Applied Security

Fault Attacks on RSA (2)

A whole Zoo of attacks

- Hardware: i.e. an attacker targets the hardware itself and/or ‘physical’ inputs (such as the input voltage, or clock supplied)
- Software: i.e. an attacker targets the software that is running using ‘prepared’ inputs (such as too long inputs, or incorrect inputs)

- Today is about hardware attacks, we first have a look at what can be `faulted’ and how, and then at exploitation and prevention.

Classification of fault attacks

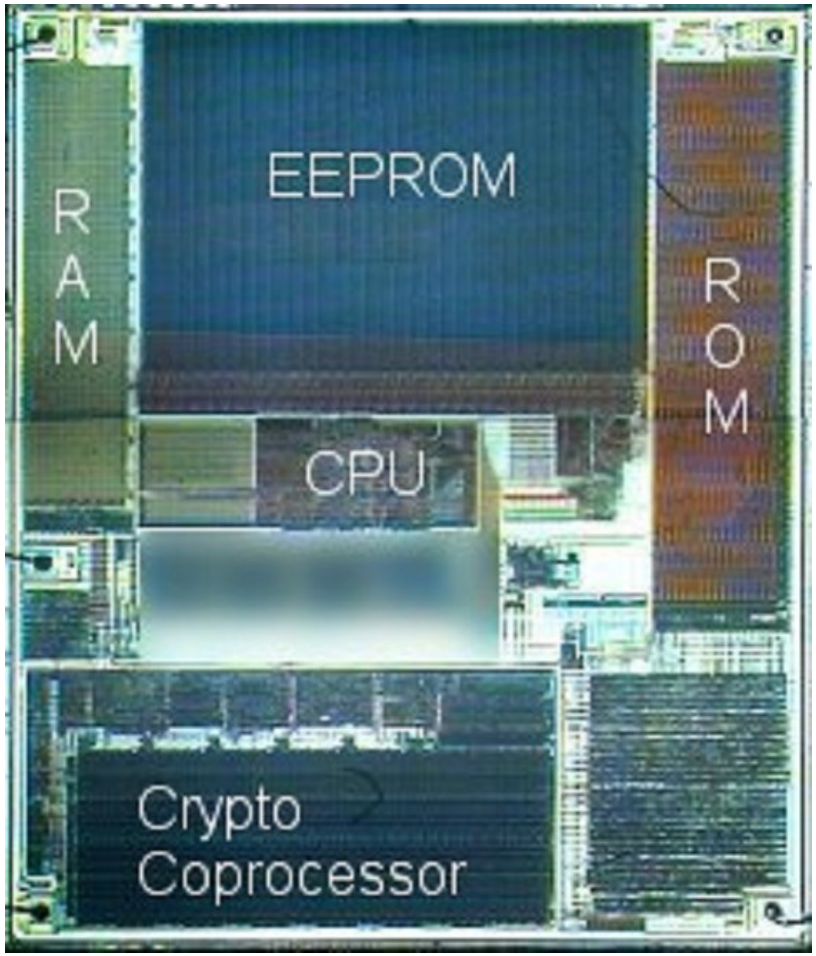
Fault attacks:

- Active, ~~passive~~
- Invasive, semi-invasive, non-invasive

Cost and expertise increase when moving from non-invasive to invasive attacks.

