Dobbertin Challenge 2012

Christian Becker

Introduction

Given information

About the service:

- Location: `cryptochallenge.nds.rub.de:50080/service`
- A user can send his encrypted PIN to the Web Service, which decrypts and stores the PIN
- The Web Service allows to use different cryptographical algorithms
  - Strong: RSA-OAEP, AES in GCM-mode
  - Weak: RSA-PKCS#1 v1.5, AES in CBC-mode
- The Web Service accepts messages, which correspond to the JSON Web Encryption standard
- Server messages:
  - Data successfully stored
  - Couldn't decrypt: data hash wrong
  - Couldn't decrypt: mac check in GCM failed
  - Couldn't decrypt: pad block corrupted
  - Unknown exception

Given information

About the task:

- We are an attacker who eavesdropped a ciphertext whicht contains Bob's PIN
- The ciphertext consists of three parts (all base64 encoded)
  - Information about the choice of algorithms used to encrypt this ciphertext
  - An asymmetric ciphertext (RSA-OAEP or RSA-PKCS#1 v1.5), which encrypts a symmetric session key
  - A symmetric ciphertext (AES-CBC or AES-GCM), which contains the payload, encrypted with the symmetric session key
- The plaintext has the format {"My PIN:":"<PIN>"}

Given information

Ciphertext

eyJhbGciOiJSU0FfT0FFUCIsIml2IjoieXY2NnZ2ck8yNjNleXZpSSIsInR5
cCI6IkpXVCIsImVuYyI6IkExMjhHQ00ifQ==.
ZBnPlwONWHxGDrtCxxopS4y4SrMZIAhUg3HI+SbLMxfPVRPW8yunejrkmfSL
O1H/OtOx4ssggygHjG7sUfxL8A==.
i2vygn2vqFpsmep3etrD5Yh5xLP9xYhJdvn63WmHEPYChA==.

Given information

Ciphertext

{’’alg’’:’’RSA_OAEP’’,’’iv’’:’’yv66vvrO263eyviI’’,
’’typ’’:’’JWT’’,’’enc’’:’’A128GCM’’}.
ZBnPlwONWHxGDrtCxxopS4y4SrMZIAhUg3HI+SbLMxfPVRPW8yunejrkmfSL
O1H/OtOx4ssggygHjG7sUfxL8A==.
i2vygn2vqFpsmep3etrD5Yh5xLP9xYhJdvn63WmHEPYChA==.

Attacking the service

There are two ways to attack the service

- Attack on AES in CBC-mode
- Attack on RSA-PKCS#1 v1.5

Attack on AES in CBC-mode

Galois Counter Mode

Some facts:

- GCM is an encryption mode which also computes a MAC (message authentication, integrity) [irrelevant]
- Encryption in counter mode (confidentiality) [relevant]
- Additional authenticated data (authenticity), which is padded to the ciphertext [irrelevant]

Galois Counter Mode

Some facts:

- GCM is an encryption mode which also computes a MAC (message authentication, integrity) [irrelevant]
- Encryption in counter mode (confidentiality) [relevant]
- Additional authenticated data (authenticity), which is padded to the ciphertext [irrelevant]

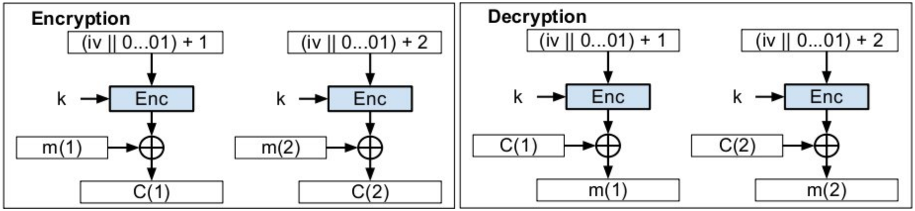
Encryption (relevant parts):

- Data must have a block size of 128 bit
- Encrypting the data using the Counter Mode (CTR)

Counter Mode

- IV with length <128 bit
- The remaining bits are the counter, which is initialized to zero
- For every block the counter will be incremented

