Padding Oracle Attack for non-standard PKCS#1 v1.5
Can non-standard implementation provide us a shelter?

Si Gao  Hua Chen  Limin Fan

Trusted Computing and Information Assurance Laboratory, Institute of Software, Chinese Academy of Sciences

November 20, 2013
Outline

Introduction
   Basic Problem
   Motivation

Non-standard implementation study
   Case I
   Case II
   Categorize Non-standard implementation
   Further improvement*

Summary
Outline

Introduction
  Basic Problem
  Motivation

Non-standard implementation study
  Case I
  Case II
  Categorize Non-standard implementation
  Further improvement*

Summary
Outline

Introduction
  Basic Problem
  Motivation

Non-standard implementation study
  Case I
  Case II
  Categorize Non-standard implementation
  Further improvement*

Summary
PKCS#1 v1.5 coding for encryption

A valid “padded message” should satisfy:

<table>
<thead>
<tr>
<th>00</th>
<th>02</th>
<th>Padding String</th>
<th>00</th>
<th>Data Block</th>
</tr>
</thead>
</table>

\[ \text{a)} \quad =0x00? \]
PKCS#1 v1.5

PKCS#1 v1.5 coding for encryption
A valid “padded message” should satisfy:

<table>
<thead>
<tr>
<th>00</th>
<th>02</th>
<th>Padding String</th>
<th>00</th>
<th>Data Block</th>
</tr>
</thead>
</table>

b) $=0x02$?
PKCS#1 v1.5 coding for encryption
A valid “padded message” should satisfy:

- Search for the first 0x00

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>Padding String</th>
<th></th>
<th></th>
<th>Data Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>02</td>
<td></td>
<td>00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

c) length $\geq 8$?  
d) find a 0x00?
Bleichenbacher’s Attack

- From implementation flaws, attackers might have access to an Oracle, which tells whether certain ciphertext is valid
  - Pick a random $r$, if $c' = r^e c \mod n$ is a valid ciphertext, we know $r m \mod n$ starts with 00 02
  - $r m \mod n$ starts with 00 02 limits $m$ to a smaller interval
  - Repeat until we can determine the value of $m$
Bleichenbacher’s Attack

- From implementation flaws, attackers might have access to an Oracle, which tells whether certain ciphertext is valid
- Significantly improved in 2012 (For RSA-1024, Mean:49001, Median:14501) [Bardou et al., 2012]
- Focus on the 00 02 in the start, ignore other conditions
- Easy countermeasure: OAEP
PKCS#1 v1.5 still matters

Only for backward compatibility, yet widespread

- TLS doesn’t support OAEP yet
- “Backward compatibility Attacks” [Jager et al., 2013]
This work is about...

Implementations don’t follow the standard step by step

- Can implementation tricks provide a “shelter”?
More specifically...

- For instance, in decryption process, after RSA decryption:

```
XX XX Padding String 00 Data Block
```

- "efficient" way to implement PKCS #1 v1.5
- Bleichenbacher’s Attack doesn’t work (no 00 02)
- leakage still exists, although very vague
- not secure, but might be good enough
Main topic

**Question** Can this kind of implementation tricks serve as a temporary “shelter”?
Outline

Introduction
   Basic Problem
   Motivation

Non-standard implementation study
   Case I
   Case II
   Categorize Non-standard implementation
   Further improvement*

Summary
Case I: Basic idea

- Oracle tells us whether $rm \mod n$ has a 0 byte
- No clear expression available (typically, this part of the information is ignored by Bleichenbacher’s Attack)

**Straightforward idea**
Can we get some clear information from this Oracle?
Property of this Oracle

If $m$ has a 0 byte, $256m \mod n$ must has a 0 byte, unless $256m > n$. Thus

**Property 1**
Let $c = m^e \mod n$, $c' = 256^e c \mod n$, if $O(c) = T$ and $O(c') = F$, we can conclude that $256m > n$

Extended to $[rn, (r + 1)n)$:

**Property 2**
Let $c_1 = (sm)^e \mod n$, $c_2 = (256sm)^e \mod n$, $sm \in [rn, (r + 1)n)$, if $O(c_1) = T$ and $O(c_2) = F$, we can conclude that $sm > rn + \frac{1}{256}n$
Simple idea, but....... 

