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

J.-S. Coron\textsuperscript{1}, C.Giraud\textsuperscript{2}, N. Morin\textsuperscript{2}, G.Piret\textsuperscript{2} and D. Vigilant\textsuperscript{3}

\textsuperscript{1} Univerisité du Luxembourg  
jean-sebastien.coron@uni.lu

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

\textsuperscript{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

such that

\[
\begin{align*}
N &= p \times q, (p, q \text{ large primes}) \\
gcd((p - 1), e) &= 1 \\
gcd((q - 1), e) &= 1 \\
d_p &= e^{-1} \text{ mod } (p - 1) \\
d_q &= e^{-1} \text{ mod } (q - 1) \\
i_q &= q^{-1} \text{ mod } p
\end{align*}
\]
Fault Attacks and Countermeasures on Vigilant’s RSA-CRT Algorithm

Background and Context

CRT-RSA system

RSA-CRT process

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

\[ S_p = m^{d_p} \mod p \]
\[ S_q = m^{d_q} \mod q \]
\[ S = S_q + q \times (i_q \times (S_p - S_q) \mod p) \]

\[ \text{return } S \]

\[ \Rightarrow \text{RSA-CRT is preferred (}4\times \text{ faster, handles data with size } \frac{1}{2} |N|) \]
\[ \Rightarrow \text{Better suited to embedded device constraints} \]

Public exponent \( e \) often unavailable
RSA-CRT process

**Input:** \( m \in \mathbb{Z}_N, p, q, d_p, d_q, i_q \)

**Output:** \( m^d \in \mathbb{Z}_N \)

\[
S_p = m^{d_p} \mod p \\
S_q = m^{d_q} \mod q \\
S = S_q + q \times (i_q \times (S_p - S_q) \mod p) \\
\text{return } S
\]

\[ \Rightarrow \gcd(S - S \mod N, N) = q \]
Bellcore attack ’97

RSA-CRT process

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

**Output:** $m^d \in \mathbb{Z}_N$

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

$\Rightarrow \gcd(S - S \mod N, N) = p$
Bellcore attack ’97

RSA-CRT process

**Input:** \( m \in \mathbb{Z}_N, p, q, d_p, d_q, i_q \)

**Output:** \( m^d \in \mathbb{Z}_N \)

\[
S_p = m^{d_p} \mod p \\
S_q = m^{d_q} \mod q \\
S = S_q + q \times (i_q \times (S_p - S_q) \mod p) \\
\text{return } S
\]

\[ \Rightarrow \gcd(S - S \mod N, N) = q \]
Context: exponentiation $S = m^d \mod N$

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

Let $N$ an integer and $R$ a random (e.g. 64 bits) s.t. $\gcd(N, R) = 1$

We introduce

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

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

$$\alpha = 1 - \beta \mod N.R$$
Considering now $R = r^2$ where $r$ is a random integer (e.g. 32 bits):

\begin{align*}
\beta &= N \times (N^{-1} \mod r^2) \quad \text{and} \quad \alpha = 1 - \beta \mod Nr^2 \\
\hat{m} &= \alpha m + \beta \cdot (1 + r) \mod Nr^2
\end{align*}

\begin{align*}
\hat{m} &= \begin{cases} 
m \mod N \\
1 + r \mod r^2
\end{cases} \\
S_r &= \hat{m}^d \mod Nr^2 = \alpha m^d + \beta \cdot (1 + dr) \mod Nr^2
\end{align*}

\begin{align*}
S_r &= \begin{cases} 
m^d \mod N \\
1 + dr \mod r^2
\end{cases}
\end{align*}
Fault Attacks and Countermeasures on Vigilant's RSA-CRT Algorithm

Background and Context

Vigilant's Secure Ring Exponentiation (CHES '08)

