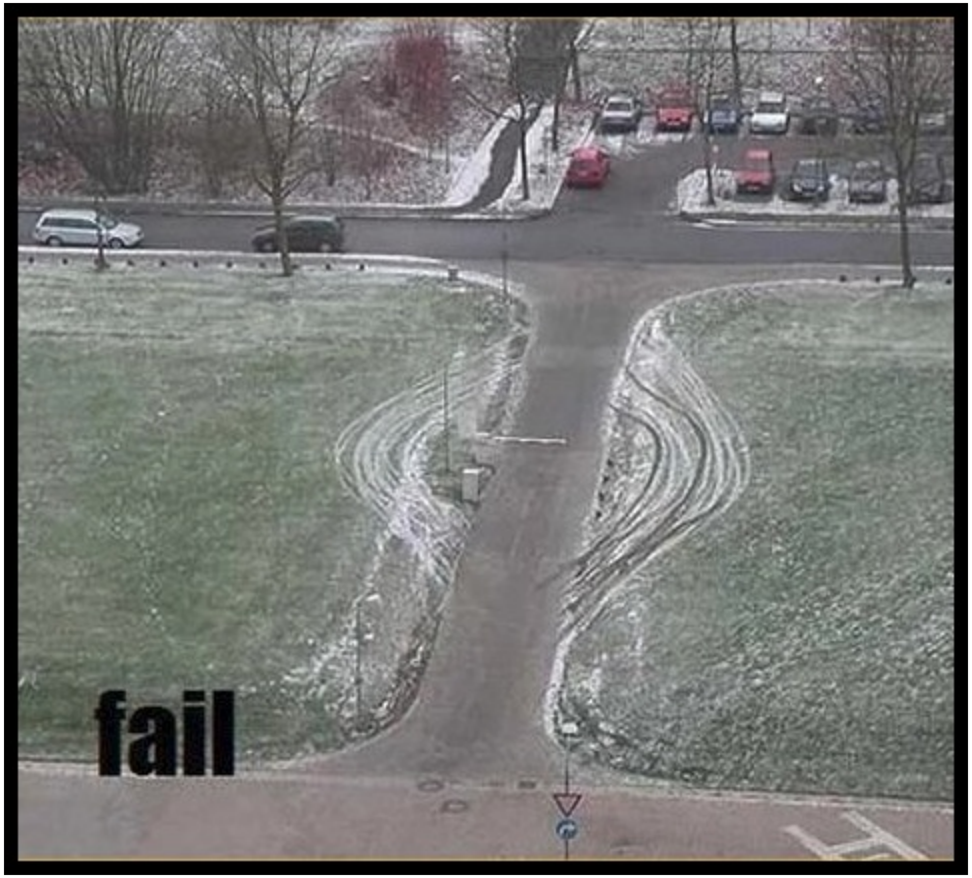
Side Channel Analysis

Chester Rebeiro
IIT Madras

Side Channels



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

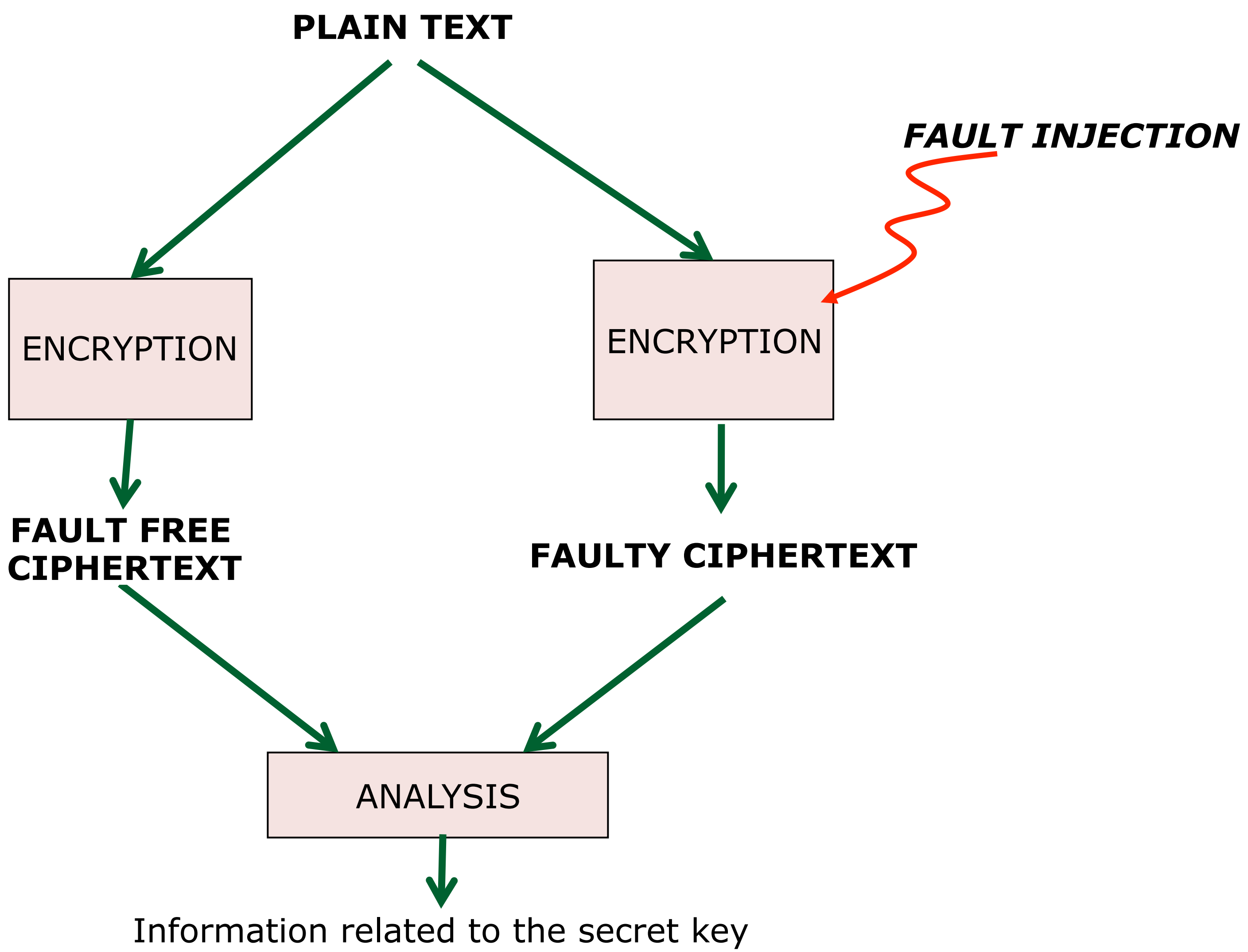
source : Elisabeth Oswald, Univ. of Bristol

Fault Attacks

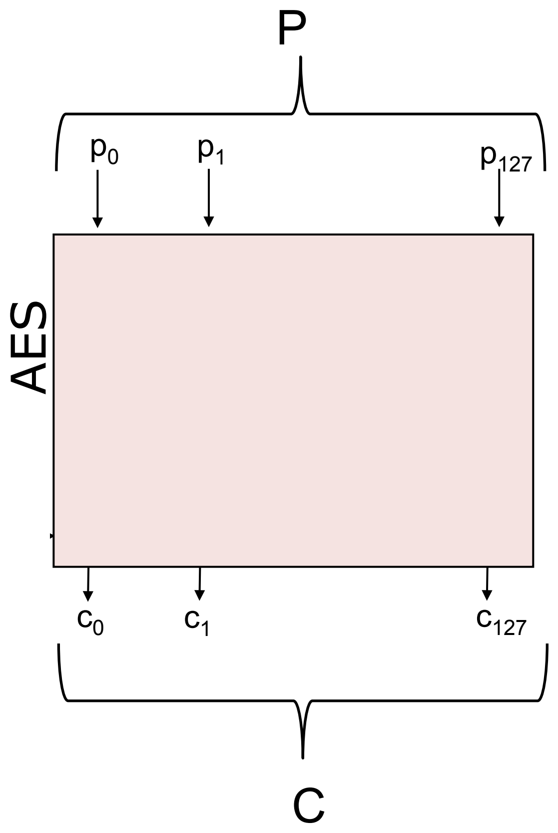
Fault Attacks

- 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
- Rowhammer based fault attacks (2016)