Implementation requirements:

- The 0 byte in the two most significant positions doesn’t count (00 02).
- The 0 byte in the least significant position doesn’t count (no data block).
Further analysis

- Even if we take both $m$ and $256m \mod n$ into consideration, we can not guarantee anything from a single Oracle access.
- The problem above rarely happens: if we can collect some points near $m$ and get corresponding replies from the Oracle, we can decide whether $256m > n$ through probability analysis.
More specifically...

<table>
<thead>
<tr>
<th>PO(c)</th>
<th>PO(c')</th>
<th>256m &lt; n</th>
<th>256m &gt; n</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>38.21%</td>
<td>14.86%</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>0.25%</td>
<td>23.50%</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>0.23%</td>
<td>23.76%</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>61.31%</td>
<td>37.88%</td>
</tr>
</tbody>
</table>

• Clearly, if we collect some points near \( m \), and find many \( O(c) \neq O(c') \), we can conclude that \( 256m > n \).
Distinguish algorithm

Require: Padding Oracle PO, ciphertext $c$, $m$'s current interval $[a, b]$, $r$
compute the possible interval of $s$ $[s_{\text{min}}, s_{\text{max}}]$

$$s_{\text{min}} = \left\lfloor \frac{(r + \frac{1}{256})n}{b} \right\rfloor, \quad s_{\text{max}} = \left\lceil \frac{(r + \frac{1}{256})n}{a} \right\rceil$$ —— Make sure $s_{\text{min}}$ is near $(r + \frac{1}{256})n$

while $s < s_{\text{max}}$ do
    collect some sample points near current $s$, $c_1 = s^e c \mod n$, $c_2 = (256s)^e c \mod n$
call PO to calculate $\Pr(PO(c_1) \neq PO(c_2))$
if $\Pr(PO(c_1) \neq PO(c_2)) > \text{threshold}$ then
    break
else
    $s++$
end if
end while

$$a' = \left\lceil \frac{r + \frac{1}{256}n}{s + \text{block} + 1} \right\rceil, \quad b' = \left\lfloor \frac{r + \frac{1}{256}n}{s - \text{block}} \right\rfloor$$

return $[a', b']$
Distinguish algorithm

Require: Padding Oracle PO, ciphertext c, m’s current interval [a, b], r
compute the possible interval of s [s_{min}, s_{max}]

\[ s_{min} = \left\lfloor \frac{(r + \frac{1}{256})n}{b} \right\rfloor, s_{max} = \left\lceil \frac{(r + \frac{1}{256})n}{a} \right\rceil \]

while \( s < s_{max} \) do
    collect some sample points near current s, \( c_1 = s^{e}c \mod n \), \( c_2 = (256s)^{e}c \mod n \)
call PO to calculate \( Pr(PO(c_1) \neq PO(c_2)) \)
if \( Pr(PO(c_1) \neq PO(c_2)) > \text{threshold} \) then
    break —— This tells us \( s^m > \left( r + \frac{1}{256} \right) n \)
else
    \( s++ \)
end if
end while

\( a' = \left\lceil \frac{r + \frac{1}{256}n}{s + \text{block} + 1} \right\rceil, b' = \left\lfloor \frac{r + \frac{1}{256}n}{s - \text{block}} \right\rfloor \)

return \([a', b']\)
Distinguish algorithm

**Require:** Padding Oracle PO, ciphertext $c$, $m$'s current interval $[a, b]$, $r$

compute the possible interval of $s$ $[s_{\text{min}}, s_{\text{max}}]$

$s_{\text{min}} = \left\lfloor \frac{r + \frac{1}{256} b}{n} \right\rfloor$, $s_{\text{max}} = \left\lceil \frac{r + \frac{1}{256} a}{n} \right\rceil$

**while** $s < s_{\text{max}}$ **do**

collect some sample points near current $s$, $c_1 = s^e c \mod n$, $c_2 = (256s)^e c \mod n$
call PO to calculate $Pr(PO(c_1) \neq PO(c_2))$

**if** $Pr(PO(c_1) \neq PO(c_2)) > \text{threshold}$ **then**

break

**else**

$s++$

**end if**

**end while**

$a' = \left\lfloor \frac{r + \frac{1}{256} n}{s + \text{block} + 1} \right\rfloor$, $b' = \left\lfloor \frac{r + \frac{1}{256} n}{s - \text{block}} \right\rfloor$ —— Update the interval of $m$