Working principle of fault attacks

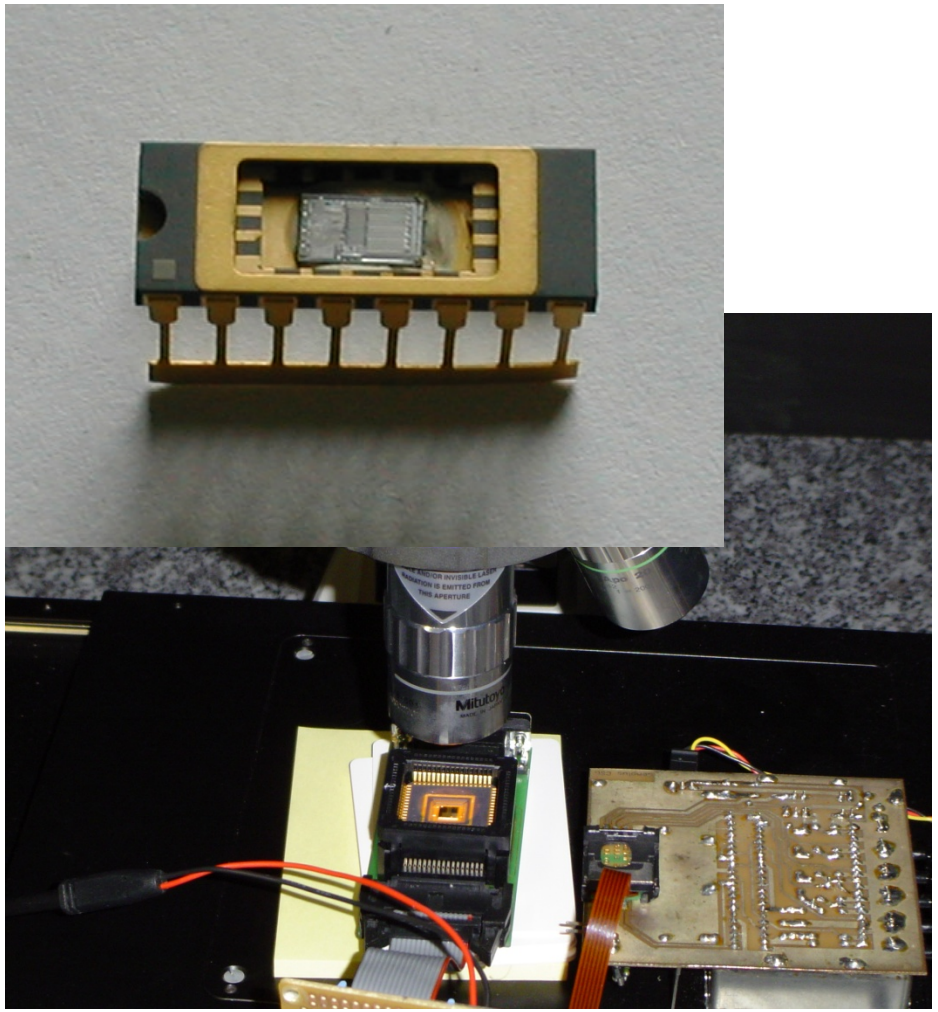
- Preparation (optional)
  - Removing device from package
  - Removing layers, drilling holes
  - Addition/restoration of pins/input lines
- Injection of the fault
  - Via standard inputs
  - By manipulating memory cells/banks, busses
    - Laser
    - Focused ion beam
- Exploitation of faulty outputs



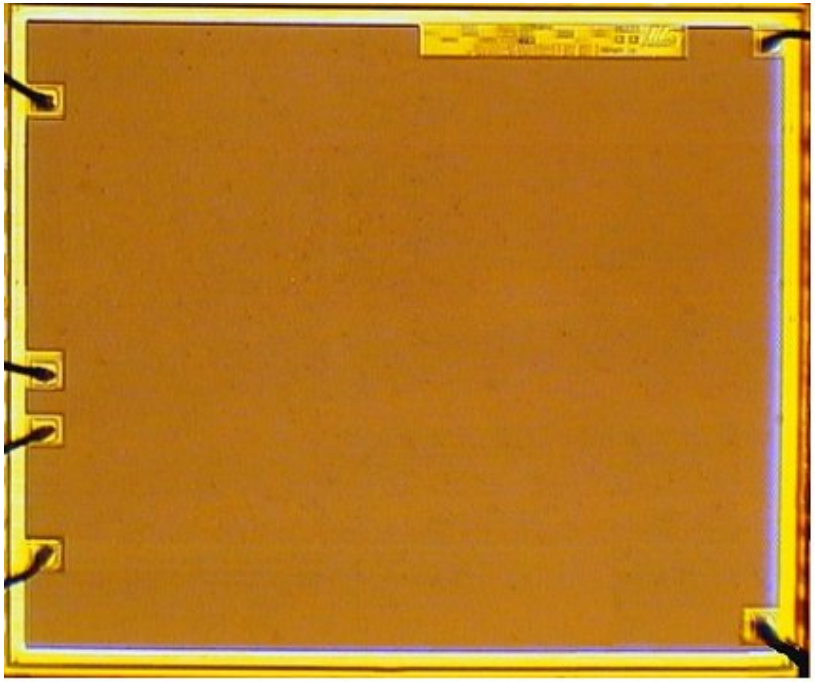
Practical fault attacks: preparation for gaining easy access for invasive attacks

