

Challenge: 14-15-4ème étage

Write-up : Forgerie de Signature RSA - Challenge Badge SAS

1. Contexte du Challenge

On est dans le couloir 14-15/4 et en face on a :

- une salle serveur de la PPTI (Plateforme Pédagogique et Technologique Informatique) protégée par un système de badge.
- bureau d'un administrateur
- une salle TME ouverte (pour les challenges tme1 et tme2)
- une clé publique

Pour commencer on récupère la clé:

```
>>> voir clef
-----BEGIN PUBLIC KEY-----
MIIBIDANBgkqhkiG9w0BAQEFAAOCAQ0AMIIBCAKCAQEA7WuPBtWwMNJH6XmU3R0E
8wFpgdveiOW495mjCBfOj3CXxuigul91eeunRud9G9NifQHkt5PRGFcQqcTYgiMQ
Rxu2ASb5W/wNrDjuJsT7Z0LTmsSfDoKOTNr0L0kbEMy03YR0r4jm7TGl994gt4Qm
Copwr24CBmBkkaMrK6PWzFBlwxg1qQu58zd5YiyEALSkbEokfWrr8CI1hhNTy5zt
lIAncGYzLEniv2+CNzDcfZ9uXCtSqINZ/P0YLB0ddTAJAwpkiKoWDQIccV41Bl1U
LA9vYu9f0f0C1PgOzvavKr9pXL/fri/naQN08oScmpLQEJcKNznA8377ca/fFqva
TQIBAw=
-----END PUBLIC KEY-----
```

On accède ensuite à la salle serveur et l'affiche à côté du lecteur nous indique les conditions d'accès :

Affiche de sécurité :

ACCÈS AUX SERVEURS :

Votre badge de service doit contenir une signature qui prouve vos droit d'accès. Elle doit avoir l'ID "ppti_server_room".

Il doit s'agir d'une signature de la chaine :

```
"PPTI SERVER ACCESS ON yyyy-mm-dd hh:mm UTC"
```

La signature doit vérifier correctement avec la clef de la PPTI.
Elle doit obéir à la norme RSA / PKCS#1 v1.5.

1.1 Étape Préalable : Consulter l'Horloge Murale

Avant de forger la signature, il est CRUCIAL de consulter l'horloge murale dans la salle serveur, car le message à signer doit contenir la date/heure exacte en UTC.(on a ajouter 1min pour qu'on puisse avoir le temps de valider la signature)

```
>>> salle serveur
Vous êtes dans la salle serveur de la PPTI... ou plutôt, vous êtes devant
le sas du couloir froid qui contient les racks de serveurs de la PPTI.
Il y a un lecteur de badge à côté de la porte. Au mur une affiche indique :

+-----+
| ACCÈS AUX SERVEURS : |
|                         |
| Votre badge de service doit contenir une signature qui prouve vos |
| droit d'accès. Elle doit avoir l'ID ``ppti_server_room'`. |
|                         |
| Il doit s'agir d'une signature de la chaîne : |
| "PPTI SERVER ACCESS ON yyyy-mm-dd hh:mm UTC" |
| (en remplaçant yyyy par l'année, mm par le mois, dd par le jour |
| du mois, hh par l'heure et mm par les minutes). |
|                         |
| La signature doit vérifier correctement avec la clef de la PPTI. |
| Elle doit obéir à la norme RSA / PKCS#1 v1.5. |
|                         |
| Attention, la date doit être en temps universel (cf. horloge). |
|                         |
| *** Le responsable |
+-----+

Ici se trouve un écran LCD.
Ici se trouve un sas d'accès au couloir froid qui contient les serveurs.
Ici se trouve une horloge murale.
```

```
>>> voir horloge
Elle affiche :
2026-01-18 11:26:04
```

2. Analyse de la Vulnérabilité

2.1 Identification de la Faille

La vulnérabilité exploitée est une **attaque Cube Root Attack** sur RSA avec petit exposant ($e=3$).

