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

Outline

Introduction
- Basic Problem
- Motivation

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

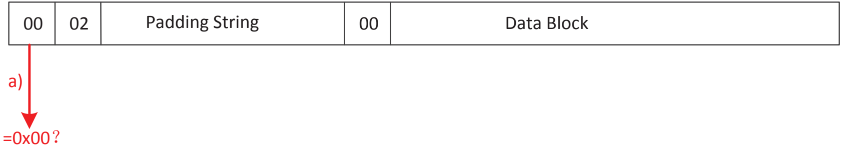
Summary

Outline

PKCS#1 v1.5

PKCS#1 v1.5 coding for encryption

A valid “padded message” should satisfy:



PKCS#1 v1.5

PKCS#1 v1.5 coding for encryption

A valid “padded message” should satisfy:

00	02	Padding String	00	Data Block

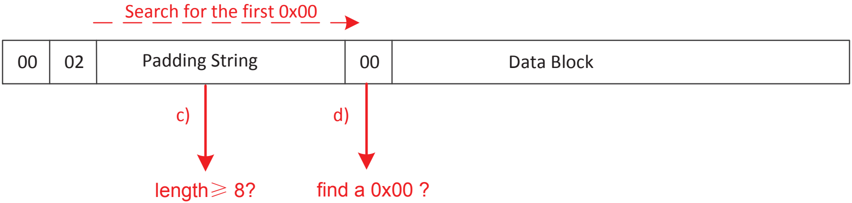
b)

=0x02？

PKCS#1 v1.5

PKCS#1 v1.5 coding for encryption

A valid “padded message” should satisfy:



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 $c' = r^e c \bmod n$ is a valid ciphertext, we know $rm \bmod n$ starts with 00 02
  - $rm \bmod n$ 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

- More usage than OAEP [European Network of Excellence in Cryptology II, 2012]
- TLS doesn’t support OAEP yet
- “Backward compatibility Attacks” [Jager *et al.*, 2013]

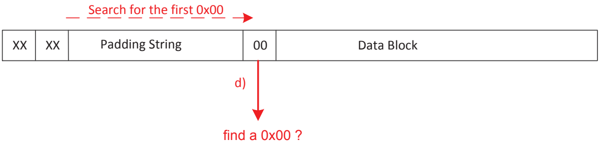
This work is about...

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:



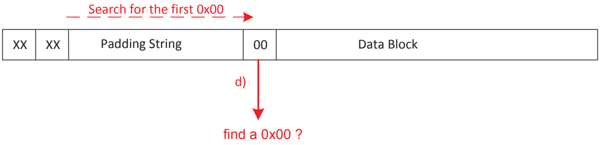
- “efficient” way to implement PKCS #1 v1.5
- Bleichenbacher’s Attack doesn’t work (no 00 02)
- leakage still exists, although very vague
- not secure, but might be good enough

Main topic

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

Outline

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

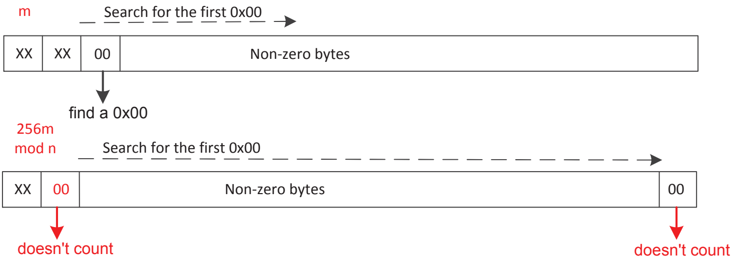
Property 2

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 $256m \bmod n$ 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...

PO(c)	PO(c')	$256m < n$	$256m > n$
T	T	38.21%	14.86%
T	F	0.25%	23.50%
F	T	0.23%	23.76%
F	F	61.31%	37.88%

- 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 $[s_{min}, s_{max}]$

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

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

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

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

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

    break

  else

    s++

  end if

end while

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

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, $c_1 = s^e c \bmod n$, $c_2 = (256s)^e c \bmod n$

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

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

    break —— This tells us $sm > \left(r + \frac{1}{256}\right) n$

  else

    s++

  end if

end while

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

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, $c_1 = s^e c \bmod n$, $c_2 = (256s)^e c \bmod n$

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

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

    break

  else

    s++

  end if

end while

$a' = \left\lceil \frac{r+\frac{1}{256}n}{s+block+1} \right\rceil, b' = \left\lfloor \frac{r+\frac{1}{256}n}{s-block} \right\rfloor$ —— 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 $[s_{min}, s_{max}]$

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

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

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

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

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

    break

  else

    $s{+}{+}$

  end if

end while

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

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), $[s_{min}, s_{max}]$ is too small
- No enough points for analysis(stuck)

Solution:

- multiplier 2: more powerful, less efficient

PO(c)	PO(c')	$2m < n$	$2m > n$
T	T	25.19%	15.01%
T	F	13.20%	23.75%
F	T	13.35%	23.60%
F	F	48.25%	37.64%