Fault Attacks

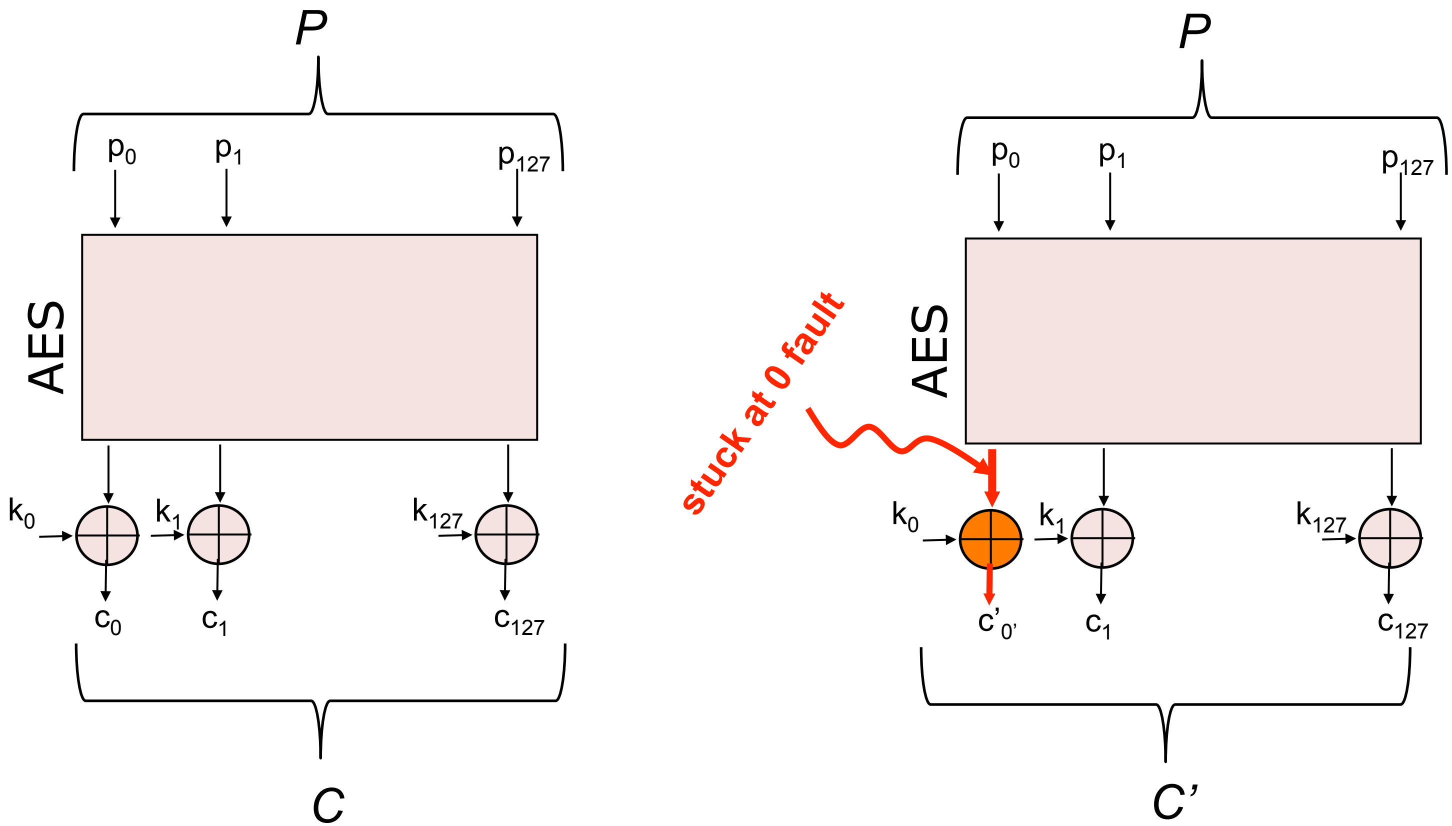


A Simple AES Fault Attack



Last Round Key Addition

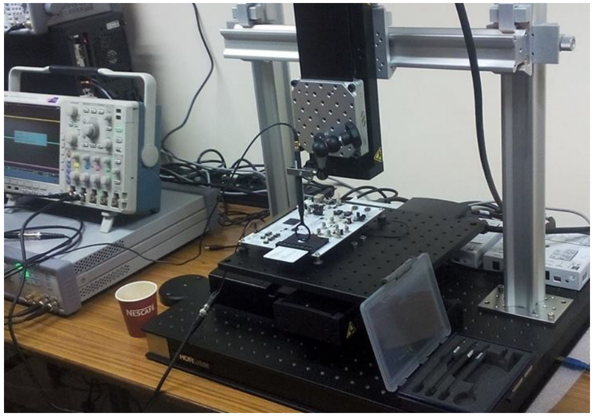
A Simple AES Fault Attack



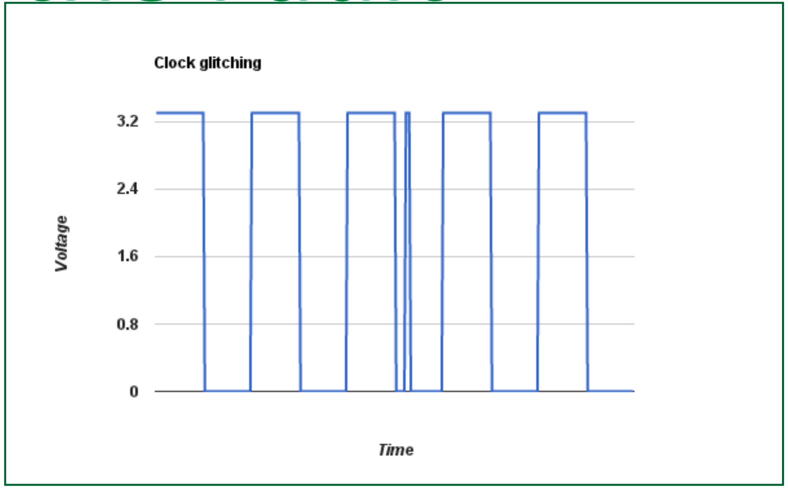
$$k_0 = c_0^{'}$$

Requires 128 faults to recover the complete key …. can we do better!!