**return** $[a', b']$
Distinguish algorithm

**Require:** Padding Oracle PO, ciphertext \( c \), \( m \)'s current interval \([a, b]\), \( r \)

compute the possible interval of \( s \) \([s_{\text{min}}, s_{\text{max}}]\)

\[
\begin{align*}
s_{\text{min}} &= \left\lfloor \frac{(r + \frac{1}{256})n}{b} \right\rfloor, \\
s_{\text{max}} &= \left\lceil \frac{(r + \frac{1}{256})n}{a} \right\rceil
\end{align*}
\]

**while** \( s < s_{\text{max}} \)** do**

collect some sample points near current \( s \), \( c_1 = s^e c \mod n \), \( c_2 = (256s)^e c \mod n \)

call PO to calculate \( Pr(PO(c_1) \neq PO(c_2)) \)

**if** \( Pr(PO(c_1) \neq PO(c_2)) > \text{threshold} \)** then

break

**else**

\( s++ \)

**end if**

**end while**

\( a' = \left\lfloor \frac{r + \frac{1}{256}n}{s + \text{block} + 1} \right\rfloor, \quad b' = \left\lceil \frac{r + \frac{1}{256}n}{s - \text{block}} \right\rceil \)

**return** \([a', b']\)

Repeat with larger \( r \), until there is only one \( m \) left in \([a, b]\)
Some other troubles...

“stuck” problem

- After the first round \(r=0\), \([s_{\text{min}}, s_{\text{max}}]\) is too small
- No enough points for analysis (stuck)

Solution:
- multiplier 2: more powerful, less efficient

<table>
<thead>
<tr>
<th>PO(c)</th>
<th>PO(c')</th>
<th>2m &lt; n</th>
<th>2m &gt; n</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>25.19%</td>
<td>15.01%</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>13.20%</td>
<td>23.75%</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>13.35%</td>
<td>23.60%</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>48.25%</td>
<td>37.64%</td>
</tr>
</tbody>
</table>
Some other troubles...

“stuck” problem

- After the first round (r=0), \([s_{\min}, s_{\max}]\) is too small
- No enough points for analysis (stuck)

Solution:

- multiplier 2: more powerful, less efficient

<table>
<thead>
<tr>
<th>PO(c)</th>
<th>PO(c')</th>
<th>2m &lt; n</th>
<th>2m &gt; n</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>25.19%</td>
<td>15.01%</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>13.20%</td>
<td>23.75%</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>13.35%</td>
<td>23.60%</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>48.25%</td>
<td>37.64%</td>
</tr>
</tbody>
</table>
Complete attack for case I

Complete attack algorithm:

- Prefer multiply 256 method (efficiency)
- Typically, only the first few (4-5) rounds need multiply 2 method
- About 0.11 million oracle accesses for RSA-1024 (stable performance)

Works when implementation also checks PS length

---

<table>
<thead>
<tr>
<th>XX</th>
<th>XX</th>
<th>Padding String</th>
<th>00</th>
<th>Data Block</th>
</tr>
</thead>
</table>

---

Search for the first 0x00

c) length \( \geq 8 \) ?

d) find a 0x00 ?
Outline

Introduction
  Basic Problem
  Motivation

Non-standard implementation study
  Case I
  Case II
  Categorize Non-standard implementation
  Further improvement*

Summary
Another case...

Now let's move on

- much easier to analyze (0x00 in the start)
- Any $m$ passes the check means $m$ must start with 0x00
Oracle Analysis

Let \( T = 2^{8(k-1)} \), Oracle’s behavior:

- For each \( r \), both \( rn \) and \( rn + T \) can be used
- Similar as before, need both to complete the attack
- Much more efficient, 12 000 oracle access for RSA-1024 (stable performance)
- Also works when PS length is checked
Outline

Introduction
  Basic Problem
  Motivation

Non-standard implementation study
  Case I
  Case II
  Categorize Non-standard implementation
  Further improvement*

Summary
Categorize Non-standard implementation

Stronger and weaker:

- a) and b) are (00 02) stronger checks (more leakage)
- Relatively, c) and d) are weaker (less leakage)
- stronger checks dominate the corresponding analysis
- Thus, Non-standard implementations can be divided into 4 Groups:
  - Group I (none): case I
  - Group II (only 02): probably variant of Bleichenbacher’s Attack
  - Group III (only 00): case II
  - Group IV (both): Bleichenbacher’s Attack
Table 1. Implementation types with corresponding attack algorithm