Encryption: $y_i = e_k(IV||CTR_i) \oplus x_i \; i \geq 1$
Decryption: $x_i = e_k(IV||CTR_i) \oplus y_i \; i \geq 1$



Source: One Bad Apple: Backwards Compatibility Attacks on State-of-the-Art Cryptography

Galois Counter Mode

Some facts:

- GCM is an encryption mode which also computes a MAC (message authentication, integrity) [irrelevant]
- Encryption in counter mode (confidentiality) [relevant]
- Additional authenticated data (authenticity), which is padded to the ciphertext [irrelevant]

Encryption (relevant parts):

- Data must have a block size of 128 bit
- Encrypting the data using the Counter Mode (CTR)

Choose:

- $J_0 = (IV \,||\, 0^{31} \,||\, 1)_2$, [96 bit + 31 bit + 1 bit]
- $C = GCTR(J_0, x)$

Given IV

Encryption

IV = base64_decode(yv66vvrO263eyvil) , [96 bit]

J_0 = IV || 0^{31} || 1

J_0 = ca fe ba be fa ce db ad de ca f8 88 || 00 00 00 01

C_0 = GCTR(J_0,x)

C_0 = AES-Enc$_k$(ca fe ba be fa ce db ad de ca f8 88 || 00 00 00 02) $\oplus$ x

Transform from GCM to CBC

Some facts:

- length(IV) = length(x) = length (y)

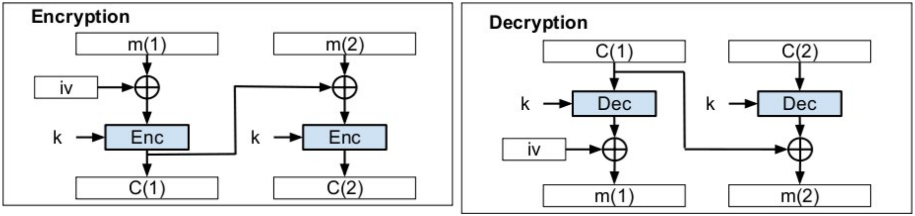
CBC mode

Encryption (first block): $y_1 = e_k(x_1 \oplus IV)$

Encryption: $y_i = e_k(x_i \oplus y_{i-1}), i \geq 2$

Decryption (first block): $x_1 = e_k(y_1) \oplus IV$

Decryption: $x_i = e_k(y_i) \oplus y_{i-1}, i \geq 2$



Source: One Bad Apple: Backwards Compatibility Attacks on State-of-the-Art Cryptography

Enough theory let's start an attack!

Step 1

Since our IV is only 12 bytes long we have to expand it according to the GCTR function.

GCE

IV = base64_decode(yv66vvrO263eyvil) , [96 bit]

J_0 = IV || 0^{31} || 1

J_0 = ca fe ba be fa ce db ad de ca f8 88 || 00 00 00 01

C_0 = GCTR(J_0,x)

C_0 = AES-Enc$_k$(ca fe ba be fa ce db ad de ca f8 88 || 00 00 00 02) $\oplus$ x

newIV = ca fe ba be fa ce db ad de ca f8 88 00 00 00 02

The given ciphertext is 34 bytes long, but we are only interested in the first block.

Cipher

C' = substr(Cipher,0,16)

Step 2

We have to change the header to enable the CBC mode

Header

```
{"alg":"RSA_OAEP","iv":"yv66vvrO263eyviI",
"typ":"JWT","enc":"A128GCM"}.
```

New Header

```
base64_encode({"alg":"RSA_OAEP","iv":"encode_base64(cafebabef
acedbaddecaf88800000002)","typ":"JWT","enc":"A128CBC"}).
```

Step 3

We can use the service as a padding oracle. For this we compute:

M’ = base64_encode({"My PIN:":"XXXX), with XXXX in [0000,9999]

and send

New request

```
base64_encode({”alg”:”RSA_OAEP”,”iv”:”encode_base64(cafebabef
acedbaddecaf88800000002)”,”typ”:”JWT”,”enc”:”A128CBC”}).
ZBnPlwONWHxGDrtCxxopS4y4SrMZIAhUg3HI+SbLMxfPVRPW8yunejrkmfSL
O1H/OtOx4ssggygHjG7sUfxL8A==.
base64_encode(M’ ⊕ C’)
```