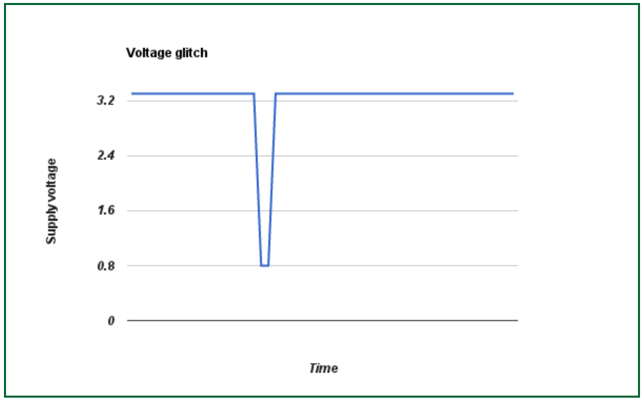
Inducing the Fault



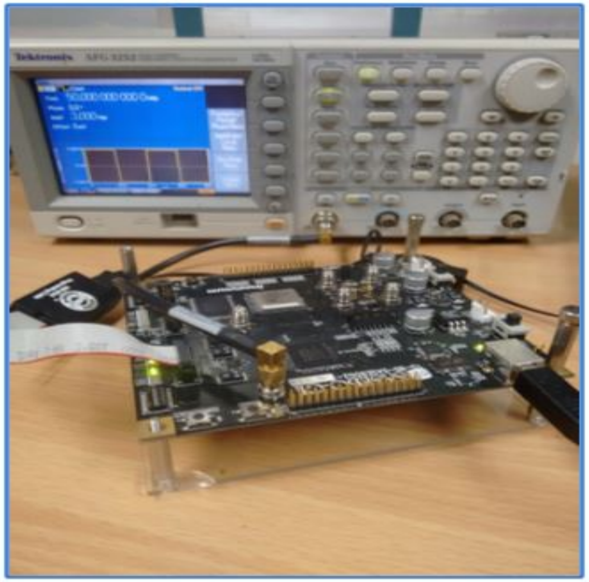
Optical Fault Injection



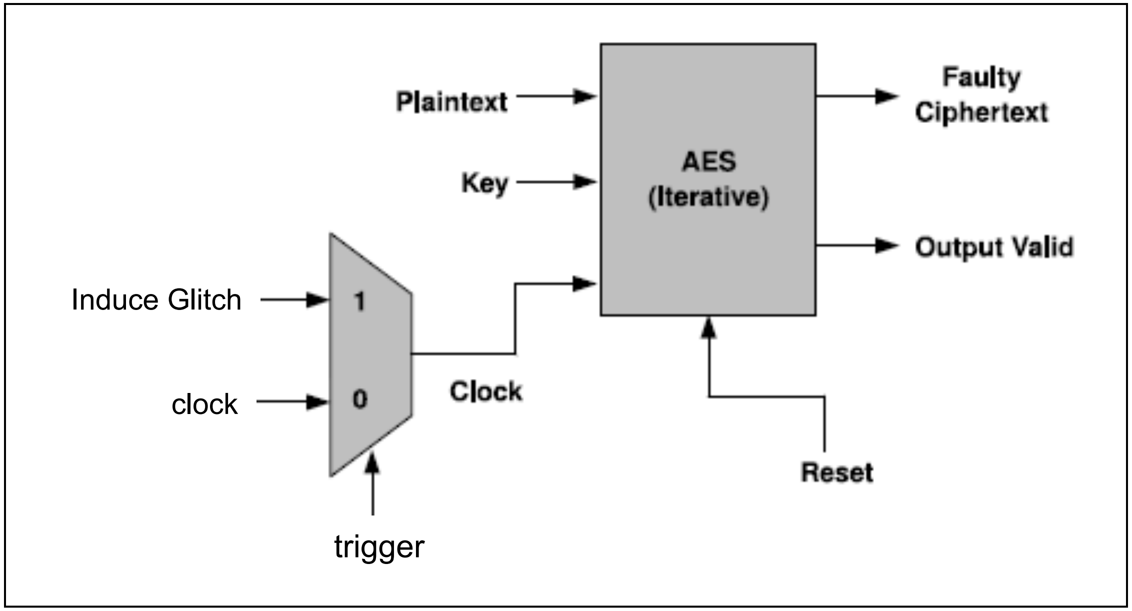
Clock Glitching



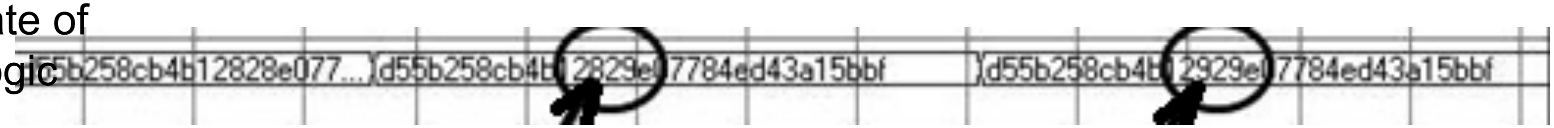
Voltage Glitching



Inducing a Fault in AES



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

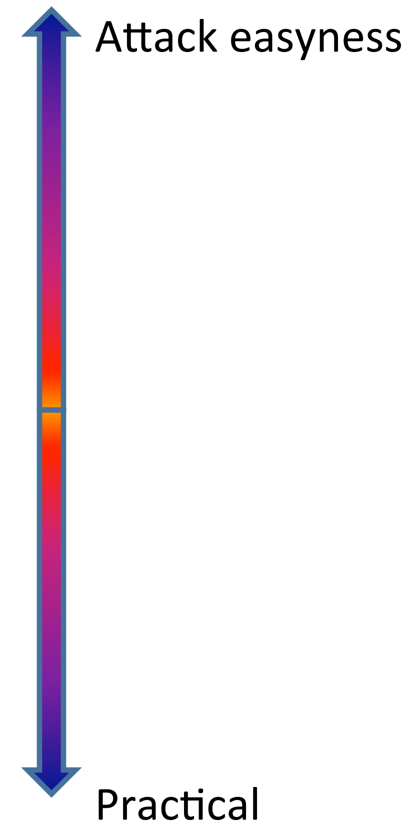
  eg. 8823124345 → 8833124345

- 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 ith bit of a. Thus she would get two ciphertexts:

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

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

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

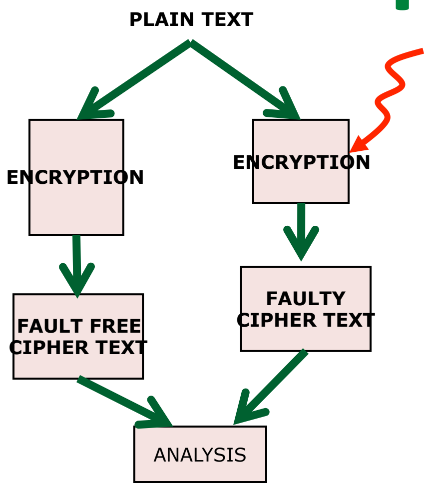
$$\frac{x}{\tilde{x}} = \frac{y^a}{y^{\tilde{a}}} \bmod n = y^{a-\tilde{a}} \bmod n$$

