Applied Security

Countermeasures for RSA (5)

Countermeasures w.r.t. fault attacks, analysis of Shamir‘s trick, cont.

By Aumüller et al. (published in CHES 2002):

- Error in computation of p‘ =p_e*t or q‘=q*t and p_e is used later
  - Not detected, allows to factor N
- If either S_p or S_q are modified just before the recombination
  - Not detected, allows to factor N
- If q^{-1} mod p is modified
  - Not detected, allows to factor N

1. $S_{pt} := m^{dmod\ (p-1)(t-1)} \bmod p \cdot t$
2. $S_{qt} := m^{dmod\ (q-1)(t-1)} \bmod q \cdot t$
3. $S_p := S_{pt} \bmod p$
4. $S_q := S_{qt} \bmod q$
5. $S := S_q + q \cdot [(S_p - S_q) \cdot q^{-1} \bmod p]$
6. If $((S_{pt} \bmod t) <> (S_{qt} \bmod t)$ then Error otherwise return S

Analysis to the left shows that Shamir‘s trick really only protects the two modular exponentiations by randomising them. This however does not protect the rest of the CRT and hence is not good enough for practice.

Countermeasures w.r.t. fault attacks on RSA, cont.

- Aumüller et al. then show in their paper how to patch Shamir‘s trick, essentially by also randomising the computation of p*t and q*t and checking the result
- However it turns out that if m=0 mod t then certain faults can still not be detected.
- A number of further works over the last few years have developed the idea further but almost all have been broken (see table next page).

Overview of countermeasures against fault attacks on RSA

Author	Where	When	Broken?
Shamir	Eurocrypt	1997	Aumüller, CHES 2002
Yen et al	ICISC	2001	Yen, FDTC 2004
Aumüller et al	CHES	2002	Yen, ICISC 2002
Blömer et al	CCS	2003	Wagner, CCS 2004
Kim et al	HPCC	2005	-
Joye, Ciet	FDTC	2005	Berzati, FDTC 2008
Giraud	FDTC	2005	-
Fumaroli	FDTC	2006	Kim, FDTC 2007
Vigilant	FDTC	2008	

So how does a ‚working' countermeasure look like?

- Giraud, FDTC 2005: key observation is that when using a so-called Montgomery ladder exponentiation routine, $A_0 \cdot m = A_1$

  $(A_0, A_1) := (1, m);$
  for i:=n-1 to 0 by -1 do
  $(A_0, A_1) := (A_0 \cdot A_{di}, A_{di} \cdot A_1);$
  // now $(A_0 \cdot m = A_1)$
  end;
  return A0;

- Any error will destroy the natural relationship
- Giraud then uses this observation paired with the blinding trick by Shamir to compute 2 pairs of values $(S_p, S_p' = S_p \cdot m^{-1})$ and $(S_q, S_q' = S_q \cdot m^{-1})$. Before returning a signature S it is checked that $S = S' \cdot m$.
- Note that because the Montgomery ladder always performs the same operations in each step this algorithm is also resistant against SPA attacks.

Just a quick discussion

- Fault attacks are extremely effective as often a single faulty computation is sufficient to reveal the key
- However, they require a faulty and a correct computation on the same data using the same secret key
  - Probabilistic cryptographic schemes ensure that this never happens
    - RSA-PSS is a probabilistic signature scheme in which the input to the signature generation is randomised
    - RSA-OAEP is a probabilistic encryption scheme in which the input to the encryption is randomised. But the decryption routine will first decrypt the ciphertext and then remove the padding and so this initial RSA decryption is not randomised and vulnerable w.r.t. fault attacks

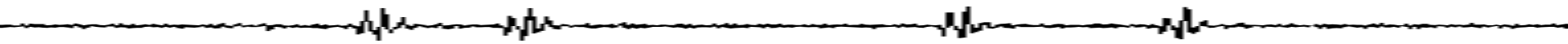
What about countermeasures against timing and power analysis?

- Remember both differential power and timing analysis require several observations because they exploit very small dependencies
  - Make such dependencies smaller with regards to the rest of the computation (signal vs. noise)
    - Do something at random (in such a way that it cannot be easily detected)
      - Add some dummy computations
      - Change sequence of instructions (if possible)
- Randomise intermediate values such that they are not predictable by an attacker (next slide)
  - Such measures can both help and hinder fault attacks

Countermeasures against DPA attacks are typically based on masking (aka blinding)

- Blinding refers to concealing intermediate values with a random value
  - Masks are applied at the beginning of algorithm to a value and can be (easily) removed at the end
    - Blinding of message
    - Blinding of exponent

- Inherent properties of representation of values allow further randomizations
  - If certain residue number systems are used some of their parameters can be randomised (but this is hardly done for RSA)



Case study: blinding applied to RSA

Exponent blinding :

m … random value

$d_m = d + m \times \phi(n)$

Works because

$v^{d_m} = v^{d+m\times\phi(n)} \equiv v^d \bmod n$

Message blinding :

m … random value

$v_m = v \times m^e$

Works because

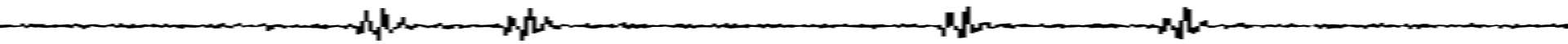
$(v_m)^d \equiv v^d \times m \bmod n$

Both types of blinding require considerable additional effort:

- Exponent blinding: more steps in exponentiation
- Message blinding: removing mask at the end

Blinding cont.

- Let‘s stick wit DPA and DTA attacks for the moment
  - They rely on an attacker‘s ability to use several traces with the same secret key
  - Hence blinding is effective against such attacks
- Can the initial blinding operation be attacked?
  - If not protected against SPA then yes!
    - However it might be easier to protect a multiplication or addition against SPA than an exponentiation!



Blinding does not automatically prevent SPA attacks

- Suppose an implementation uses exponent blinding
  - Exponentiation uses d_m instead of d
- Assume attacker can reveal d_m using an SPA attack
  - Remember that $d_m \equiv d \bmod \phi(n)$
  - Attacker can use d_m to decrypt messages!
- An implementation using message blinding only does not protect against SPA anyway
- Conclusion: one needs to ensure that SPA attacks are not possible AND apply blinding to thwart DPA attacks

Does CRT prevent DPA and DTA attacks?

- Not if an attacker can target/exploit intermediate values in the initial reduction of the input c mod p and c mod q
- Another point for attack might be the final combination $S := S_q + q \cdot [(S_p - S_q) \cdot q^{-1} \bmod p]$
  - If p and q are of equal size then S_q is about half the size of S and so $S \approx q \cdot [(S_p - S_q) \cdot q^{-1} \bmod p]$
  - Hence we can guess the top bits of q and compute S/q some bits of $q \cdot [(S_p - S_q) \cdot q^{-1} \bmod p]$
    - Make model based on those bits and decide about key guess w.r.t. q

Summary

- When implementing PKC one clearly has a tough job
  - We must NOT have any instruction flow that is data dependent
  - We must conceal somehow any other observable behaviour that is dependent on the secret data
  - We must check for faults and ensure that faulty outputs are not given to the attacker
  - And that all without too much increasing size, memory requirements, running time, required randomness etc.
- To succeed one must have in-depth knowledge of many implementation options/tricks, and their properties