Points clés :

- L'exposant public est très petit : **$e = 3$**
- Le message est padé en PKCS#1 v1.5
- Avec $e=3$, la signature valide s élevée au cube donne : $s^3 = \text{message_padé} \pmod{N}$
- **Si la racine cubique exacte du message padé est inférieure à N** , alors il n'y a pas de réduction modulo N !
- Cela signifie que la signature est simplement la racine cubique du bloc PKCS#1 v1.5

2.2 Conditions Requises

Pour que cette attaque fonctionne :

1. ☒ L'exposant public doit être petit ($e=3$)
2. ☒ Nous avons accès à la clé publique
3. ☒ Nous pouvons calculer le message padé
4. ☒ La racine cubique du message padé $< N$ (cas courant pour de petits e)

3. Implémentation de l'Exploit

3.1 Étapes de la Solution

Étape 1 : Charger la clé publique

```
from Crypto.PublicKey import RSA

def load_public_key(pem_file: str):
    with open(pem_file, 'r') as f:
        public_key = RSA.import_key(f.read())
    return public_key.n, public_key.e
```

Résultat :

- Module N : `0x...` (clé 1024-bit)
- Exposant e : **3** (La faille !)

```
Module N: 0xed6b8f06d5b030d247e97994dd1d04f3016981dbde88e5b8f799a30817ce8f7097c6e8a0ba5f7579eba746e77d1bd3627d01cab793
d1185710a9c4d8822310471bb60126f95bfc0dac38ee26c4fb6742d39ac49f0e828e4cdf42f491b10cc8edd8474af88e6ed31a5f7de20b784260a
8a70af6e0206606491a32b2ba3d6cc5065c31835a90bb9f33779622c8400b4a46c4a247d6aebf02235861353cb9ced948027706633944362bf6f82
3730dc7d9f6e5c2b52a88359fcd182dbd1d753009030a6488aa160d021c715e35065d542c0f6f62ef5fd1fd02d4f80ecef6af2abf695cbfdfae2f
e769034ef2849c9a92d010970a3739c0f37efb71afdf16abda4d
Exposant e: 3
```

Étape 2 : Créer le message cible

```
message = "PPTI SERVER ACCESS ON 2026-01-13 14:25 UTC"
hash_value = hashlib.sha256(message.encode()).digest()
```

Étape 3 : Appliquer le padding PKCS#1 v1.5

```
def pad_pkcs1_v1_5(hash_value: bytes, N_len: int) → int:
    # OID SHA-256
    hash_id = b"\x30\x31\x30\x0d\x06\x09\x60\x86\x48\x01\x65\x03\x04\x02\x01\x05\x00\x04\x20"

    # Calcul du padding
    padding_length = N_len - len(hash_value) - len(hash_id) - 3

    # Construction : 0x00 0x01 [0xFF...] 0x00 [SHA-256 OID] [Hash]
    FF = b"\xFF" * 10
    autre_bourrage = b"\xFF" * (padding_length - 10)
    padding = b"\x00\x01" + FF + b"\x00"

    padded_block = padding + hash_id + hash_value + autre_bourrage
    return int.from_bytes(padded_block, byteorder="big")
```

Structure du bloc PKCS#1 v1.5 généré :

```
00 01 [FFFFFF...] 00 [SHA256 OID] [HASH SHA-256] [PADDING COMPLÉMENTAIRE]
```