Thus,

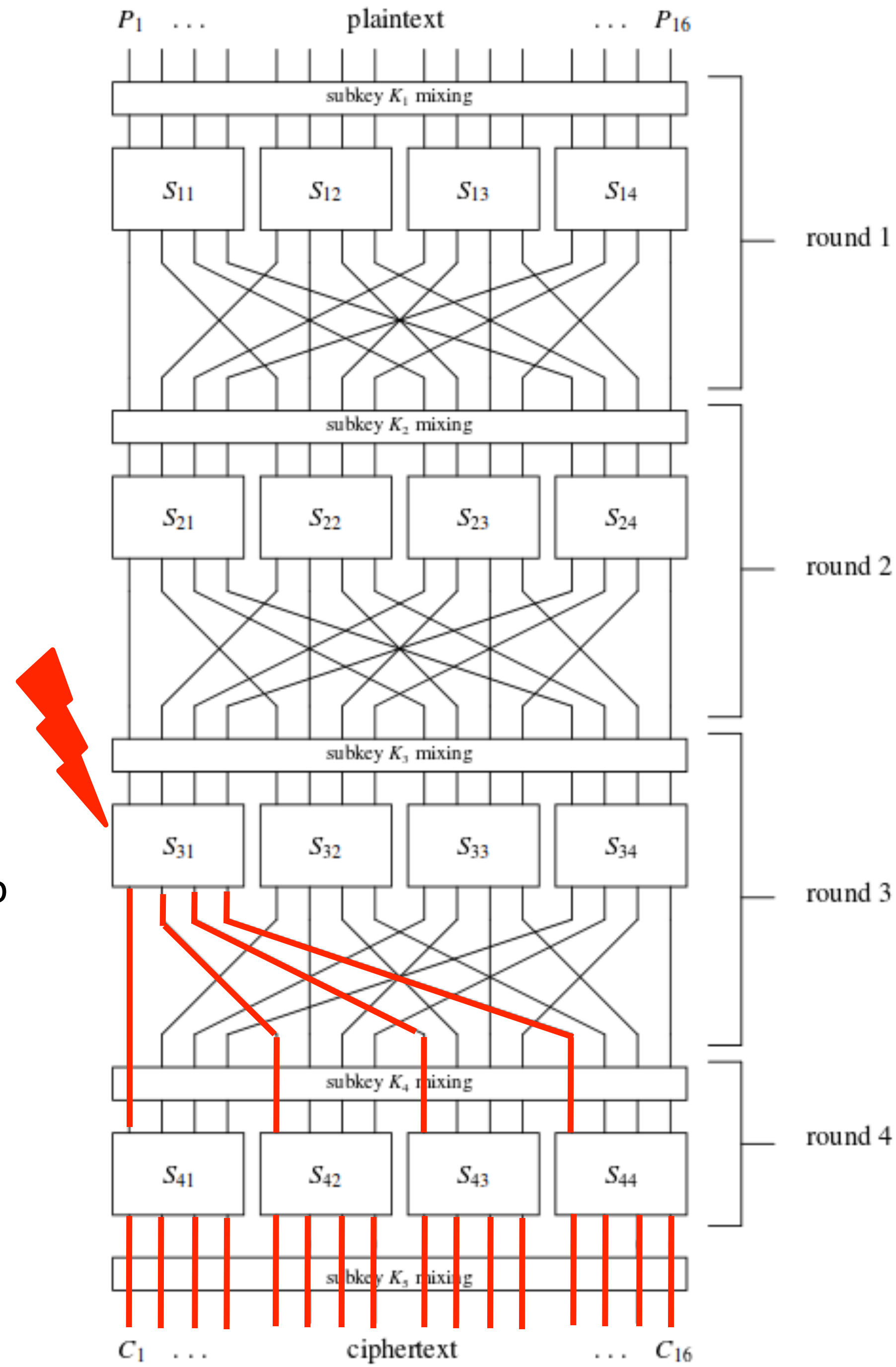
$$\frac{x}{\tilde{x}} = \begin{cases} y^{2^i} & \text{if } a_i = 1 \\ y^{-2^i} & \text{if } a_i = 0 \end{cases}$$

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

What a fault does to a block cipher?



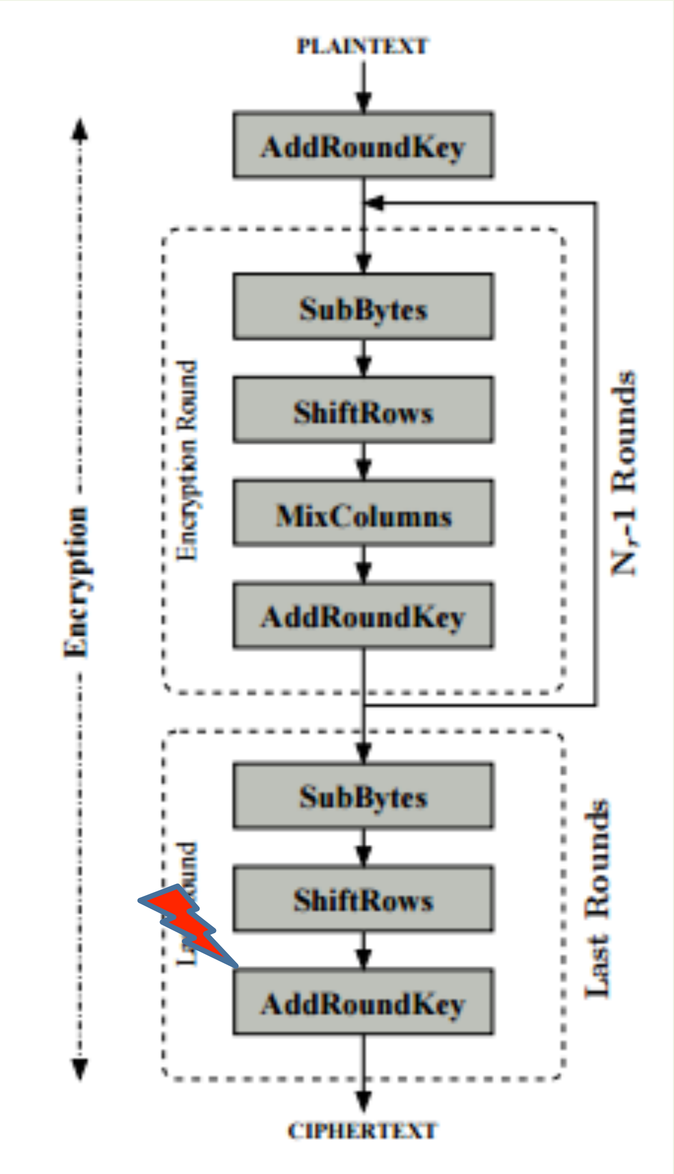
- A fault (generally at the s-box input) creates a difference wrt the fault free encryption
- This difference is propagated and diffused to multiple output bytes of the cipher
- The attacker thus has 2 cipertexts :
  (1) the fault free ciphertext (C)
  (2) the faulty ciphertext (C*)



A Simple Fault Attack on AES

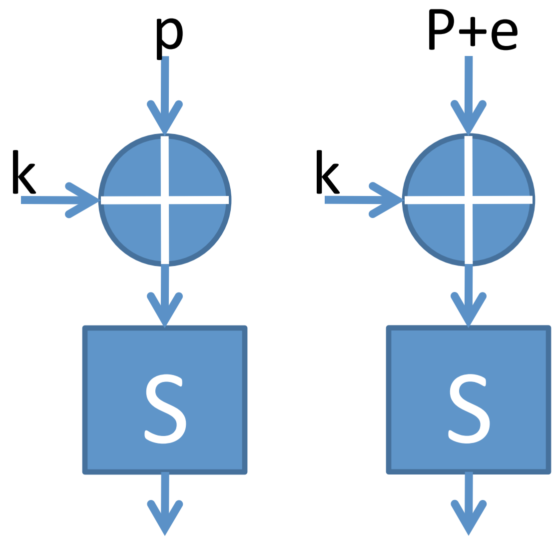
- Let's assume that the attacker has the capability of resetting a particular line during the AES round key addition.

  (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?