Step 3

The service will decrypt this to

Decryption progress

$\texttt{AES-Dec}_k(\texttt{y}_1) \oplus \texttt{IV}$

Step 3

The service will decrypt this to

Decryption progress

$\texttt{AES-Dec}_k(\texttt{y}_1) \oplus \texttt{IV}$

$\texttt{AES-Dec}_k(\texttt{M'} \oplus \texttt{C'}) \oplus \texttt{IV}$

Step 3

The service will decrypt this to

Decryption progress

$\texttt{AES-Dec}_k(\texttt{y}_1) \oplus \texttt{IV}$

$\texttt{AES-Dec}_k(\texttt{M'} \oplus \texttt{C'}) \oplus \texttt{IV}$

$\texttt{AES-Dec}_k(\texttt{M'} \oplus \texttt{AES-Enc}_k(\texttt{ca fe ba be fa ce db ad de ca f8 88 || 00 00 00 02}) \oplus \texttt{x}) \oplus \texttt{IV}$

Step 3

The service will decrypt this to

Decryption progress

$\texttt{AES-Dec}_k(\texttt{y}_1) \oplus \texttt{IV}$

$\texttt{AES-Dec}_k(\texttt{M'} \oplus \texttt{C'}) \oplus \texttt{IV}$

$\texttt{AES-Dec}_k(\texttt{M'} \oplus \texttt{AES-Enc}_k(\texttt{ca fe ba be fa ce db ad de ca f8 88 || 00 00 00 02}) \oplus \texttt{x}) \oplus \texttt{IV}$

If M' = x

Decryption progress for M' = x

Step 3

The service will decrypt this to

Decryption progress

$\texttt{AES-Dec}_k(\texttt{y}_1) \oplus \texttt{IV}$

$\texttt{AES-Dec}_k(\texttt{M'} \oplus \texttt{C'}) \oplus \texttt{IV}$

$\texttt{AES-Dec}_k(\texttt{M'} \oplus \texttt{AES-Enc}_k(\texttt{ca fe ba be fa ce db ad de ca f8 88 || 00 00 00 02}) \oplus \texttt{x}) \oplus \texttt{IV}$

If M' = x

Decryption progress for M' = x

$\texttt{AES-Dec}_k(\texttt{AES-Enc}_k(\texttt{ca fe ba be fa ce db ad de ca f8 88 || 00 00 00 02})) \oplus \texttt{IV}$

Step 3

The service will decrypt this to

Decryption progress

$\texttt{AES-Dec}_k(\texttt{y}_1) \oplus \texttt{IV}$

$\texttt{AES-Dec}_k(\texttt{M'} \oplus \texttt{C'}) \oplus \texttt{IV}$

$\texttt{AES-Dec}_k(\texttt{M'} \oplus \texttt{AES-Enc}_k(\texttt{ca fe ba be fa ce db ad de ca f8 88 || 00 00 00 02}) \oplus \texttt{x}) \oplus \texttt{IV}$

If M' = x

Decryption progress for M' = x

$\texttt{AES-Dec}_k(\texttt{AES-Enc}_k(\texttt{ca fe ba be fa ce db ad de ca f8 88 || 00 00 00 02})) \oplus \texttt{IV}$

$\texttt{IV} \oplus \texttt{IV}$

Step 3

The service will decrypt this to

Decryption progress

$\texttt{AES-Dec}_k(\texttt{y}_1) \oplus \texttt{IV}$

$\texttt{AES-Dec}_k(\texttt{M'} \oplus \texttt{C'}) \oplus \texttt{IV}$

$\texttt{AES-Dec}_k(\texttt{M'} \oplus \texttt{AES-Enc}_k(\texttt{ca fe ba be fa ce db ad de ca f8 88 || 00 00 00 02}) \oplus \texttt{x}) \oplus \texttt{IV}$

If M' = x

Decryption progress for M' = x

$\texttt{AES-Dec}_k(\texttt{AES-Enc}_k(\texttt{ca fe ba be fa ce db ad de ca f8 88 || 00 00 00 02})) \oplus \texttt{IV}$