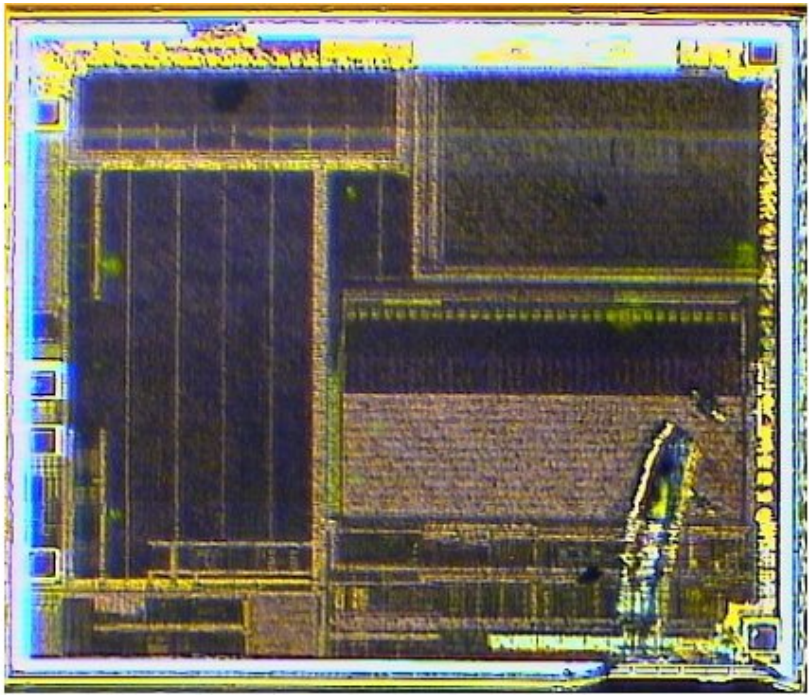
- A smartcard can be removed from the card and glued into another package (requires new bonding wires).

- An alternative being to modify a card reader such that an attacker has access to the back of a chip.



Preparation: modern smart cards come with a protective shield/mesh





- Shield/mesh detects tampering and triggers deletion of keys
- Must drill through mesh without being detected