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 8 different values:

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

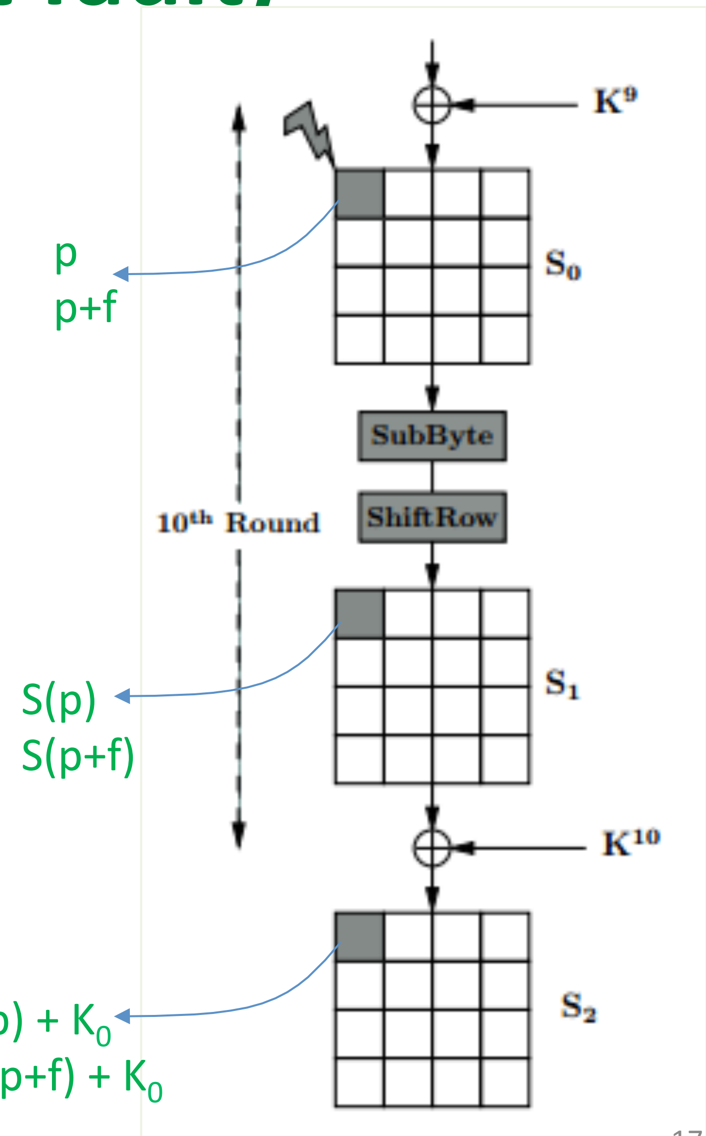
(000010000), …. , (10000000)

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.



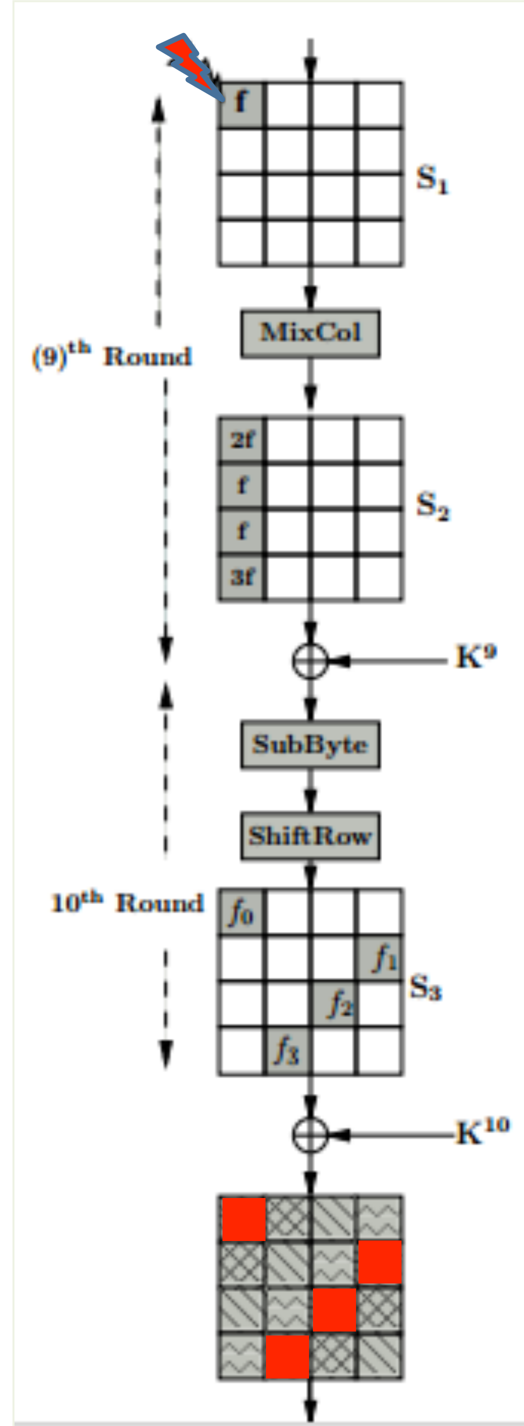
DFA on last round of AES (using a single bit fault)

- 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

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

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



CR

Solving the Difference Equations

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

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

Solving the Difference Equations

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

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 $p(2^8)^5 = 2^{8(5-4)} = 2^8$.

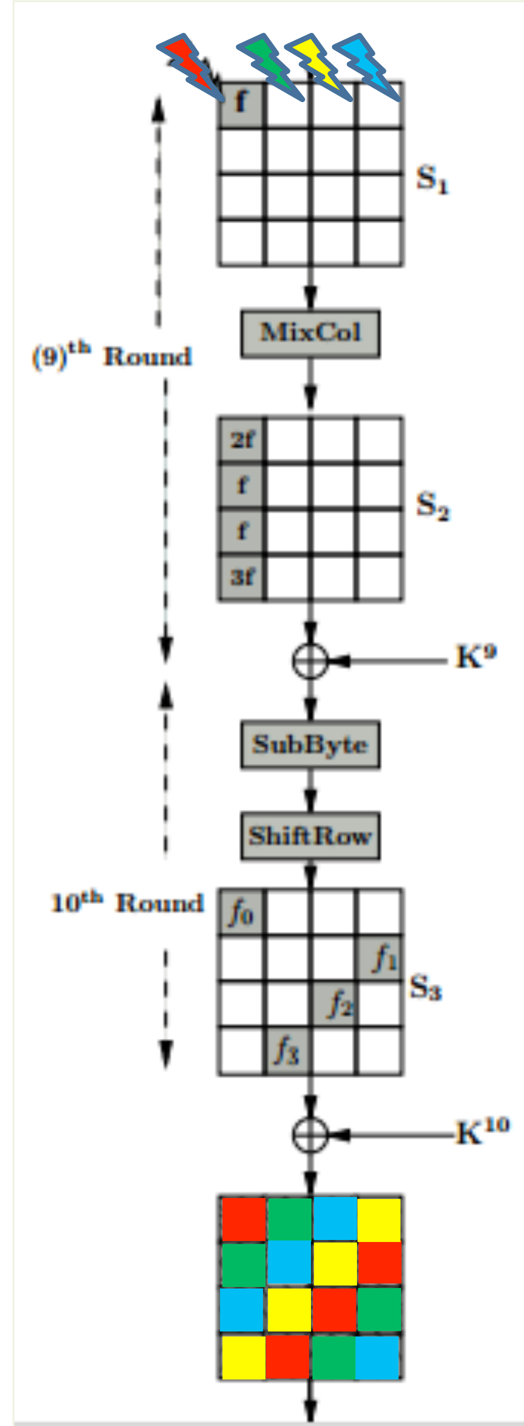
The key space is 2^{32}. From the above, it has reduced to just 2^8.

