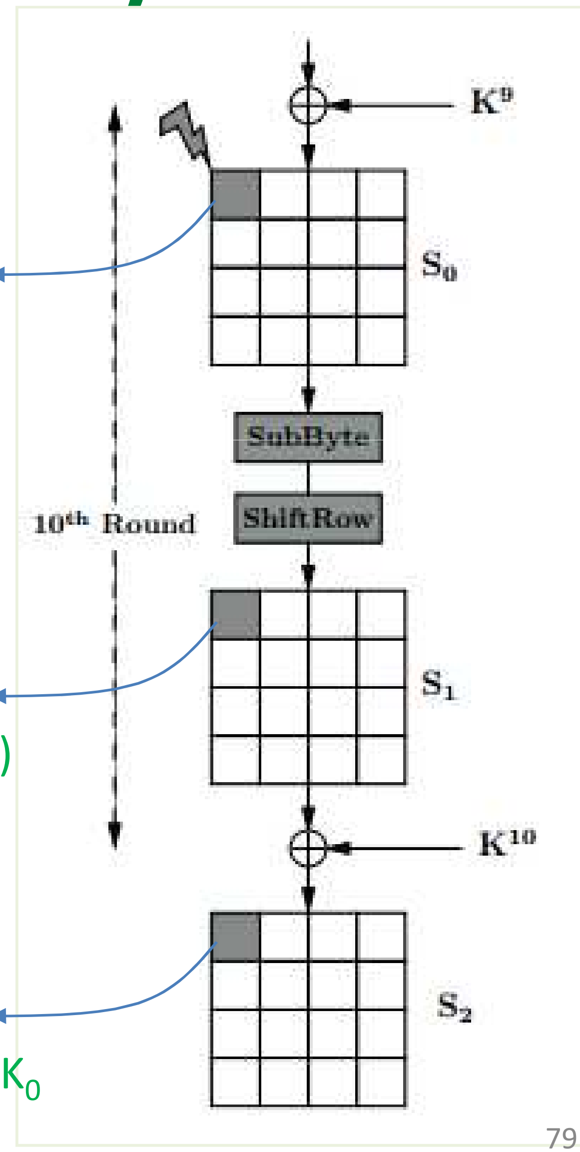
(000010000), …. , (10000000)

The above equation on average will

have around 8 different solutions for p.

Each value of p would give a candidate for k.

Thus, there are 8 key candidates.



DFA on last round of AES (using a single bit fault)

- Each bit fault results in 8 potential key values for the byte
- There are 16 key bytes. Thus 16 faults need to be injected.
- In total key space reduces from 2^{128} to 8^{16} (ie. 2^{48})
  - A key space search of 2^{48} do-able in reasonable time

CR

DFA on 9th Round of AES (fault in a byte)

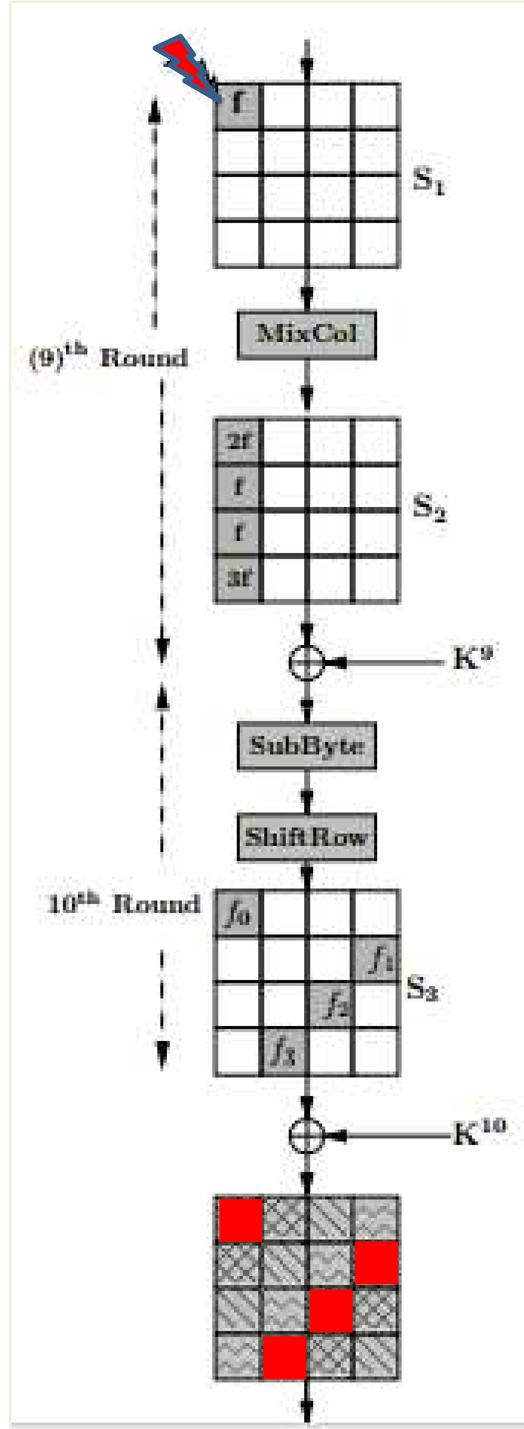
- Fault injected after s-box operation in the 9th round.
- It is a byte level fault, thus, the fault ‘f’ can take on any of 256 values (0, 1, 2, …. , 255)
- Due to the mix-column, 4 difference equations can be derived

