Fault Attacks and Countermeasures on Vigilant's RSA-CRT Algorithm

J.-S. Coron[1], C.Giraud[2], N. Morin[2], G.Piret[2] and **D. Vigilant**[3]

[1] Univerisité du Luxembourg
jean-sebastien.coron@uni.lu

[2] Oberthur Technologies
[c.giraud, n.morin, g.piret]@oberthur.com

[3] **Gemalto**
david.vigilant@gemalto.com

FDTC - August 21, 2010

Outline

1. Background and Context
   - CRT-RSA system
   - Vigilant's Secure Ring Exponentiation (CHES '08)
   - Application to RSA-CRT
2. Fault Attacks and Countermeasures
   - Fault Model
   - Exponent randomization Disturbance
   - Modulus Computation Disturbance
3. Conclusion

Outline

1. Background and Context
   - CRT-RSA system
   - Vigilant's Secure Ring Exponentiation (CHES '08)
   - Application to RSA-CRT
2. Fault Attacks and Countermeasures
   - Fault Model
   - Exponent randomization Disturbance
   - Modulus Computation Disturbance
3. Conclusion

RSA-CRT system

RSA-CRT parameters:
(N, e) Public key
(p, q, d_p, d_q, i_q) Private key

$$\text{such that } \begin{cases} N = p \times q, (p, q \text{ large primes}) \\ \gcd((p-1), e) = 1 \\ \gcd((q-1), e) = 1 \\ d_p = e^{-1} \bmod (p-1) \\ d_q = e^{-1} \bmod (q-1) \\ i_q = q^{-1} \bmod p \end{cases}$$

RSA-CRT system

RSA-CRT process

Input: $m \in \mathbb{Z}_N, p, q, d_p, d_q, i_q$
Output: $m^d \in \mathbb{Z}_N$

$S_p = m^{d_p} \bmod p$
$S_q = m^{d_q} \bmod q$
$S = S_q + q \times (i_q \times (S_p - S_q) \bmod p)$
return S

$\Rightarrow$ RSA-CRT is preferred ($4\times$ faster , handles data with size $\frac{1}{2}|N|$)
$\Rightarrow$ Better suited to embedded device constraints
Public exponent e often unavailable

Bellcore attack '97

RSA-CRT process

Input: $m \in \mathbb{Z}_N, p, q, d_p, d_q, i_q$
Output: $m^d \in \mathbb{Z}_N$

$\underline{S_p} = m^{d_p} \bmod p \Leftarrow$
$S_q = m^{d_q} \bmod q$
$\underline{S} = S_q + q \times (i_q \times (\underline{S_p} - S_q) \bmod p)$
return S

$\Rightarrow \gcd(\underline{S} - S \bmod N, N) = q$

Bellcore attack '97

RSA-CRT process

Input: $m \in \mathbb{Z}_N, p, q, d_p, d_q, i_q$
Output: $m^d \in \mathbb{Z}_N$

$$S_p = m^{d_p} \bmod p$$
$$\underline{S_q} = m^{d_q} \bmod q \Leftarrow$$
$$\underline{S} = \underline{S_q} + q \times (i_q \times (S_p - \underline{S_q}) \bmod p)$$
return S

$$\Rightarrow \gcd(\underline{S} - S \bmod N, N) = p$$

Bellcore attack '97

RSA-CRT process

Input: $m \in \mathbb{Z}_N, p, q, d_p, d_q, i_q$
Output: $m^d \in \mathbb{Z}_N$

$S_p = m^{d_p} \mod p$
$S_q = m^{d_q} \mod q$
$\underline{S} = S_q + q \times \underline{(i_q \times (S_p - S_q) \mod p)} \Leftarrow$
return S

$\Rightarrow \gcd(\underline{S} - S \mod N, N) = q$

Vigilant's Secure Ring Exponentiation (CHES '08)

Context: exponentiation $S = m^d \bmod N$

Variant of Shamir's countermeasure ('97):

- Introduction of a random R
- Exponentiation made modulo NR instead of modulo N
- Verification of the exponentiation result consistency modulo R
- Exponentiation result reduced modulo N

$\Rightarrow$ Allows the fault detection

Vigilant's Secure Ring Exponentiation (CHES '08)

Context: exponentiation $S = m^d \bmod N$
Let N an integer and R a random (e.g. 64 bits) s.t. $\gcd(N, R) = 1$

We introduce

$$\alpha \equiv \begin{cases} 1 \bmod N \\ 0 \bmod R \end{cases} \quad \text{and} \quad \beta \equiv \begin{cases} 0 \bmod N \\ 1 \bmod R \end{cases}$$