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

CR

DFA on AES with a single fault

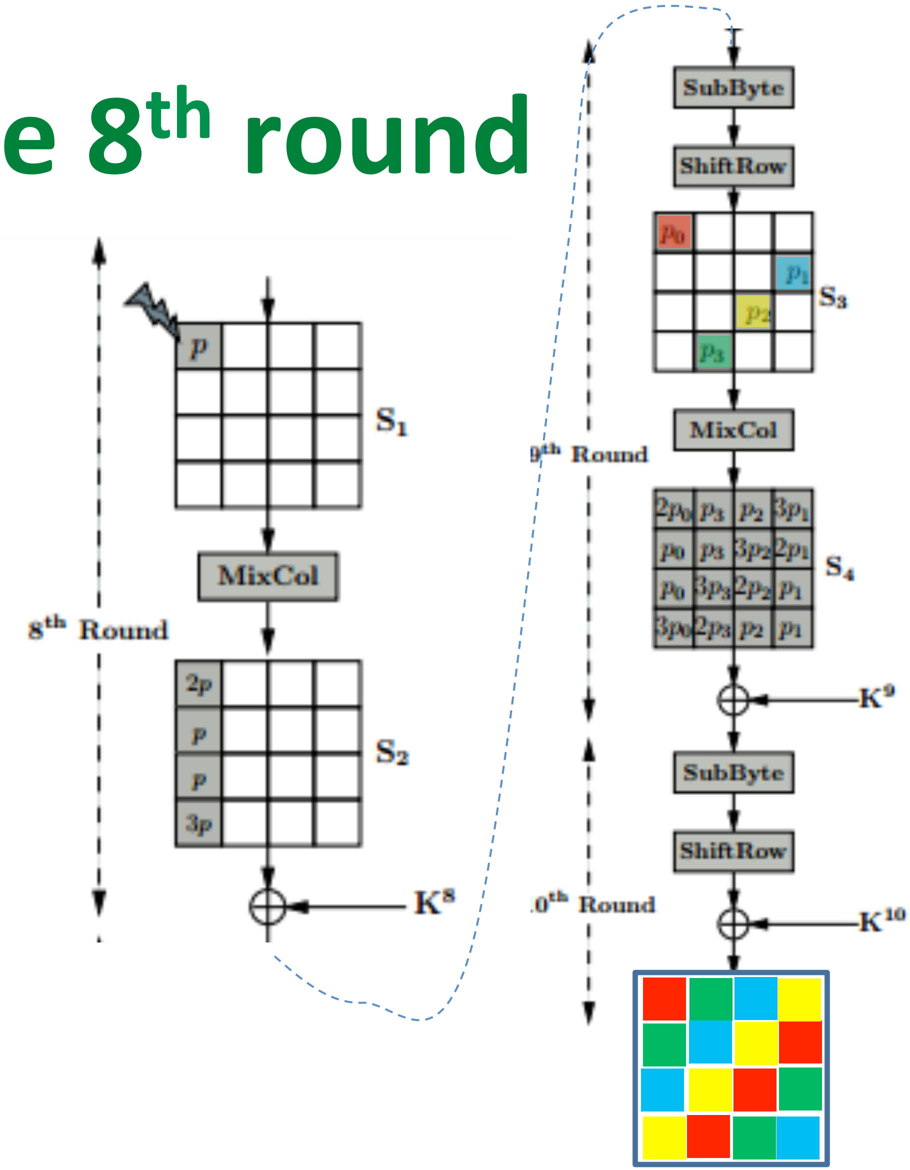
- As mentioned previously, 4 faults are required in the 9th round to reveal the entire key
- Instead of the 9th round, suppose we inject the fault in the 8th round



CR

DFA on AES in the 8^{th} round

- A single fault injected in the 8^{th} round will spread to 4 bytes in the 9^{th} 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



Remote Timing Attacks on RSA

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

1
2

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

where

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

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

3

Derive x from x_1 and x_2

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

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

$$x = x_1 + h \cdot q$$

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

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**

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

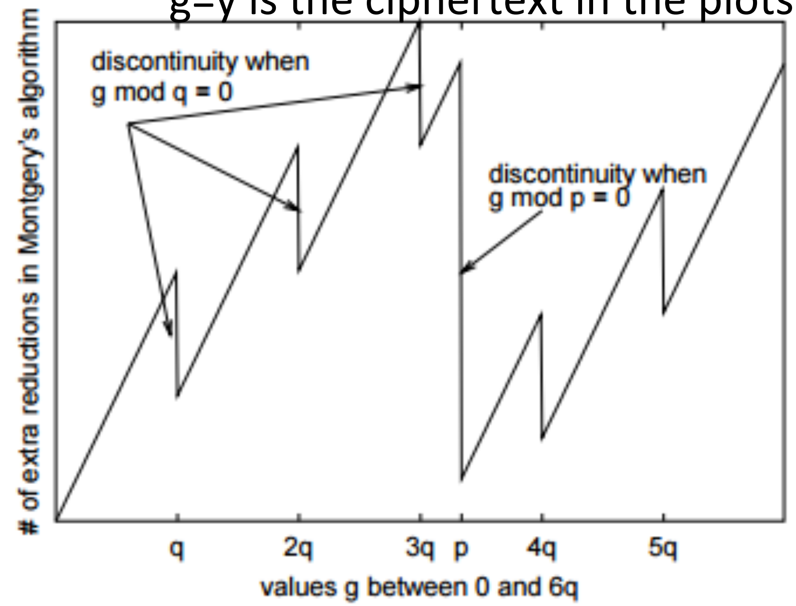
$$\Pr[\text{ExtraReduction}] = \frac{y \bmod q}{2R}$$

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

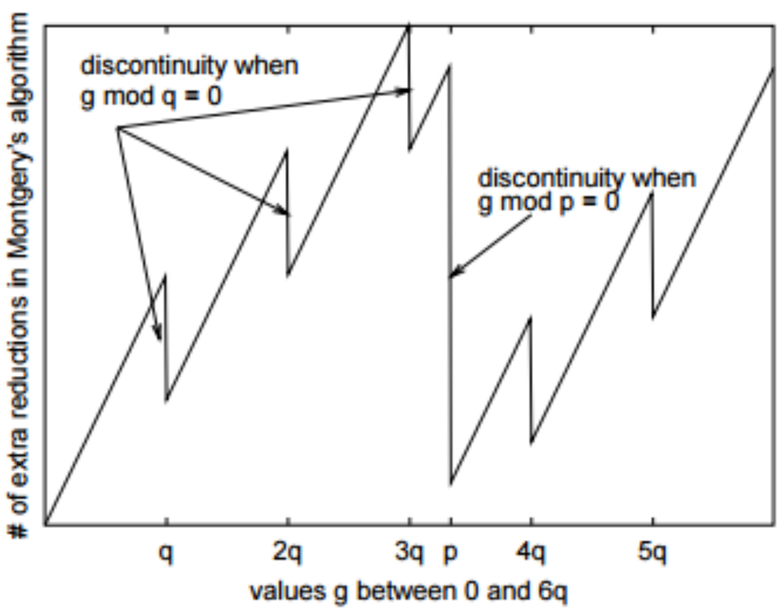
- 30-40% of OpenSSL RSA decryption execution time is spent on integer multiplication
- If multiplicands have the same number of words n, OpenSSL uses Karatsuba multiplication $O(n^{\log_2 3})$
- If integers have unequal number of words n and m, OpenSSL uses normal multiplication $O(nm)$