$$2f = S^{-1}(C_{0,0} \oplus K^{10}_{0,0}) \oplus S^{-1}(C^{*}_{0,0} \oplus K^{10}_{0,0})$$

$$f = S^{-1}(C_{1,3} \oplus K^{10}_{1,3}) \oplus S^{-1}(C^{*}_{1,3} \oplus K^{10}_{1,3})$$

$$f = S^{-1}(C_{2,2} \oplus K^{10}_{2,2}) \oplus S^{-1}(C^{*}_{2,2} \oplus K^{10}_{2,2})$$

$$3f = S^{-1}(C_{3,1} \oplus K^{10}_{3,1}) \oplus S^{-1}(C^{*}_{3,1} \oplus K^{10}_{3,1})$$



Solving the Difference Equations

Each equation has the form : $A = B \oplus C$

where, A, B, C are of 8 bits each.

$$2f = S^{-1}(C_{0,0} \oplus K^{10}_{0,0}) \oplus S^{-1}(C^*_{0,0} \oplus K^{10}_{0,0})$$
$$f = S^{-1}(C_{1,3} \oplus K^{10}_{1,3}) \oplus S^{-1}(C^*_{1,3} \oplus K^{10}_{1,3})$$
$$f = S^{-1}(C_{2,2} \oplus K^{10}_{2,2}) \oplus S^{-1}(C^*_{2,2} \oplus K^{10}_{2,2})$$
$$3f = S^{-1}(C_{3,1} \oplus K^{10}_{3,1}) \oplus S^{-1}(C^*_{3,1} \oplus K^{10}_{3,1})$$

For a uniformly random choice of A, B, and C,
the probability that the above equation is satisfied is $(1/2^8)$
The maximum space of (A,B,C) is 2^{24}. Of these values, 2^{16} will satisfy the above equation

Solving the Difference Equations

$$
\begin{aligned}
2f &= S^{-1}(C_{0,0} \oplus K_{0,0}^{10}) \oplus S^{-1}(C_{0,0}^{*} \oplus K_{0,0}^{10}) \\
f &= S^{-1}(C_{1,3} \oplus K_{1,3}^{10}) \oplus S^{-1}(C_{1,3}^{*} \oplus K_{1,3}^{10}) \\
f &= S^{-1}(C_{2,2} \oplus K_{2,2}^{10}) \oplus S^{-1}(C_{2,2}^{*} \oplus K_{2,2}^{10}) \\
3f &= S^{-1}(C_{3,1} \oplus K_{3,1}^{10}) \oplus S^{-1}(C_{3,1}^{*} \oplus K_{3,1}^{10})
\end{aligned}
$$

Each equation has the form : $A = B \oplus C$

where, A, B, C are of 8 bits each.

For a uniformly random choice of A, B, and C,
the probability that the above equation is satisfied is $(1/2^8)$
The maximum space of (A,B,C) is 2^{24}. Of these values, 2^{16} will satisfy the above equation

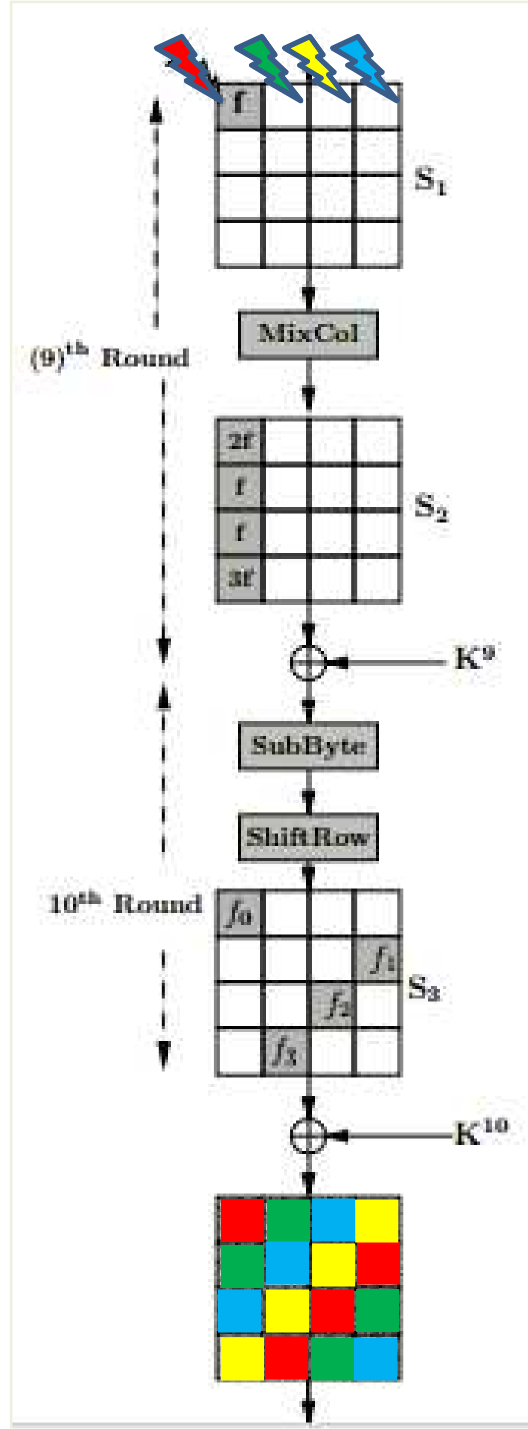
In our case, there are 5 unknowns (4 keys and f) and 4 equations.
For uniformly random chosen values of the 5 unknowns, the probability that all 4 equations are satisfied is $p=(1/2^8)^4$.
The space reduction for the 5 variables is therefore from $p(2^8)^5 = 2^{8(5-4)} = 2^8$.

