Fault Attacks

- Overview
  - Fault sources
  - Exponentiation
- Attacks on RSA
- Attacks on PRNG

Fault Attacks Overview

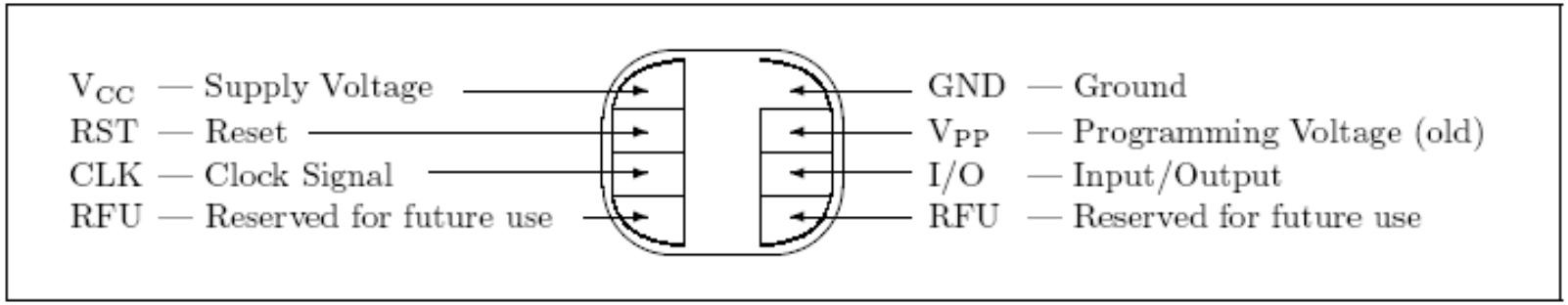
- Adversary induces faults into a device, while it executes a known program, and observes the reaction.
- Idea: make sure an error occurs yielding to a faulty result. If the computation depends on some secret key, a comparison between correct data and faulty data may allow to obtain the secret key.
  - Dan Boneh, Richard DeMillo, and Richard Lipton from Bellcore Labs in Sep 25 1996… (aka Belcore Attack).
- They are real and used to break security mechanisms even before the cryptographic community became aware of them.
  - Pay TV card hackers used rapid transient changes in the clock signal, called clock glitches, to access pay TV channels before 1996.

Inducing Faults in the Physical World

- Cosmic Rays- very high-energy subatomic particles originating in outer space. Research in electrical equipment used in aviation or space travel found that cosmic rays can flip single bits in the memory of an electronic device. The DRAM cells of a typical PC were expected to suffer such a bit ip about once per month. --- not practical tool for adversary.
- α-, β-, and X-rays. α-, β-rays are not practical . Although standard commercial X-ray sources, e.g., an airport baggage scanner, produce X-rays which have not enough energy per particle to interact with DRAM circuitry, there are high-energy \hard" X-ray sources, which \might possibly do the job“
  - All three approaches can flip single bits in memory if applied successfully, however, targeting a specific bit is very difficult.

Physical Faults (cont.)

- Heat / Infrared Radiation: infrared radiation coming from a simple 50-watt spotlight clip-on lamp together with a variable AC power supply shown to successfully to induce faults into a desktop PC.
  - This attack succeeded to induce single bit flips for temperatures between 80 and 100 Celsius. However, the experiment also showed that unless finely tuned, the attacks often cause the operating system to crash
- Power Spikes.
  - a smartcard must tolerate a certain variation in the power supply VCC e.g., 10% of the standard voltage of 5V. However, if the variation is significantly higher than 10%, the card is no longer required to work properly. ISO 7816-2 spec for smartcards:



Faults (Cont).

- Clock Glitches:
  - Hardware must also work properly with deviations of clock rise and clock fall times of 9% from the standard period clock cycle. Smartcards are usually provided with a 3.5 MHz signal.
  - Since the adversary may replace the card reader by laboratory equipment, he may provide the card with a clock signal, which incorporates short massive deviations from the standard signal, which are beyond the required tolerance bounds. Such signals are called glitches.
  - clock glitch attacks did not emerge in the scientific community but in the pay-TV hacker community

Countermeasures

- Sensors and filters, which aim to detect attacks, e.g., using anomalous frequency detectors, anomalous voltage detectors, or light detectors
- use redundancy, i.e., dual-rail logic, where memory is doubled, doubled hardware, capable of computing a result twice in parallel, or doubled computations, where a computation is performed twice on the same hardware
- use of a randomized clock to achieve an unstable internal frequency, bus line and memory encryption, dummy random cycles, and active and passive shields protecting the internal circuits.---- we will not cover these!!