$\beta = N \times (N^{-1} \bmod R) \bmod N.R$
$\alpha = 1 - \beta \bmod N.R$

Vigilant's Secure Ring Exponentiation (CHES '08)

Considering now $R = r^2$ where r is a random integer (e.g. 32 bits):
$\beta = N \times (N^{-1} \bmod r^2)$ and $\alpha = 1 - \beta \bmod Nr^2$

$$\hat{m} = \alpha m + \beta \cdot (1 + r) \bmod Nr^2$$

$$\hat{m} \equiv \begin{cases} m \bmod N \\ 1 + r \bmod r^2 \end{cases}$$

$$S_r = \hat{m}^d \bmod Nr^2 = \alpha m^{d} + \beta \cdot (1 + dr) \bmod Nr^2$$

$$S_r \equiv \begin{cases} m^d \bmod N \\ 1 + dr \bmod r^2 \end{cases}$$

Vigilant's Secure Ring Exponentiation (CHES '08)

We want to compute $S = m^d \bmod N$
How to check if no disturbance?

example: flipping exponent bit attack (Boneh et al. '01)

1. Pick a random r coprime with N and compute α and β
2. Compute $\hat{m} = \alpha m + \beta \cdot (1 + r) \bmod Nr^2$
3. Check that: $m = \hat{m} \bmod N$
4. Compute $S_r = \hat{m}^d \bmod Nr^2$
5. Reduce modulo N: $S = S_r \bmod N$
6. Check that: $S_r = \alpha S + \beta \cdot (1 + dr) \bmod Nr^2$

Vigilant's Secure Ring Exponentiation (CHES '08)

We want to compute $S = m^d \bmod N$
How to check if no disturbance?

example: flipping exponent bit attack (Boneh et al. '01)

1. Pick a random r coprime with N and compute α and β
2. Compute $\hat{m} = \alpha m + \beta \cdot (1 + r) \bmod Nr^2$
3. Check that: $m = \hat{m} \bmod N$
4. Compute $S_r = \hat{m}^d \bmod Nr^2$
5. Reduce modulo N: $S = S_r \bmod N$
6. Check that: $S_r = \alpha S + \beta \cdot (1 + dr) \bmod Nr^2$

Vigilant's Secure Ring Exponentiation (CHES '08)

We want to compute $S = m^d \mod N$
How to check if no disturbance?

example: flipping exponent bit attack (Boneh et al. '01)

1. Pick a random r coprime with N and compute α and β
2. Compute $\hat{m} = \alpha m + \beta \cdot (1 + r) \mod Nr^2$
3. Check that: $m = \hat{m} \mod N$
4. Compute $S_r = \hat{m}^{d \ xor \ 2^i} \mod Nr^2 \Leftarrow$ transient fault
5. Reduce modulo N: $S = S_r \mod N$
6. Check that: $S_r = \alpha S + \beta \cdot (1 + dr) \mod Nr^2$

detected : $S_r = \alpha S + \beta \cdot (1 + ((d \ xor \ 2^i) \cdot r)) \mod Nr^2$

Vigilant's Secure Ring Exponentiation (CHES '08)

We want to compute $S = m^d \bmod N$
How to check if no disturbance?

example: flipping exponent bit attack (Boneh et al. '01)

1. Pick a random r coprime with N and compute α and β
2. Compute $\hat{m} = \alpha m + \beta \cdot (1 + r) \bmod Nr^2$
3. Check that: $m = \hat{m} \bmod N$
4. Compute $S_r = \hat{m}^{d \ xor \ 2^i} \bmod Nr^2 \Leftarrow$ transient fault
5. Reduce modulo N: $S = S_r \bmod N$
6. Check that: $S_r = \alpha S + \beta \cdot (1 + dr) \bmod Nr^2$
   detected : $S_r = \alpha S + \beta \cdot (1 + ((d \ xor \ 2^i) \cdot r)) \bmod Nr^2$

Application to RSA-CRT ('08): Half exponentiation

r is a 32-bit random integer and R_1 is a 64-bit random integer
(Critical verifications in red)

	$\bmod p$	$\bmod r^2$	$\bmod p-1$
1. $m_p = m \bmod pr^2$	m	m	
2. $\beta_p = p \cdot (p^{-1} \bmod r^2)$	0	1	
3. $\alpha_p = 1 - \beta_p \bmod pr^2$	1	0	
4. $\hat{m}_p = \alpha_p m_p + \beta_p \cdot (1+r)$	m	$1+r$	
5. $d'_p = d_p + R_1 \cdot (p-1)$			d_p
6. $S_{pr} = \hat{m}_p^{d'_p} \bmod pr^2$	m^{d_p}	$1+d'_p r$	