$\texttt{IV} \oplus \texttt{IV}$

$\texttt{0}$

But the oracle answers Couldn't decrypt: pad block corrupted, because it is not a valid PKCS#7

Some theory again

We need a padding for the CBC mode, which can be looked up in this table:

```
PS = 01                               if len(P) mod 128 = 120,
PS = 0202                             if len(P) mod 128 = 112,
PS = 030303                           if len(P) mod 128 = 104,
PS = 04040404                         if len(P) mod 128 = 96,
PS = 0505050505                       if len(P) mod 128 = 88,
PS = 060606060606                     if len(P) mod 128 = 80,
PS = 07070707070707                   if len(P) mod 128 = 72,
PS = 0808080808080808                 if len(P) mod 128 = 64,
PS = 090909090909090909               if len(P) mod 128 = 56,
PS = 0A0A0A0A0A0A0A0A0A0A             if len(P) mod 128 = 48,
PS = 0B0B0B0B0B0B0B0B0B0B0B           if len(P) mod 128 = 40,
PS = 0C0C0C0C0C0C0C0C0C0C0C0C         if len(P) mod 128 = 32,
PS = 0D0D0D0D0D0D0D0D0D0D0D0D0D       if len(P) mod 128 = 24,
PS = 0E0E0E0E0E0E0E0E0E0E0E0E0E0E     if len(P) mod 128 = 16,
PS = 0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F   if len(P) mod 128 = 8,
PS = 10101010101010101010101010101010 if len(P) mod 128 = 0.
```

To get the desired padding we $\oplus$ 0x10 * 16 with our IV before sending it to the server.

newIV = ca fe ba be fa ce db ad de ca f8 88 00 00 00 02 $\oplus$
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

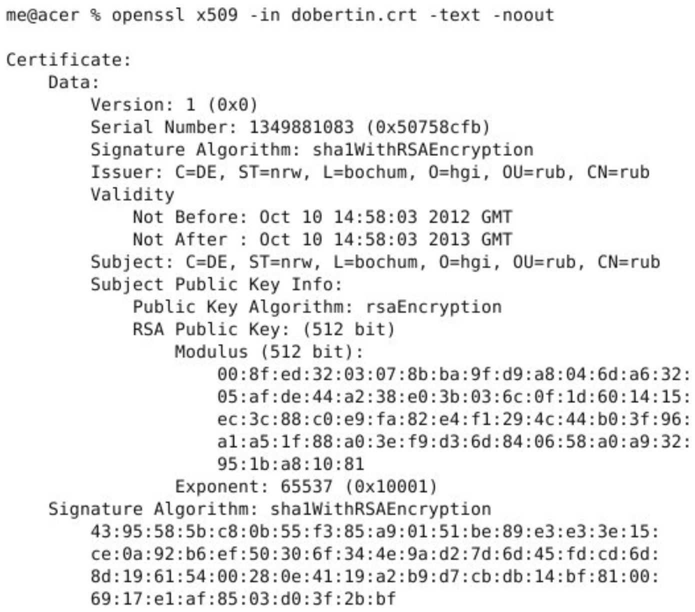
If we choose the right PIN the padding will be correct and we should get the answer Data successfully stored.

Final step

Testing all possible PINS from 0000 to 9999 returns one valid PIN which is 5983.

Attack on RSA-PKCS#1 v1.5

Additional information



```
me@acer % openssl x509 -in dobertin.crt -text -noout

Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number: 1349881083 (0x50758cfb)
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=DE, ST=nrw, L=bochum, O=hgi, OU=rub, CN=rub
        Validity
            Not Before: Oct 10 14:58:03 2012 GMT
            Not After : Oct 10 14:58:03 2013 GMT
        Subject: C=DE, ST=nrw, L=bochum, O=hgi, OU=rub, CN=rub
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (512 bit)
                Modulus (512 bit):
                    00:8f:ed:32:03:07:8b:ba:9f:d9:a8:04:6d:a6:32:
                    05:af:de:44:a2:38:e0:3b:03:6c:0f:1d:60:14:15:
                    ec:3c:88:c0:e9:fa:82:e4:f1:29:4c:44:b0:3f:96:
                    a1:a5:1f:88:a0:3e:f9:d3:6d:84:06:58:a0:a9:32:
                    95:1b:a8:10:81
                Exponent: 65537 (0x10001)
    Signature Algorithm: sha1WithRSAEncryption
        43:95:58:5b:c8:0b:55:f3:85:a9:01:51:be:89:e3:e3:3e:15:
        ce:0a:92:b6:ef:50:30:6f:34:4e:9a:d2:7d:6d:45:fd:cd:6d:
        8d:19:61:54:00:28:0e:41:19:a2:b9:d7:cb:db:14:bf:81:00:
        69:17:e1:af:85:03:d0:3f:2b:bf
```