Other types of cryptographic hardware

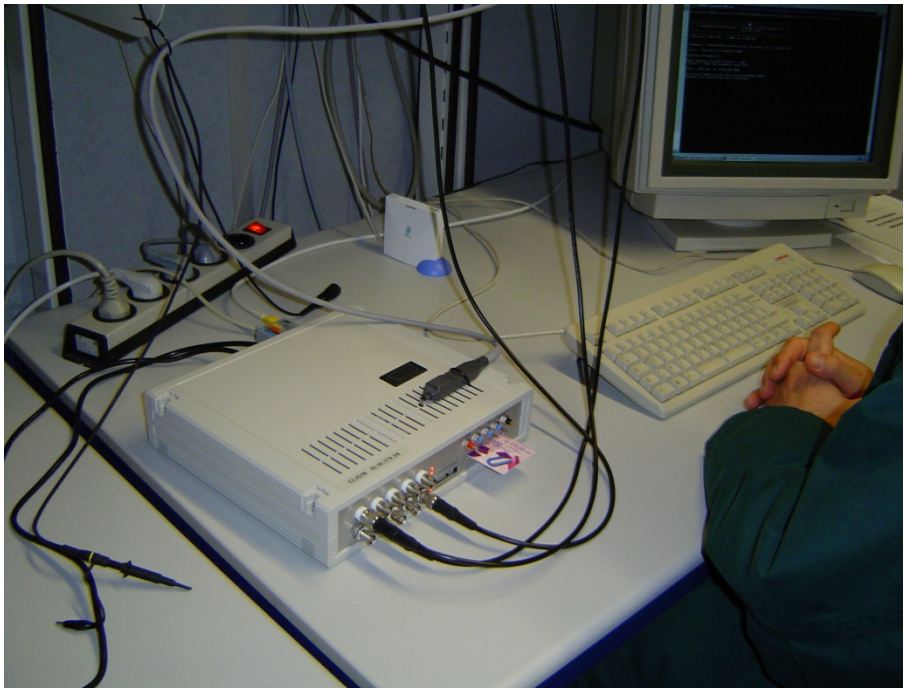
- Provides at least the same protection mechanisms as we listed already for smart cards
- However, since there is more space, power and memory available, more fancy tricks can be applied
  - Tamper proof package

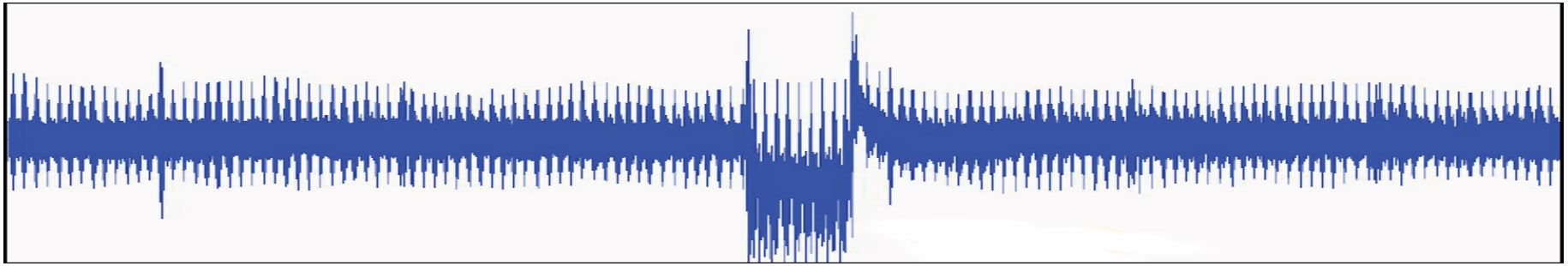
Fault injection: glitches

- Via the power supply
  - Works only in old or non-security specific devices
  - Very old smart cards had a programing voltage supply too
- Via the clock supply
  - Works only in old or non-security specific devices

- In both techniques one hopes to produce a fault and not destroy the device!
  - Can be combined with SPA/DPA to target certain instructions

Equipment for glitches





Fault injection: test circuits

- Manufacturers test circuits led to other attacks on smartcards. Such test circuits are used during the testing phase after the fabrication of the smartcard. After the testing phase, the test circuits are disconnected from the microprocessor. An attacker has to find and repair the disconnected wires to use the test circuit.
- Such circuitry can in general be used to read out data and his hence very critical to security