Application to RSA-CRT: Half exponentiation

r is a 32-bit random integer and R_2 is a 64-bit random integer
(Critical verifications in red)

	mod q	mod r^2	mod $q-1$
1. $m_q = m \bmod qr^2$	m	m	
2. $\beta_q = q \cdot (q^{-1} \bmod r^2)$	0	1	
3. $\alpha_q = 1 - \beta_q \bmod qr^2$	1	0	
4. $\hat{m}_q = \alpha_q m_q + \beta_q \cdot (1+r)$	m	$1+r$	
5. $d'_q = d_q + R_2 \cdot (q-1)$			d_q
6. $S_{qr} = \hat{m}_q^{d'_q} \bmod qr^2$	m^{d_q}	$1 + d'_q r$	

Application to RSA-CRT: Recombination

R_3 and R_4 are 64-bit random integers
Recombination:

1. Transform

$$S_{pr} \text{ into } S'_p \text{ s.t. } \begin{cases} S'_p \equiv m^{d_p} \bmod p \\ S'_p \equiv R_3 \bmod r^2 \end{cases}$$

$$\text{and } S_{qr} \text{ into } S'_q \text{ s.t. } \begin{cases} S'_q \equiv m^{d_q} \bmod q \\ S'_q \equiv R_4 \bmod r^2 \end{cases}$$

2. $S = S'_q + q \cdot (i_q \cdot (S'_p - S'_q) \bmod pr^2)$
3. Check $S \bmod r^2 \ ? = R_4 + qi_q \cdot (R_3 - R_4) \bmod r^2$
4. Return $S \bmod N$ if all checks positive

Outline

1. Background and Context
   - CRT-RSA system
   - Vigilant's Secure Ring Exponentiation (CHES '08)
   - Application to RSA-CRT
2. Fault Attacks and Countermeasures
   - Fault Model
   - Exponent randomization Disturbance
   - Modulus Computation Disturbance
3. Conclusion

Random Fault Model

As in the original paper, it is considered that an attacker can:

- modify a value in memory with a random value (permanent fault)
- modify a value during the computation with a random value (transient fault)
- not modify the code execution or Boolean results of comparisons
- not inject permanent faults in p, q, d_p, d_q, i_q.
  (associated to an integrity value)

Exponent randomization Disturbance

Exponent randomization Disturbance: Attack

Reading RSA-CRT pseudo-code in the original paper:

- $d'_p = d_p + R_1 \cdot (p-1)$
- Check that $d'_p \ ?= d_p \bmod (p-1)$

A natural way of implementing these steps is to perform the following:

- $pminusone = p - 1$
- $d'_p = d_p + R_1 \cdot pminusone$
- Check that $d'_p \ ?= d_p \bmod pminusone$

- The value of $pminusone$ is not used anymore

Exponent randomization Disturbance: Attack

Reading RSA-CRT pseudo-code in the original paper:

- $d'_p = d_p + R_1 \cdot (p - 1)$
- Check that $d'_p \; ? = d_p \bmod (p - 1)$

A natural way of implementing these steps is to perform the following:

- $pminusone = p - 1$
- $d'_p = d_p + R_1 \cdot pminusone$
- Check that $d'_p \; ? = d_p \bmod pminusone$

- The value of $pminusone$ is not used anymore

Exponent randomization Disturbance: Attack

Reading RSA-CRT pseudo-code in the original paper:

- $d'_p = d_p + R_1 \cdot (p - 1)$
- Check that $d'_p \; ?= d_p \bmod (p - 1)$

A natural way of implementing these steps is to perform the following:

- $\underline{pminusone = p - 1}$ $\Leftarrow$ **sensitive to transient or permanent fault**
- $d'_p = d_p + R_1 \cdot pminusone$
- Check that $d'_p \; ?= d_p \bmod pminusone$

- The value of $pminusone$ is not used anymore

Exponent randomization Disturbance: Attack

Reading RSA-CRT pseudo-code in the original paper:

- $d'_p = d_p + R_1 \cdot (p - 1)$
- Check that $d'_p \; ? = d_p \bmod (p - 1)$

A natural way of implementing these steps is to perform the following:

- $\underline{pminusone = p - 1}$ $\Leftarrow$ **sensitive to transient or permanent fault**
- $d'_p = d_p + R_1 \cdot pminusone$
- $\underline{\text{Check that } d'_p \; ? = d_p \bmod pminusone}$
  Test true even if $pminusone$ **faulty**
- The value of $pminusone$ is not used anymore