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 $\alpha$ and $\beta$
2. Compute $\hat{m} = \alpha m + \beta \cdot (1 + r) \mod Nr^2$
3. Check that: $m = \hat{m} \mod N$
4. Compute $S_r = \hat{m}^d \mod Nr^2$
5. Reduce modulo $N$: $S = S_r \mod N$
6. Check that: $S_r = \alpha S + \beta \cdot (1 + dr) \mod Nr^2$
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 $\alpha$ and $\beta$
2. Compute $\hat{m} = \alpha m + \beta \cdot (1 + r) \mod Nr^2$
3. Check that: $m = \hat{m} \mod N$
4. Compute $S_r = \hat{m}^d \mod Nr^2$
5. Reduce modulo $N$: $S = S_r \mod N$
6. Check that: $S_r = \alpha S + \beta \cdot (1 + dr) \mod Nr^2$
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 \( \alpha \) and \( \beta \)
2. Compute \( \hat{m} = \alpha m + \beta \cdot (1 + r) \mod Nr^2 \)
3. Check that: \( m = \hat{m} \mod N \)
4. Compute \( S_r = \hat{m}^d \mod Nr^2 \iff \text{transient fault} \)
5. Reduce modulo \( N \): \( S = S_r \mod N \)
6. Check that: \( S_r = \alpha S + \beta \cdot (1 + dr) \mod Nr^2 \)
   detected: \( S_r = \alpha S + \beta \cdot (1 + ((d \cdot r) \cdot r)) \mod Nr^2 \)
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 \( \alpha \) and \( \beta \)
2. Compute \( \hat{m} = \alpha m + \beta \cdot (1 + r) \mod Nr^2 \)
3. Check that: \( m = \hat{m} \mod N \)
4. Compute \( S_r = \hat{m}^d \mod Nr^2 \leftarrow \text{transient fault} \)
5. Reduce modulo \( N \): \( S = S_r \mod N \)
6. Check that: \( S_r = \alpha S + \beta \cdot (1 + dr) \mod Nr^2 \)
   detected: \( S_r = \alpha S + \beta \cdot (1 + ((d \ xor \ 2^i) \cdot r)) \mod Nr^2 \)
Fault Attacks and Countermeasures on Vigilant's RSA-CRT Algorithm

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)

1. \( m_p = m \mod pr^2 \)

\[
\begin{align*}
\text{mod } p & \quad m \\
\text{mod } r^2 & \quad m \\
\text{mod } p - 1 & \quad m
\end{align*}
\]

2. \( \beta_p = p \cdot (p^{-1} \mod r^2) \)

\[
\begin{align*}
\text{mod } p & \quad 0 \\
\text{mod } r^2 & \quad 1 \\
\text{mod } p - 1 & \quad 1
\end{align*}
\]

3. \( \alpha_p = 1 - \beta_p \mod pr^2 \)

\[
\begin{align*}
\text{mod } p & \quad 1 \\
\text{mod } r^2 & \quad 0 \\
\text{mod } p - 1 & \quad 1 + r
\end{align*}
\]

4. \( \hat{m}_p = \alpha_p m_p + \beta_p \cdot (1 + r) \)

\[
\begin{align*}
\text{mod } p & \quad m \\
\text{mod } r^2 & \quad 1 + r \\
\text{mod } p - 1 & \quad d_p
\end{align*}
\]

5. \( d'_p = d_p + R_1 \cdot (p - 1) \)

\[
\begin{align*}
\text{mod } p & \quad m^{d'_p} \\
\text{mod } r^2 & \quad 1 + d'_p r \\
\text{mod } p - 1 & \quad d_p
\end{align*}
\]

6. \( S_{pr} = \hat{m}_p^{d'_p} \mod pr^2 \)
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)

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

$R_3$ and $R_4$ are 64-bit random integers

Recombination:

1. Transform

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

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

2. \[S = S'_q + q \cdot (i_q \cdot (S'_p - S'_q) \mod r^2)\]

3. Check $S \mod r^2 \ ? = R_4 + qi_q \cdot (R_3 - R_4) \mod r^2$

4. Return $S \mod N$ 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)*
Fault Attacks and Countermeasures on Vigilant's RSA-CRT Algorithm

Exponent randomization Disturbance

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 \( d'_p \equiv d_p \mod (p - 1) \)

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

- \( p\text{-minusone} = p - 1 \)
- \( d'_p = d_p + R_1 \cdot p\text{-minusone} \)
- Check that \( d'_p \equiv d_p \mod p\text{-minusone} \)