<table>
<thead>
<tr>
<th>Type</th>
<th>Conditions</th>
<th>Group</th>
<th>Available Attack Algorithm</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a) b) c) d)</td>
<td></td>
<td></td>
<td>Median</td>
</tr>
<tr>
<td>1</td>
<td>✓</td>
<td>Group I</td>
<td>Group I Attack</td>
<td>113 520</td>
</tr>
<tr>
<td>2</td>
<td>✓</td>
<td>Group I</td>
<td>Unnecessary</td>
<td>—</td>
</tr>
<tr>
<td>3</td>
<td>✓ ✓</td>
<td>Group I</td>
<td>Group I Attack</td>
<td>111 890</td>
</tr>
<tr>
<td>4</td>
<td>✓</td>
<td>Group II</td>
<td>Variant of Bleichenbacher’s attack</td>
<td>—</td>
</tr>
<tr>
<td>5</td>
<td>✓ ✓</td>
<td>Group II</td>
<td></td>
<td>—</td>
</tr>
<tr>
<td>6</td>
<td>✓ ✓</td>
<td>Group II</td>
<td></td>
<td>—</td>
</tr>
<tr>
<td>7</td>
<td>✓ ✓ ✓</td>
<td>Group II</td>
<td></td>
<td>—</td>
</tr>
<tr>
<td>8</td>
<td>✓</td>
<td>Group III</td>
<td>Manger’s attack</td>
<td>1 168</td>
</tr>
<tr>
<td>9</td>
<td>✓ ✓</td>
<td>Group III</td>
<td>Group III Attack</td>
<td>12 843</td>
</tr>
<tr>
<td>10</td>
<td>✓ ✓</td>
<td>Group III</td>
<td>Unnecessary</td>
<td>—</td>
</tr>
<tr>
<td>11</td>
<td>✓ ✓ ✓</td>
<td>Group III</td>
<td>Group III Attack</td>
<td>13 047</td>
</tr>
<tr>
<td>12</td>
<td>✓ ✓</td>
<td>Group IV</td>
<td>Bleichenbacher’s attack</td>
<td>4 762</td>
</tr>
<tr>
<td>13</td>
<td>✓ ✓ ✓</td>
<td>Group IV</td>
<td>Bleichenbacher’s attack</td>
<td>15 315</td>
</tr>
<tr>
<td>14</td>
<td>✓ ✓ ✓</td>
<td>Group IV</td>
<td>Unnecessary</td>
<td>—</td>
</tr>
<tr>
<td>15</td>
<td>✓ ✓ ✓ ✓</td>
<td>Group IV</td>
<td>Bleichenbacher’s attack</td>
<td>17 473</td>
</tr>
</tbody>
</table>
Outline

Introduction
  Basic Problem
  Motivation

Non-standard implementation study
  Case I
  Case II
  Categorize Non-standard implementation
  Further improvement*

Summary
Further improvement*

Although practical, our attack for case I isn’t efficient enough:
- Idea comes from Dr. Cheng Chen
- push the “0x00 ”to the start

\[ C = 2^{8(k-3)}, \ k \text{ is the byte length of } n \]
Further improvement*

Although practical, our attack for case I isn’t efficient enough:

- Idea comes from Dr. Cheng Chen
- push the “0x00 ” to the start
- Let $C = 2^{8(k-3)}$, $k$ is the byte length of $n$
Further improvement

- Looks like case II, yet slightly different from case II
- Similar to the Step 2.c in Bleichenbacher’s attack

\[ m(rn+C)/b \]  
\[ m(rn+C)/a \]

Any F means \( sm > rn+C \)

- To ensure this works, start with several rounds of our case I attack
- about 20 000 oracle access (80% off, stable)
- Also works when implementation check PS length (more rounds of case I attack)
Outline

Introduction
  Basic Problem
  Motivation

Non-standard implementation study
  Case I
  Case II
  Categorize Non-standard implementation
  Further improvement*

Summary
Summary

• We propose analysis for two types of non-standard implementations
• Together with Bleichenbacher’s attack, we can conclude most of non-standard implementations are vulnerable
• Moreover, some non-standard implementations are even worse!
THE END

Thanks for listening!