Exponent randomization Disturbance: Attack

The attacker injects a transient fault in $pminusone$ computation, or a permanent fault in $pminusone$ juste before d'_p computation

Thus the attacker obtains a faulty $\underline{S}$ which is faulty only modulo p

The attacker can perform a $\gcd$ attack to recover $p = \gcd(\underline{S}^e - m \bmod N, N)$

Exponent randomization Disturbance: Countermeasures

A secure implementation must:

- Either use $pminusone$ in the sequel of the signature calculation: Indeed, recompute p from $pminusone$: Add a step $p = pminusone+1$
- Or compute $pminusone$ twice and verify that both results are equal

The same holds for $qminusone$

Modulus Computation Disturbance

Modulus Computation Disturbance: Attack

In the original paper, final steps are exactly written as follows:

- $N = pq$
- Check $N.[S - R_4 - qi_q.(R_3 - R_4)] \bmod Nr^2 \; ? = 0$

  and $q.i_q \bmod p \; ? = 1$
- Return $S \bmod N$ if all checks positive

Modulus Computation Disturbance: Attack

In the original paper, final steps are exactly written as follows:

- $\underline{N = pq}$ $\Leftarrow$**sensitive to transient fault**
- Check $N.[S - R_4 - qi_q.(R_3 - R_4)] \bmod Nr^2 \; ?= 0$

  and $q.i_q \bmod p \; ?= 1$
- Return $S \bmod N$ if all checks positive

Modulus Computation Disturbance: Attack

In the original paper, final steps are exactly written as follows:

- $\underline{N = pq}$ $\Leftarrow$**sensitive to transient fault**
- $\underline{\text{Check } N.[S - R_4 - qi_q.(R_3 - R_4)] \bmod Nr^2 \; ? = 0}$
  Test true whatever is N
  and $q.i_q \bmod p \; ? = 1$
- Return $S \bmod N$ if all checks positive

Modulus Computation Disturbance: Attack

The attacker injects a transient fault in p during the computation of N, $\underline{N} = \underline{p} \times q$

$S \bmod \underline{N}$ is returned

The attacker has a signature faulty modulo p, and correct modulo q

Again, he can compute $p = \gcd(\underline{S}^e - m \bmod N, N)$

Modulus Computation Disturbance: Countermeasure

Clear need to verify the integrity of the modulus.
It can be done through different simple ways, for instance:

- Replace "Check $N.[S - R_4 - qi_q.(R_3 - R_4)] \bmod Nr^2 \; ? = 0$ " by "Check $p.q.[S - R_4 - qi_q.(R_3 - R_4)] \bmod Nr^2 \; ? = 0$ " before returning $S \bmod N$
- Add a final step "Check $N.i_{qr} \bmod r^2 \; ? = p \bmod r^2$ " before returning $S \bmod N$
- Select a random T, compute $T_p = p \bmod T$, $T_q = q \bmod T$ and add a final step "Check that $N \bmod T \; ? = T_p.T_q \bmod T$" before returning $S \bmod N$
- ...

Outline

1. Background and Context
   - CRT-RSA system
   - Vigilant's Secure Ring Exponentiation (CHES '08)
   - Application to RSA-CRT
2. Fault Attacks and Countermeasures
   - Fault Model
   - Exponent randomization Disturbance
   - Modulus Computation Disturbance
3. Conclusion

Conclusion

We have shown 2 attacks:

- **Modulus computation disturbance**: A transient fault in p or q during the modulus computation before final reduction ...
- **Exponent randomization disturbance**: A transient fault during $p-1$ or $q-1$ computation, or a permanent fault in $p-1$ and $q-1$ values before the computation of d'_p or d'_q ...

They allow performing $\gcd$ attacks and recovering the secret key on Vigilant's RSA-CRT algorithm

We have given simple countermeasures thwarting both attacks

- Verification of modulus integrity, before returning the result
- Verification or reusing of $p-1$ and $q-1$ values

Conclusion

Since countermeasures have a negligible cost,

The combination of the original scheme with presented countermeasures

- Remains well-suited to constraints of embedded device
- Gives very high level of fault detection capability when public exponent is unknown

These attacks may impact most of others RSA-CRT schemes
(e.g.) Exponent randomization disturbance feasible on Aumüller et al.'s scheme (CHES'02)
$\Rightarrow$ Impact of attacks on all other schemes to be evaluated

Thanks for your attention

Any Questions?