Characterization of Faults

- Control on the fault location: 1-no control, 2-loose control (a selected variable can be targeted), and 3-complete control (selected bits can be targeted).
- Control on the timing: 1-no control, 2-loose control (a selected variable can be targeted), and 3-complete control
- The number of bits faulted: 1-single faulty bit", 2- few faulty bits" (e.g., a byte), and 3- a random number of faulty bits" (bounded by the length of the variable).
- Fault type : also known in the literature as the “fault model “it describes the efect of a fault on each individual bit.
- Success probability: For example, some physical attacks might have a greater probability of resetting a bit than of setting that bit.
- Fault duration: 1-transient faults, 2-permanent faults, 3-destructive faults.

Fault Models

Let $B = \{ b_1, \ldots, b_n \}$ be an arbitrary set of bits stored in memory.

- Bit flip fault: all bits of B are set to their complementary values
- Random fault. Then the bits of B are set to random values
- Bit set or reset fault. Then the bits of B are set to chosen values
- Single bit fault: addition or subtraction of a single bit so that a variable X is changed to $X \rightarrow \tilde{X} = X \pm 2^k$ for $0 \leq k \leq l(X)-1$

The bit position k is assumed to be chosen according to the uniform distribution and subsequent choices are assumed to be i.i.d..

Right2Left Repeated Squaring

Input: a message $M \in \mathbb{Z}_N$, a secret RSA key $1 \leq d$, where $l(d)$ denotes the binary length of d, i.e., the number of bits of d, and an RSA modulus N

Output: $M^d \bmod N$

\# **init**

1 Set $y := 1$

2 Set $z := M$

\# **main**

3 For k from 0 to $l(d) - 1$ do

4 If $d_k = 1$ then set $y := y \cdot z \bmod N$

5 Set $z := z^2 \bmod N$

6 **Output** y

Fault Model

[Boneh-DeMillo-Lipton01]

- Single Bit Fault Model
- Adversary runs a polynomial number of times while inducing faults.
- Knows both the input messages and the faulty signatures $(M_i, \tilde{S_i})$
- The analysis of the attack assumes that the messages are chosen i.i.d. from $\mathbb{Z}_N$ according to the uniform distribution.
- Targets the intermediate variable *y* used on the right hand side in Line 4.

Analysis

Let M_v be an input to Algorithm

$S_v = M_v^d$ be the correct result, and let $e(y) = \pm 2^b$, with $0 \leq b < l(N)$, be the error.

the faulty result S_v can be described as

$$\tilde{S}_v \equiv \left(\prod_{j=0}^{i-1} M_v^{2^j d_j} \pm 2^b \right) \cdot \prod_{j=i}^{n-1} M_v^{2^j d_j} \equiv S_v \pm 2^b \cdot \prod_{j=i}^{n-1} M_v^{2^j d_j} \bmod N$$

$$\Rightarrow S_v \equiv \tilde{S}_v \pm 2^b \cdot M_v^w \bmod N, \qquad \text{where} \quad w := \sum_{j=i}^{n-1} d_j 2^j$$

$$\Rightarrow M_v \equiv \left(\tilde{S}_v \pm 2^b \cdot M_v^w \right)^e \bmod N.$$

Note that adversary does not need to know
the correct signatures S_v in order to describe the faulty values.

Attack

recover all bits of *d* in blocks of *m* bits starting from the most significant bit d_{n-1} where m is a parameter, such that 2^m is an acceptable amount of offline work.

- Find values *w* and $\pm 2^b$, such that Equation is satisfied. then the bits of w correctly describe the corresponding bits of *d*, and $\pm 2^b$ correctly represents the error

- the adversary needs to collect at least c = (n/m) log(2n) many faulty signatures to guarantee a success probability of 1/2.

- Tradeoff between m and c.
  i.e., between the acceptable amount of offline work and the required number of faulty signatures.

CRT-RSA

Algorithm 3.1: CRT-RSA Algorithm

Input: $m \in \mathbb{Z}_N$ the message
Output: $s = m^d \bmod N$
In Memory: the secret RSA key d with $d_p = d \bmod (p-1)$ and $d_q = d \bmod (q-1)$, the RSA modulus $N = p \cdot q$, and both primes p and q

1 $S_p := m^{d_p} \bmod p$
2 $S_q := m^{d_q} \bmod q$
3 $s := \mathrm{CRT}(S_p, S_q)$

$$s = S_p + X \cdot (S_q - S_p) \bmod N, \qquad \text{where } X = p \cdot (p^{-1} \bmod q).$$

CRT-RSA and the Bellcore Attack

- Why CRT based implementation?
  - Exponentiation using CRT is much faster than repeated squaring modulo *N*
- $S = x^d \bmod N$. For efficiency most implementations exponentiate as follows: using repeated squaring they first compute
- $S1 = x^d \bmod p$, and $S2 = x^d \bmod q$
- $S = aS1 + bS2 \bmod N$
- $S1 = x^d \bmod p = x^{d \bmod p-1} \bmod p$ → d is O(N) but d mod (p-1) is O(p) → ½ gain. Also the intermediate values are O(p) not O(N).
  → quadratic time multiplication gain ¼ of working with N
  → S1 takes 1/8 of the time for S → S1 and S2 4 times faster.