Étape 4 : Calculer la Racine Cubique

Étape 5 : Convertir en Format Hexadécimal

5

```

from Crypto.PublicKey import RSA
import hashlib

def load_public_key(pem_file: str):
    with open(pem_file, 'r') as f:
        public_key = RSA.import_key(f.read())
    return public_key.n, public_key.e

def sha256(message: str) → bytes:
    return hashlib.sha256(message.encode()).digest()

def pad_pkcs1_v1_5(hash_value: bytes, N_len: int) → int:
    hash_id = b"\x30\x31\x30\x0d\x06\x09\x60\x86\x48\x01\x65\x03\x04\x02\x01\x05\x00\x04\x20"
    padding_length = N_len - len(hash_value) - len(hash_id) - 3

    FF = b"\xFF" * 10
    autre_bourrage = b"\xFF" * (padding_length - 10)
    padding = b"\x00\x01" + FF + b"\x00"

    padded_block = padding + hash_id + hash_value + autre_bourrage
    return int.from_bytes(padded_block, byteorder="big")

def cube_root(x: int) → int:
    guess = x >> (x.bit_length() // 3)
    while True:
        next_guess = (2 * guess + x // (guess ** 2)) // 3
        if next_guess >= guess:
            return guess
        guess = next_guess

def sign(message: str, pem_file: str) → str:
    N, e = load_public_key(pem_file)
    N_len = N.bit_length() // 8
    hash_value = sha256(message)

    padded_block = pad_pkcs1_v1_5(hash_value, N_len)
    signature = cube_root(padded_block)

```

```
signature_hex = hex(signature)[2:].zfill(2 * N_len)
return signature_hex
```

```
# Utilisation
```

```
message = "PPTI SERVER ACCESS ON 2026-01-13 14:25 UTC"
signature = sign(message, "ppti_pubkey.pem")
print(f"Signature valide : {signature}")
```

4. Programmation du Badge

4.1 Localisation du Programmeur de Badge

Après avoir forgé la signature il nous faut un outil pour programmer notre badge qui se trouve dans le **bureau de l'administrateur** de la PPTI (même couloir 14-15/4).

Pour accéder au bureau :

```
>>> administrateur
Vous êtes dans le bureau d'un administrateur de la PPTI.
Il n'y a personne et le bureau n'est pas fermé à clef.

Ici se trouve un programmeur de badge.
```

```
>>> administrateur
Vous êtes dans le bureau d'un administrateur de la PPTI. Il n'y a personne
et le bureau n'est pas fermé à clef.

Ici se trouve un programmeur de badge.
Ici se trouve un plan du réseau.
```

4.2 Programmer le Badge

```
Supported commands: add, del
```

```
add : Add a signature to the service tag.
```

```
del : Remove a signature from the service tag.
```

```
Well, that was obvious. What else did you expect?
```

Le programmeur de badge demande les informations suivantes :

Paramètre	Valeur	Notes
ID	ppti_server_room	Identifiant de la signature
Message	"PPTI SERVER ACCESS ON 2026-01-18 11:26 UTC"	Doit correspondre à celui qui a été signé
Signature	[SIGNATURE_HEX_CALCULÉE]	Résultat de l'algorithme de forgerie

Étapes pour programmer le badge :

1. Lancer le programmeur de badge :

```
>>> programmer badge
Programmeur de Badge - Saisie des paramètres
```

1. Saisir l'ID :

Entrez l'ID de la signature : ppti_server_room

1. Saisir le message :

Entrez le message : PPTI SERVER ACCESS ON 2026-01-18 11:26 UTC

1. Saisir la signature forgée :

Entrez la signature (hex) : [COLLER LA SIGNATURE CALCULÉE]

[illegible]

4.3 Validation de la Programmation

Une fois le badge programmé, le système affichera :

[illegible]

5. Test d'Accès à la Salle Serveur

5.1 Utilisation du Badge

Avec le badge programmé, on retourne à la salle serveur et l'utiliser sur le lecteur de badge et l'accès authentifié, on peut maintenant accéder aux serveurs :

```
>>> sas
Vous refaites tout le tralala de la dernière fois.

Une lumière verte s'allume sur le lecteur de badge.

Les portes du sas s'écartent et vous pénétrez à l'intérieur.

Vous êtes à l'intérieur du couloir froid qui contient les serveurs. En fait,
les serveurs eux-mêmes sont dans des armoires verrouillées et vous ne pouvez
pas y accéder. Un des serveur est étiqueté "UGLIX M1". Juste en dessous il
y en a deux autres étiquetés "UGLIX M2, v1 (don't use, dangerous)" et "UGLIX
M2, v2 (safe)". Vous vous demandez à quoi tout ceci correspond. Enfin, il
y a une étagère avec divers objets.

>>> █
```

Massi-br