- The value of \( p\text{-minusone} \) 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 \( d'_p \equiv d_p \mod (p - 1) \)

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 \( d'_p \equiv d_p \mod pminusone \)

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 \(d'_p \equiv d_p \mod (p - 1)\)

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

- \(p_{\text{minusone}} = p - 1 \iff \text{sensitive to transient or permanent fault}\)
- \(d'_p = d_p + R_1 \cdot p_{\text{minusone}}\)
- Check that \(d'_p \equiv d_p \mod p_{\text{minusone}}\)

- The value of \(p_{\text{minusone}}\) 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 \( d'_p \? = d_p \mod (p - 1) \)

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

- \( p\text{minusone} = p - 1 \) \(\Leftarrow\) sensitive to transient or permanent fault
- \( d'_p = d_p + R_1 \cdot p\text{minusone} \)
- Check that \( d'_p \? = d_p \mod p\text{minusone} \)
  Test true even if \( p\text{minusone} \) faulty
- The value of \( p\text{minusone} \) is not used anymore
Fault Attacks and Countermeasures on Vigilant’s RSA-CRT Algorithm

Exponent randomization Disturbance

Exponent randomization Disturbance: Attack

The attacker injects a transient fault in $p_{\text{minus one}}$ computation, or a permanent fault in $p_{\text{minus one}}$ juste before $d_p'$ computation.

Thus the attacker obtains a faulty $S$ which is faulty only modulo $p$.

The attacker can perform a $\text{gcd}$ attack to recover

$$ p = \text{gcd}(S^e - m \mod N, N) $$
Exponent randomization Disturbance: Countermeasures

A secure implementation must:

- Either use \( p_{\text{minus one}} \) in the sequel of the signature calculation:
  Indeed, recompute \( p \) from \( p_{\text{minus one}} \): Add a step \( p = p_{\text{minus one}} + 1 \)

- Or compute \( p_{\text{minus one}} \) twice and verify that both results are equal

The same holds for \( q_{\text{minus one}} \)
Fault Attacks and Countermeasures on Vigilant's RSA-CRT Algorithm

Modulus Computation Disturbance
Modulus Computation Disturbance: Attack

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

- \( N = pq \)

- Check \( N \cdot [S - R_4 - qi_q \cdot (R_3 - R_4)] \mod Nr^2 \equiv 0 \)

  and \( qi_q \mod p \equiv 1 \)

- Return \( S \mod N \) if all checks positive
Modulus Computation Disturbance: Attack

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

- \( N = pq \) is sensitive to transient fault

- Check \( N \cdot [S - R_4 - q_i q \cdot (R_3 - R_4)] \mod Nr^2 \equiv 0 \)

  and \( q \cdot i_q \mod p \equiv 1 \)

- Return \( S \mod N \) if all checks positive
Modulus Computation Disturbance: Attack

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

1. $N = pq \iff \text{sensitive to transient fault}$

2. Check $N. [S - R_4 - qi_q.(R_3 - R_4)] \mod Nr^2 \neq 0$
   
   \underline{Test true whatever is $N$}
   
   and $q.i_q \mod p \neq 1$

3. Return $S \mod N$ if all checks positive
Modulus Computation Disturbance: Attack

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

$S \mod N$ is returned

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

Again, he can compute $p = \gcd(S^e - m \mod N, N)$
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 - q_i_q. (R_3 - R_4)] \mod Nr^2 \neq 0 " by "Check $p.q. [S - R_4 - q_i_q. (R_3 - R_4)] \mod Nr^2 \neq 0 " before returning $S \mod N$

- Add a final step "Check $N.q_r \mod r^2 \neq p \mod r^2 " before returning $S \mod N$

- Select a random $T$, compute $T_p = p \mod T$, $T_q = q \mod T$ and add a final step "Check that $N \mod T \neq T_p.T_q \mod T" before returning $S \mod N$

...
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
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
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)
⇒ Impact of attacks on all other schemes to be evaluated
Thanks for your attention

Any Questions?