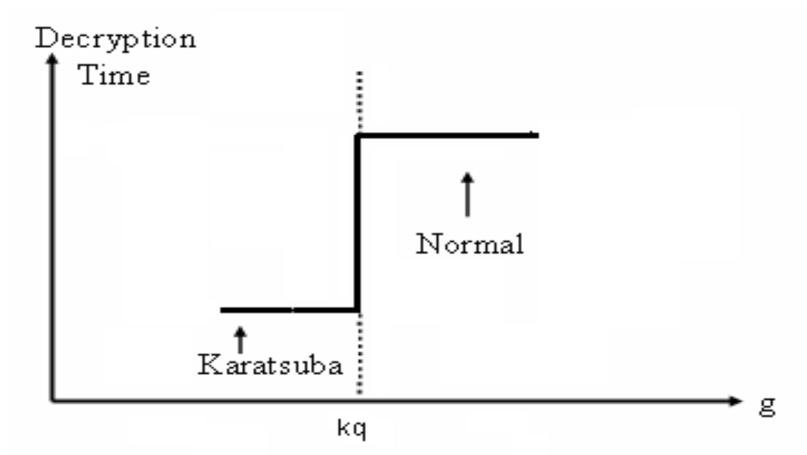
these further cause timing variations…

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

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

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

$$note\ that$$

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

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

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

$$t_0 \ :\ \ DecryptionTime(y_0)$$

$$t_1 \ :\ \ DecryptionTime(y_1)$$

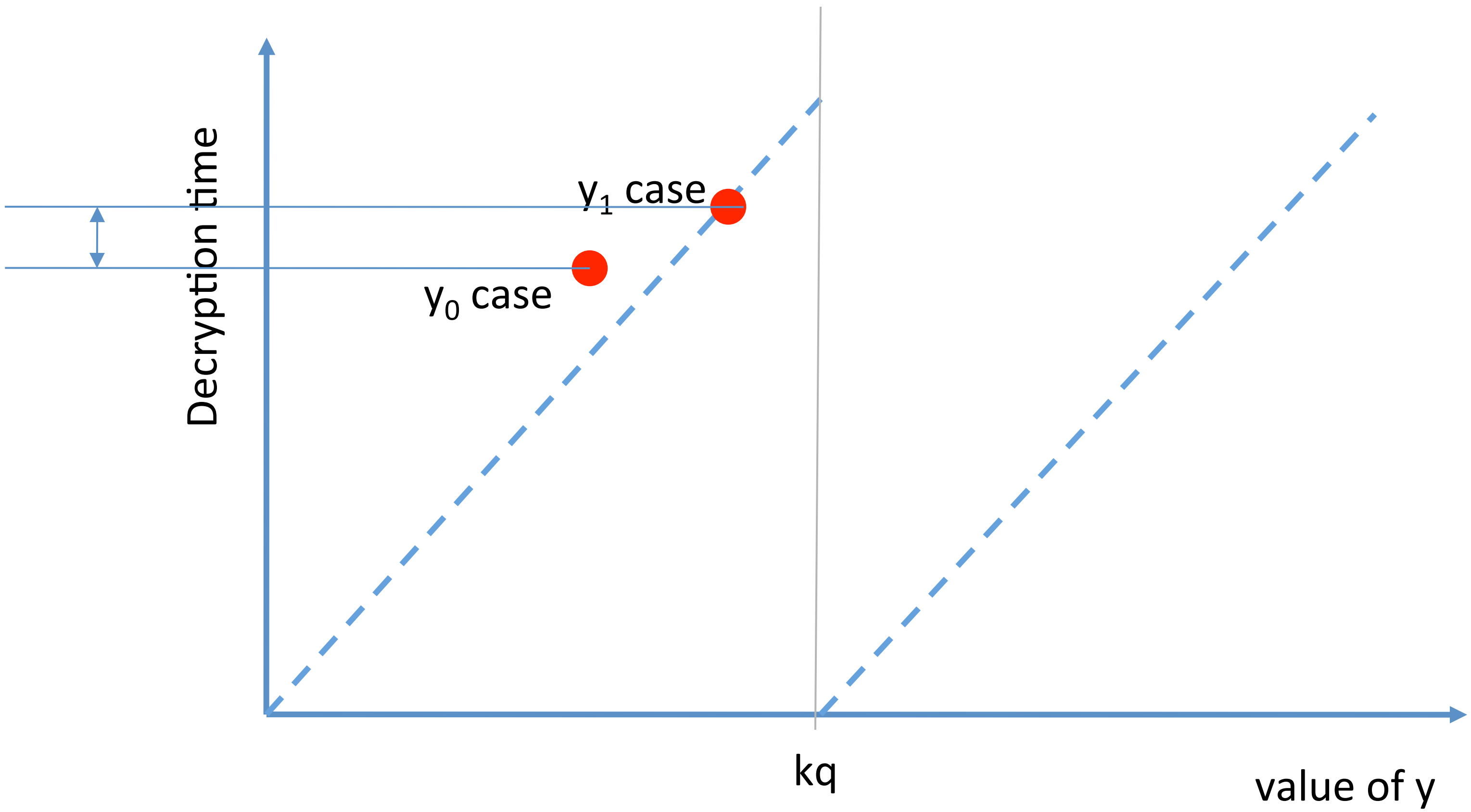
$$3.\ \ If\ \ |\, t_1 - t_0 \,|\ \ \text{is large}\ \ \rightarrow\ \ q_i = 0 \qquad (corresponds\ to\ \ y_0 \le q < y_1)$$

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

What's happening here?

Assume Montgomery multiplier dominates over Integer multiplication

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

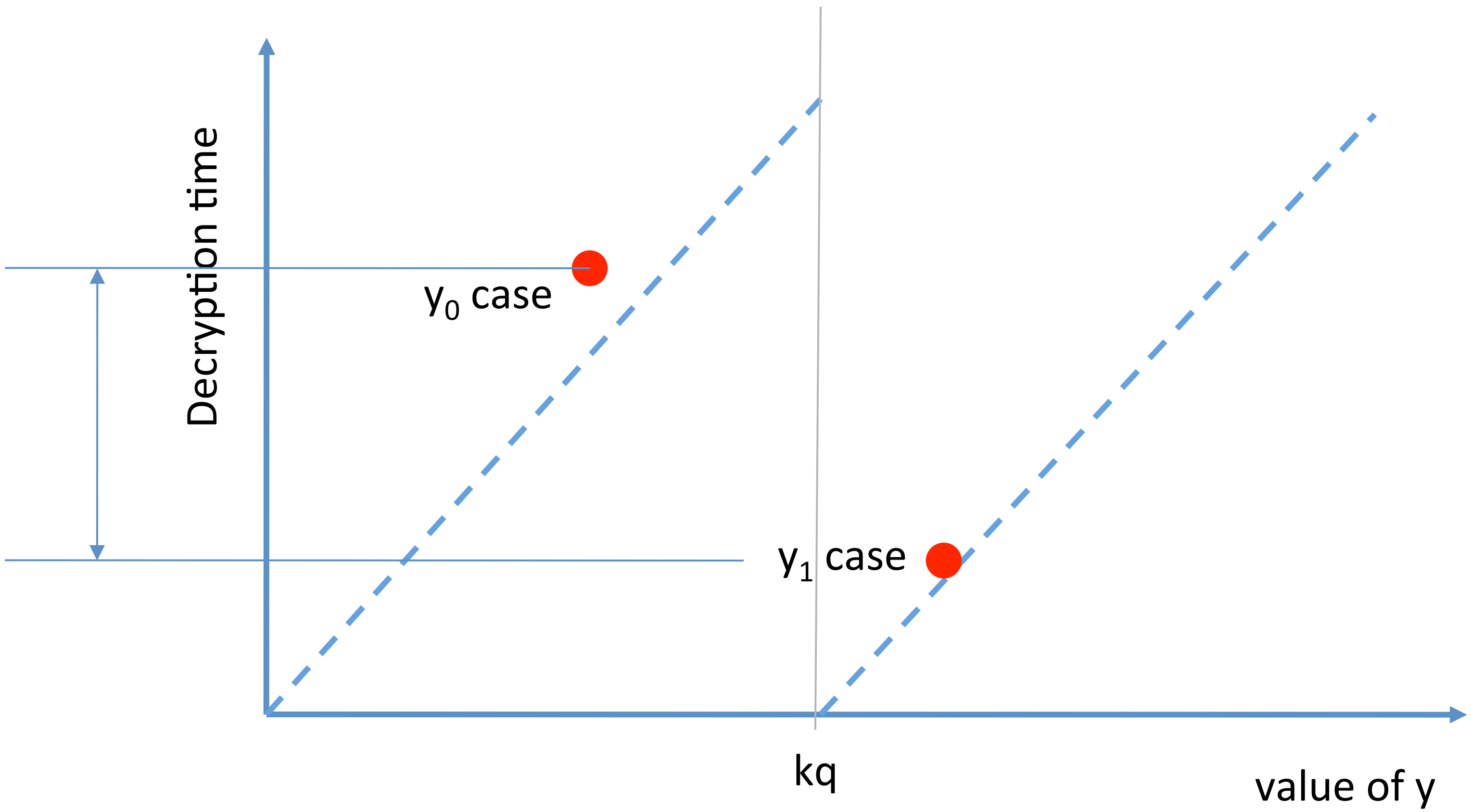


What's happening here?