RSA

- algorithm for public-key cryptography
- Public-key (e,N)
- Private-key (d,N,p,q)

RSA

Encryption: $y = x^e \bmod N$

Decryption: $x = y^d \bmod N$

PKCS#1 v1.5 Padding

```
00 || 02 || PS || 00 || D
```

- PS are random non-zero bytes, with length(PS) = k - |D| -3
- D is the message, with length(D) <= k-11

The padded message will be encrypted after the transformation

Theory

RSA-OAEP offers no useable side-channels so we have to attack RSA with PKCS#1 v1.5 Padding

1. Change the header from RSA_OAEP to RSA1_5
2. Use attack of Manger/Bleichenbacher to retrieve the padded plaintext message
3. Depad using OAEP
4. Decrypt the message using AES GCM and the secret key

Attack of Manger - Overview

Requirements:

- N [RSA modulus], e [public-key]
- k = length(N) [bytelength]
- B = $2^{8*(k-1)}$
- c [ciphertext] and unknown x [plaintext] $\in$ [0,B)
- An oracle, which indicates whether
  - x = c^d is PKCS#1 v1.5 conform (<B)
  - or not (>B)

This attack is based on the possibility of extending the ciphertext and limiting the value of x through an interval.

Attack of Manger - Overview

Extending the ciphertext

$c = x^e \bmod N$

Attack of Manger - Overview

Extending the ciphertext

c = x^e mod N

c' = s^e * c mod N

Attack of Manger - Overview

Extending the ciphertext

c = x^e mod N
c' = s^e * c mod N
c' = s^e * x^e mod N

Attack of Manger - Overview

Extending the ciphertext

$c = x^e \bmod N$

$c' = s^e * c \bmod N$

$c' = s^e * x^e \bmod N$

Decryption of the extended ciphertext

$x' = (c')^d \bmod N$

Attack of Manger - Overview

Extending the ciphertext

$c = x^e \bmod N$

$c' = s^e * c \bmod N$

$c' = s^e * x^e \bmod N$

Decryption of the extended ciphertext

$x' = (c')^d \bmod N$

$x' = [(s^e * x^e)]^d \bmod N$

Attack of Manger - Overview

Extending the ciphertext

$c = x^{e} \bmod N$

$c' = s^{e} * c \bmod N$

$c' = s^{e} * x^{e} \bmod N$

Decryption of the extended ciphertext

$x' = (c')^{d} \bmod N$

$x' = [(s^{e} * x^{e})]^{d} \bmod N$

$x' = s^{(ed)} * x^{(ed)} \bmod N$

Attack of Manger - Overview

Extending the ciphertext

$c = x^e \bmod N$

$c' = s^e * c \bmod N$

$c' = s^e * x^e \bmod N$

Decryption of the extended ciphertext

$x' = (c')^d \bmod N$

$x' = [(s^e * x^e)]^d \bmod N$

$x' = s^{(ed)} * x^{(ed)} \bmod N$

$x' = s * x \bmod N$

During the attack of Manger an attacker chooses different values for s to minimize the interval up to the point where the difference is 0.
The last intervallimit is the padded plaintext
More information about the attack:

`http://archiv.infsec.ethz.ch/education/fs08/secsem/Manger01.pdf`

Demo

Secret AES GCM Key

The key bc071859b3e7901146608cb217638ecd can be used to decrypt the given cyphertext!

Questions?

Thank you for your attention.
Any questions?