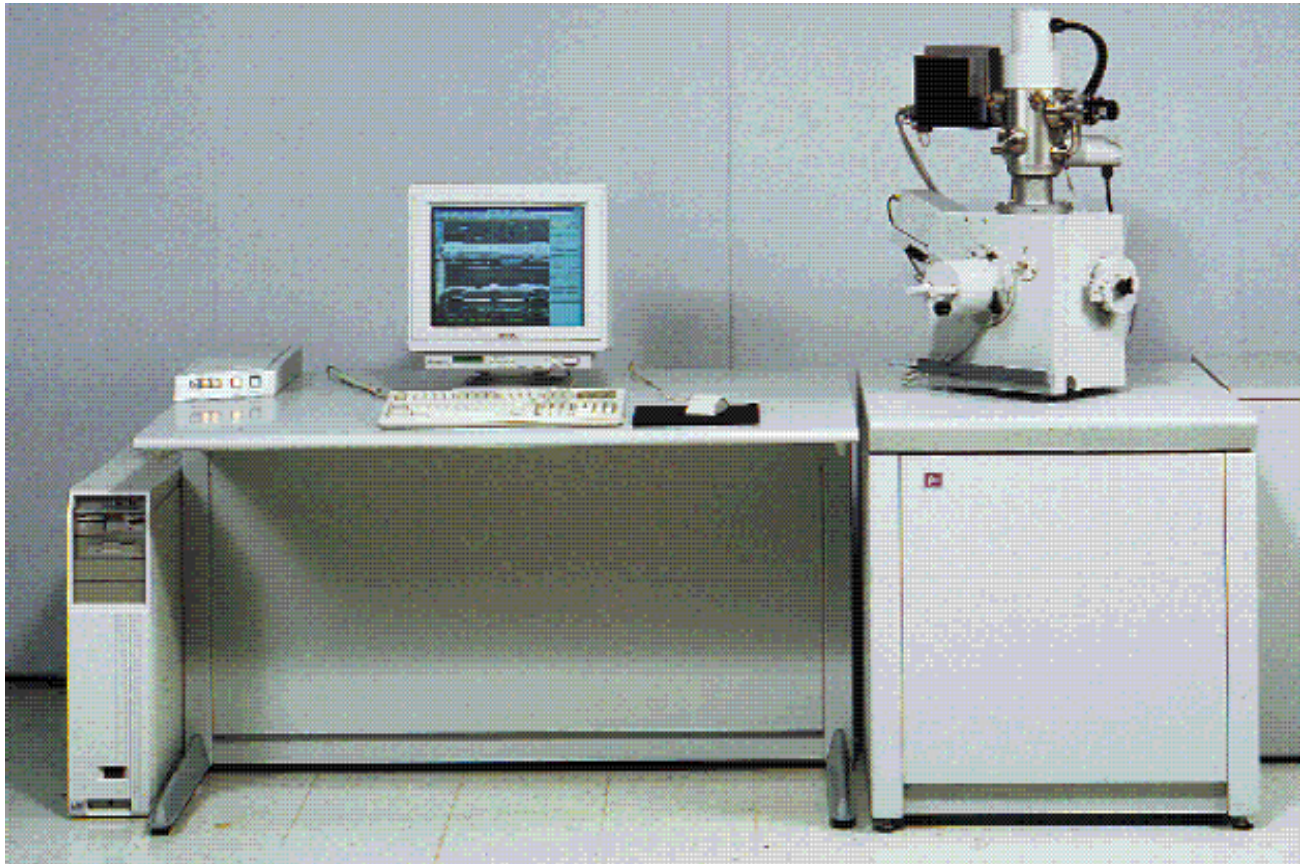
Fault injection / microprobing

- Can be used to actually read out information from e.g. data busses
  - A small needled is put onto e.g. the data bus, hence all information travelling over that bus can be extracted using that needle