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

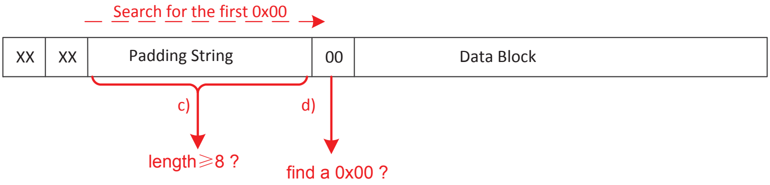
PO(c)	PO(c')	$2m < n$	$2m > n$
T	T	25.19%	15.01%
T	F	13.20%	23.75%
F	T	13.35%	23.60%
F	F	48.25%	37.64%

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)

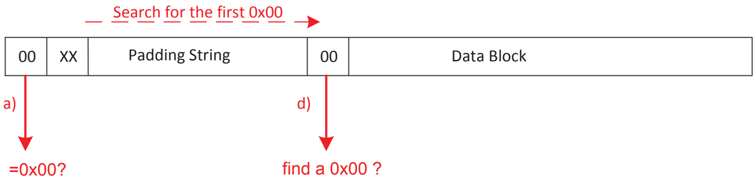
Works when implementation also checks PS length



Outline

Another case...

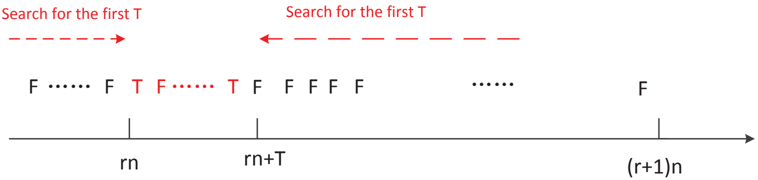
Now let's move on



- 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

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

Global view

Table 1. Implementation types with corresponding attack algorithm

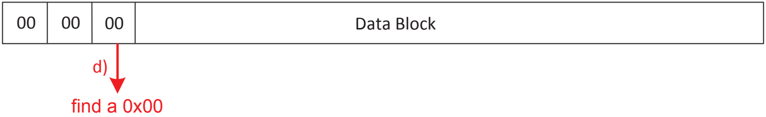
Type	Conditions a)	Conditions b)	Conditions c)	Conditions d)	Group	Available Attack Algorithm	Performance Median	Performance Mean
1				✓	Group I	Group I Attack	113 520	115 978
2			✓		Group I	Unnecessary	—	—
3			✓	✓	Group I	Group I Attack	111 890	114 331
4		✓			Group II	Variant of Bleichenbacher's attack	—	—
5		✓		✓	Group II	Variant of Bleichenbacher's attack	—	—
6		✓	✓		Group II	Variant of Bleichenbacher's attack	—	—
7		✓	✓	✓	Group II	Variant of Bleichenbacher's attack	—	—
8	✓				Group III	Manger's attack	1 168	1 174
9	✓			✓	Group III	Group III Attack	12 843	12 878
10	✓		✓		Group III	Unnecessary	—	—
11	✓		✓	✓	Group III	Group III Attack	13 047	13 058
12	✓	✓			Group IV	Bleichenbacher's attack	4 762	14 532
13	✓	✓		✓	Group IV	Bleichenbacher's attack	15 315	92 820
14	✓	✓	✓		Group IV	Unnecessary	—	—
15	✓	✓	✓	✓	Group IV	Bleichenbacher's attack	17 473	104 839

Outline

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

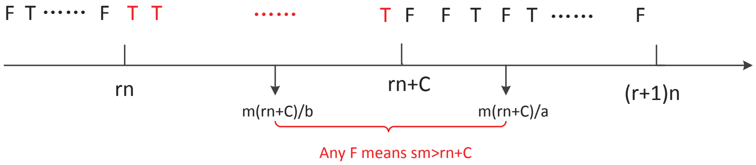
Further improvement*

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



- 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

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!

Bardou, Romain, Focardi, Riccardo, Kawamoto, Yusuke, Simionato, Lorenzo, Steel, Graham, & Tsay, Joe-Kai. 2012. Efficient Padding Oracle Attacks on Cryptographic Hardware. *Pages 608–625 of:* Safavi-Naini, Reihaneh, & Canetti, Ran (eds), *Advances in Cryptology — CRYPTO 2012*. Lecture Notes in Computer Science, vol. 7417. Springer Berlin Heidelberg.

European Network of Excellence in Cryptology II. 2012 (30 Sept). *ECRYPT II Yearly Report on Algorithms and Keysizes (2011-2012)*. Tech. rept. European Network of Excellence in Cryptology II.

Jager, Tibor, Paterson, Kenneth G., & Somorovsky, Juraj. 2013. One Bad Apple: Backwards Compatibility Attacks on State-of-the-Art Cryptography. *In: NDSS Symposium 2013*.