The Bellcore Attack

- S is a valid signature, $\hat{S}$ faulty signature of x
- Assumption: fault occurs in the computation modulo only one of the primes (say p).

$$(s - \tilde{s}) \equiv 0 \bmod q \quad \Leftrightarrow \quad q \mid (s - \tilde{s}) \quad \text{but}$$
$$(s - \tilde{s}) \not\equiv 0 \bmod p \quad \Leftrightarrow \quad p \nmid (s - \tilde{s}).$$

- Then q=gcd(S-$\hat{S}$, N)

Remedies

- Given the public key e, check whether a computed signature s satisfies $s^e \equiv m \bmod N$
- Compute a result twice, say s1 and s2 using the same data. If the two results are equal, s1 is returned, otherwise, an error is detected.
- Shamir's remedy: choose a small random integer r of about 32 bits, which can then be used to verify intermediate results with a high probability of detecting all faults

Shamir's Remedy

Input: $m \in \mathbb{Z}_N$ the message
Output: $s = m^d \bmod N$
In Memory: the secret RSA key d, a random 32-bit integer r
both primes p and q

1 Set $d_{pr} := d \bmod \varphi(pr)$
2 Set $d_{qr} := d \bmod \varphi(qr)$
3 $S_p := m^{d_{pr}} \bmod pr$
4 $S_q := m^{d_{qr}} \bmod qr$
5 If $S_p \not\equiv S_q \bmod r$ then return FAILURE
6 $s := \mathrm{CRT}(S_p, S_q)$

Fault Attack to PRNG [Zheng, Matsumoto'96]

- Breaking ElGamal Signature
- Recall: based on hardness of computing discrete logs over a large finite field
  - p: large prime
  - g: an integer in [1,….,p-1] with order p-1 mod p
  - x_a secret key (integer chosen randomly from [1,…p-1] s.t. x_a does not divide p-1
  - y is public key such that $y=g^{xa} \bmod p$
  - Signature on message m is composed of two numbers:
    - $r= g^z \bmod p$ where z is a r.# that does not divide p-1
    - $S= ((\text{hash}(m) - x_a.r)/z) \bmod p-1$
  - Given (m, r, s) if $g^{\text{hash}(m)} = y^r r^s \bmod p$ then signature is valid

Attack

- Consider PRNG implementation in 1- software, 2- hardware.
- Attack on software PRNG:
  - There is a special register S_info used to generate a new r#
    - Fetch content on S_info
    - Calculate r# z from S_info
    - Update S_info
  - Identify the location of S_info and apply physical stress
    - → force the info in S_info to temporarily a constant (say all 1s)
  - while under pressure ask for a signature for m
  - Given m and ElGamal signature (r,s)
    - Calculate r# z from all 1's using the public PRNG
    - xa= (hash(m) –s z)/r mod (p-1)

Attack

- Attack on build in Hardware PRNG:
  - Again force the info in S_info to temporarily a constant (say all 1s)
  - When the hardware is exposed to "abnormal" physical environment then PRNG produces *predictable output* z
  - If attacker know z then it is the same case as before
    - while under pressure ask for a signature for m
    - Given m and (r,s)
      - Calculate r.# z from all 1's using the public PRNG
      - xa= (hash(m) –s z)/r mod (p-1)
  - If not then get signature on two messages m1 and m2 (note z is the same since stress is maintained
  - Algebraic manipulation gives x_a away

Extracting the secret

- Two signatures with the same z value

$$r = g^z \bmod p, s1 = \frac{hash(m1) - zr}{z} \bmod (p-1)$$

and

$$r = g^z \bmod p, s2 = \frac{hash(m2) - zr}{z} \bmod (p-1)$$

- Then first obtain z

$$z = \frac{hash(m1) - hash(m2)}{s1 - s2} \bmod (p-1)$$

- And then x_a

$$x_a = \frac{hash(m1) - s1z}{r} \bmod (p-1)$$

The END