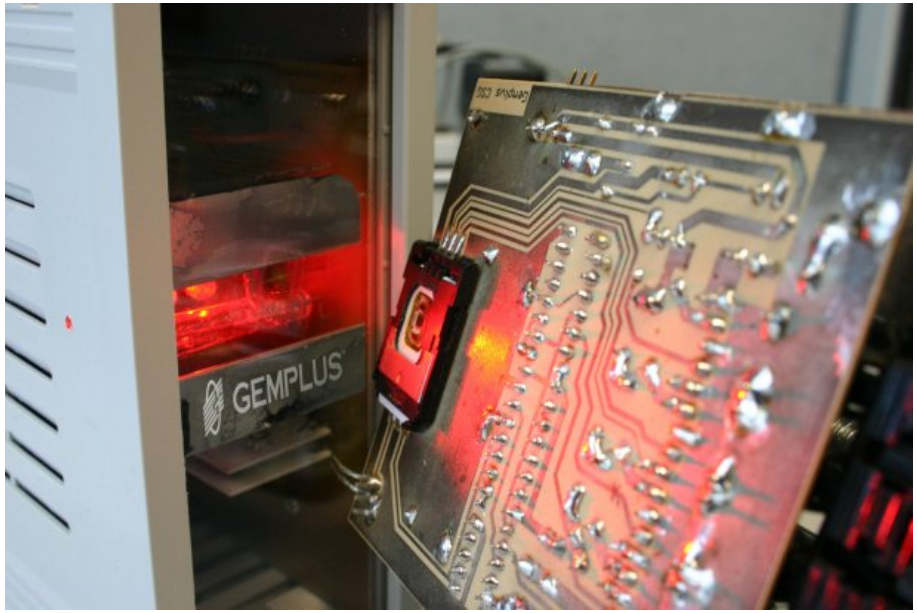
- Similarly one can aim to read out memory/memory banks (not necessarily using a needle for probing but by `looking´ at them)

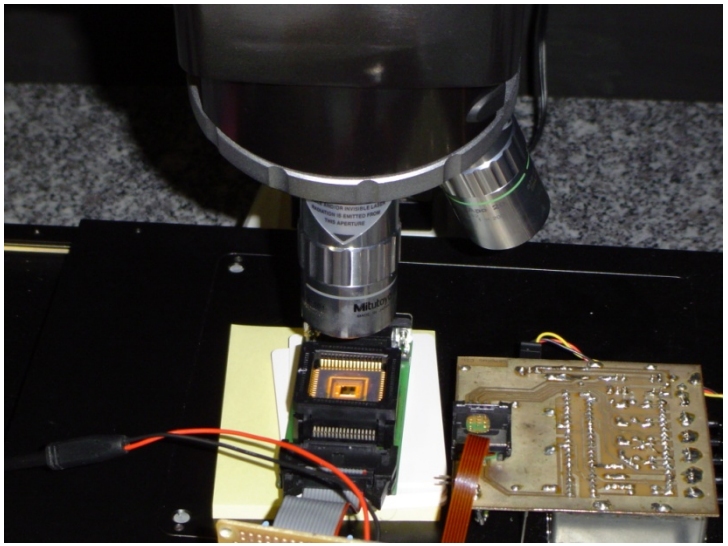
Fault injection: FIB

- If you have a quite a bit of money



Fault injection: light/Laser





- Hope is to change a (or several) bits (e.g. in memory) and not destroy the chip

What kind of faults?

- The effects of fault injection fall into three categories.
  - **Reseting data** — changing all the bits in a variable to zero or one (depends on the logic style).
  - **Ramdomising data** — changing a variable to some random value.
  - **Modifying opcodes** — changing the opcodes being processed by a microprocessor.
- The first two are typically expected to affect one computer word, whereas the third affects the program being executed.

A simple example

- The verification of a PIN number is a common command.

- Can you think of any way of injecting a fault such that we could get someone's PIN?