The key space is 2^{32}. From the above, it has reduced to just 2^8.

Each fault reveals 32 bits of the 10th round key.
Thus 4 faults are required to reveal all 128 key bits. The offline search space is 2^{32}.
Can we do better?

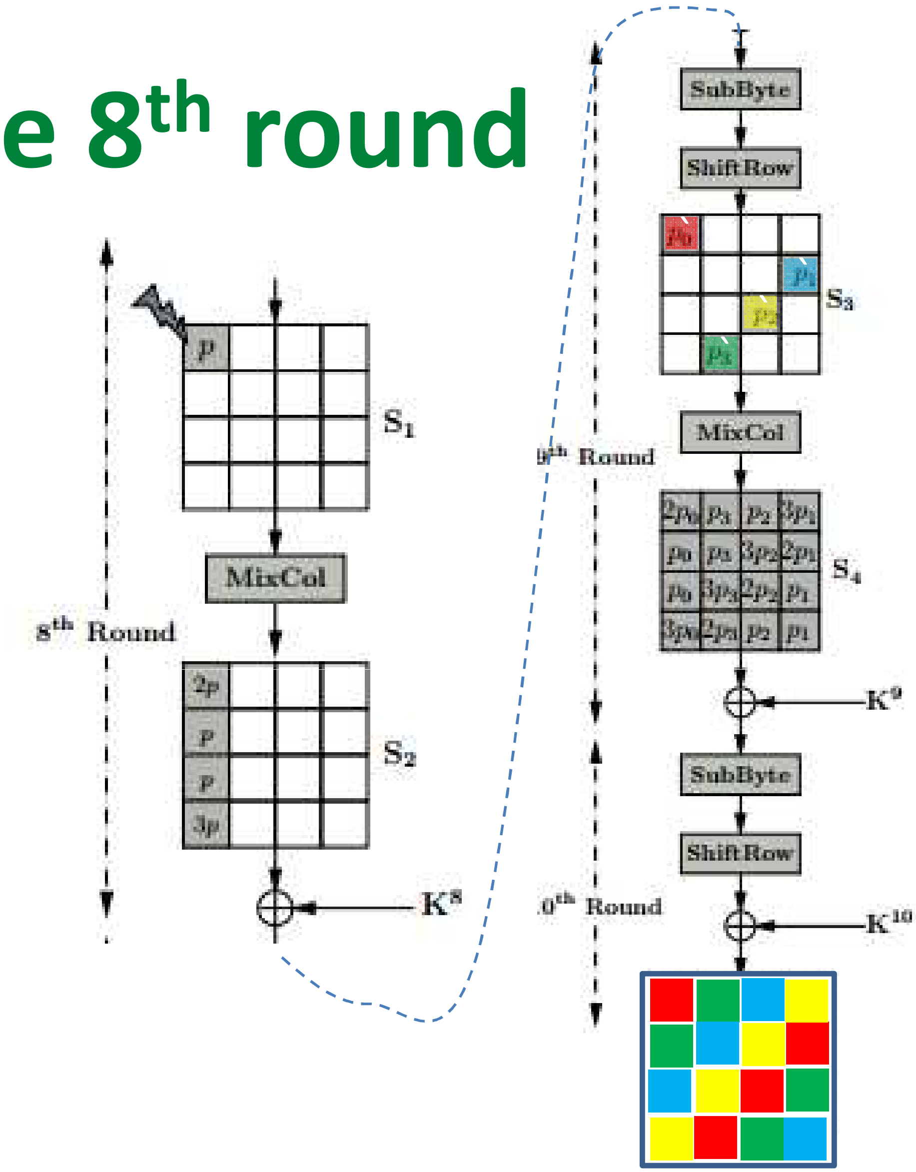
DFA on AES with a single fault

- As mentioned previously, 4 faults are required in the 9th round to reveal the entire key
- Instead of the 9th round, suppose we inject the fault in the 8th round



DFA on AES in the 8th round

- A single fault injected in the 8th round will spread to 4 bytes in the 9th round.
- This is equivalent to having 4 faults in each of the 4 columns.
- A single fault can thus be used to determine all key bytes.
- The offline key space is 2^{32} as before



CR