Assume Montgomery multiplier dominates over Integer multiplication

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

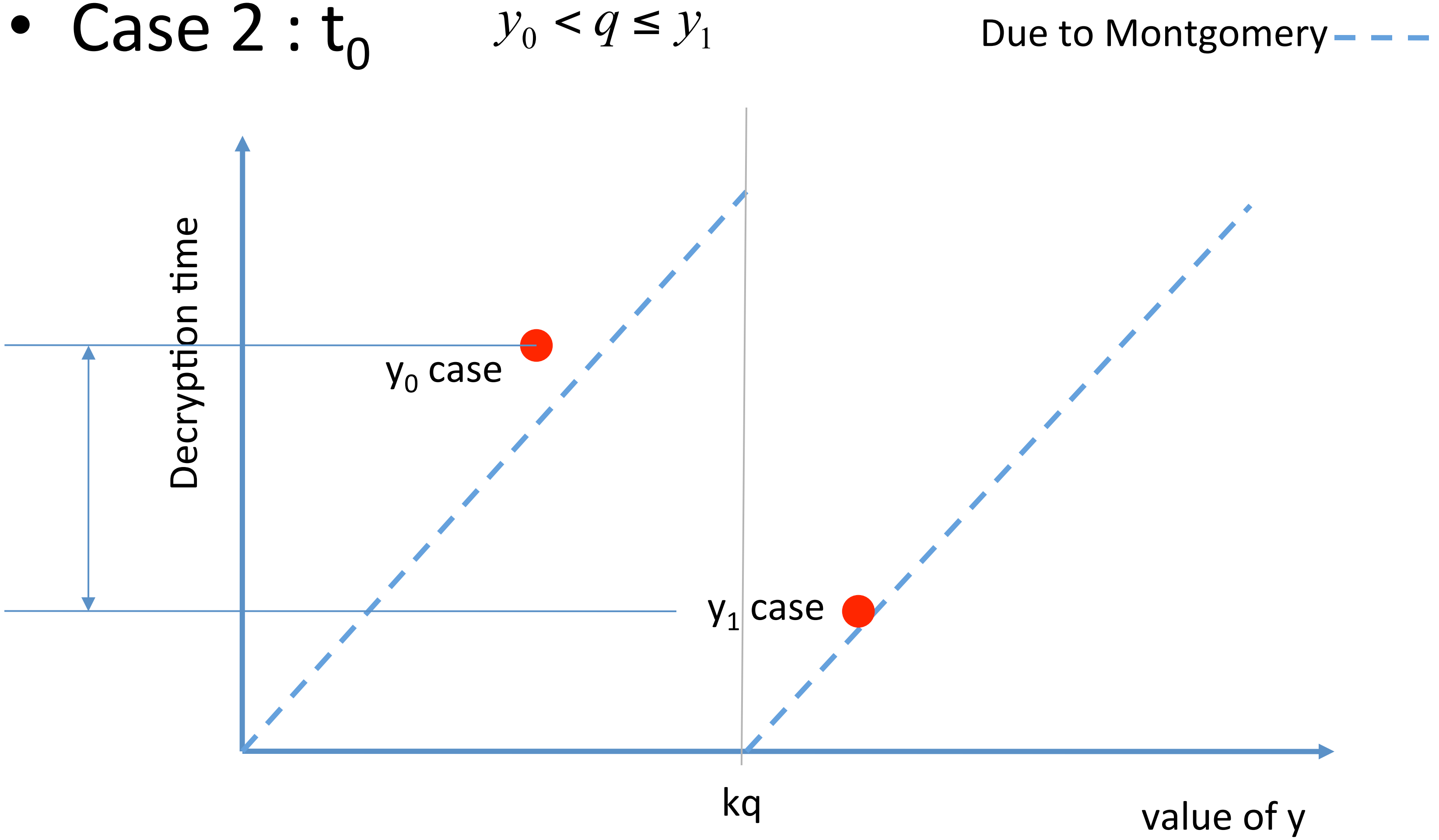
Due to Montgomery



What's happening here?

Assume Montgomery multiplier dominates over Integer multiplication

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

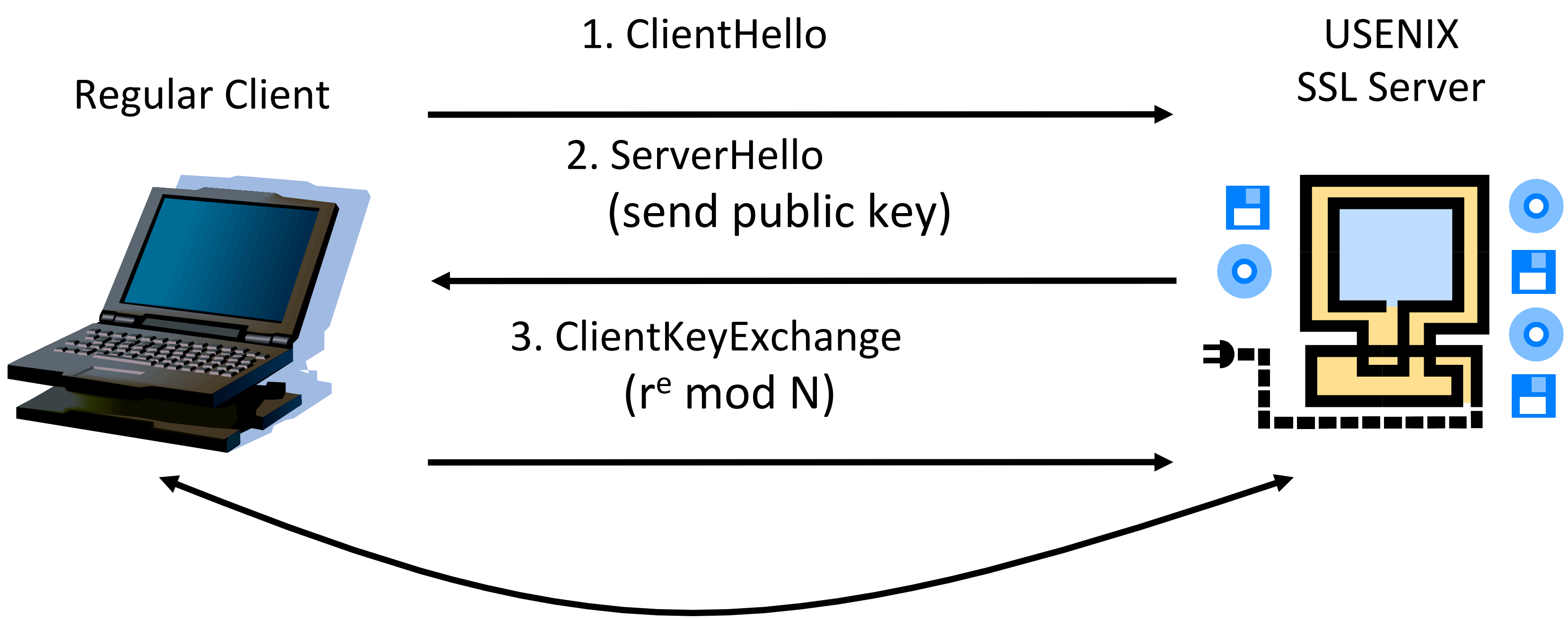


What happens when integer multiplier dominates or Montgomery multiplier?

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