1. If PIN is greater than zero.
2. Is presented PIN equal to stored PIN?
3. If PIN is equal stored PIN return OK.
4. Otherwise return NOT_OK and decrement PIN counter.

```
if (PIN_counter > 0)
{
    if (INPUT == PIN)
    {
        return(OK);
    }
    else
    {
        PIN_counter--;
        return(NOT_OK);
    }
}
```

A simple countermeasure

- Decrease the PIN_counter before the conditional branch such that this cannot be circumvented by bypassing the branch …

```
if (PIN_counter > 0)
{
   PIN_counter--;
   if (INPUT == PIN)
   {
       PIN_counter++;
       return(OK);
   }
   else
   {
      return(NOT_OK);
   }
}
```

Fault attack on CRT implementations

- Assume we use RSA for signatures
  - Sign: $s=m^d \pmod N$, Verify: $v=s^e \pmod N$
  - We assume that we can induce faults during the signature generation
  - Aim is to recover d

Suppose we want to compute

$$y = x^d \pmod N$$

where $N = p \cdot q$.

We know by Lagranges Theorem

$$x^{p-1} = 1 \pmod p$$

So we first compute $y \pmod p$ and $y \pmod q$ via

$$y_p = y \pmod p = x^d \pmod p = x^{d \pmod{p-1}} \pmod p,$$
$$y_q = y \pmod q = x^d \pmod q = x^{d \pmod{q-1}} \pmod q.$$

We then solve for y by applying the CRT to the equations

$$y \equiv y_p \pmod p$$
$$y \equiv y_q \pmod q.$$

Fault attack on CRT, cont.

- We get one correct signature:

$$s_p = h(m)^{d mod\,(p-1)} mod\ p$$
$$s_q = h(m)^{d mod\,(q-1)} mod\ q$$
$$s = CRT(s_p, s_q)\ mod\ N$$
$$CRT(s_p, s_q)$$
$$= s_q + q \cdot (q^{-1}\ mod\ p) \cdot (s_p - s_q) mod\ N$$

Fault attack on RSA, cont.

- Then we induce a fault during CRT such that only one ‚half' gets affected:

$$s = s_q + q \cdot (q^{-1}\ mod\ p) \cdot (s_p - s_q) mod\ N$$
$$s' = s_q + q \cdot (q^{-1}\ mod\ p) \cdot (s'_p - s_q) mod\ N$$
$$s - s' = \mathrm{q} \cdot (q^{-1}\ mod\ p) \cdot (s_p - s'_p)\ mod N$$

- Which implies that gcd(s-s', N)=q
- Hence we can factor N easily.

Countermeasures?

- Two generic countermeasures
  - Perform computations twice and check results against each other
  - Verify result of computation by applying it‘s ‚inverse‘, e.g. verify signature using the verification algorithm and public key
- Generic countermeasures are very costly to implement!
- Specific ones do exist but they all have been broken … see next lecture

Countermeasures: Shamir's trick

$S_{pt} := m^d \bmod p \cdot t$ $\quad [= m^{(d \bmod (p-1)(t-1))} \bmod p \cdot t]$

$S_{qt} := m^d \bmod q \cdot t$ $\quad [= m^{(d \bmod (q-1)(t-1))} \bmod q \cdot t]$

$(S_{pt} \bmod t == S_{qt} \bmod t)$? If so then No error

$S_p := S_{pt} \bmod p$

$S_q := S_{qt} \bmod q$

$S := S_q + q \cdot [(S_p - S_q) \cdot q^{-1} \bmod p]$

- The value t is some small prime number, which is chosen at random

Adi Shamir. *Method and Apparatus for protecting public key schemes from timing and fault attacks*. US Patent Office, November 1999. US Patent Number 5 991 415, also presented at EUROCRYPT 1997.

Analysis of Shamir's trick

- What if error still produces a collision of S_{pt} and S_{qt} mod t?
  - Clearly error correction works with probability 1-1/t
- Pro: t can be small (i.e. around 32 bits) and we still get good error detection
- Con: we need to do computations with d rather than d_p and d_q
  - This may be problematic in practice if d simply isn't available
- Major problem: it can be faulted in many ways

Summary

- We looked at different ways to induce faults
- We named different types of faults
- We discussed the two main fault attacks for RSA
  - And had a first glimpse at how difficult it is to come up with a practical countermeasure