Cryptography 110

oalieno

2019/12/20

Table of Contents

Chapter I - Public Key Encryption

RSA

RSA 產生金鑰

- 選三個整數 p, q, e
- 計算
  - $n := pq$
  - $\varphi(n) = (p-1)(q-1)$
  - $d := e^{-1} \mod \varphi(n)$
- 公鑰是 (n, e) 私鑰是 (n, d)

碎碎念

- n, e 要滿足 $\gcd(e, \varphi(n)) = 1$，沒有滿足就重選
- 不然 e 在模 $\varphi(n)$ 下沒有模反元素，無法解密

RSA 加解密

- 明文 m 密文 c
- 加密 $c = m^e \mod n$
- 解密 $m = c^d \mod n$

RSA 正確性

目標

驗證 $m^{ed} \equiv m \pmod{n}$

拆解成小問題

- 分別驗證
  1. $m^{ed} \equiv m \pmod{p}$
  2. $m^{ed} \equiv m \pmod{q}$
- 再用中國剩餘定理拼起來得證 $m^{ed} \equiv m \pmod{n}$

RSA 正確性

推論一下

$$
\begin{aligned}
d &\equiv e^{-1} \pmod{\varphi(n)} \\
\Rightarrow ed &\equiv 1 \pmod{\varphi(n)} \\
\Rightarrow ed &= k\varphi(n) + 1 \text{ for some k} \\
&= k(p-1)(q-1) + 1
\end{aligned}
$$

RSA 正確性

驗證 $m^{ed} \equiv m \pmod{p}$

if gcd(m, p) = 1

$$\rightarrow m^{ed} = m^{k(p-1)(q-1)+1} = (m^{(p-1)})^{k(q-1)} m \equiv m \pmod{p}$$

if gcd(m, p) = p

$$\rightarrow m^{ed} \equiv 0 \equiv m \pmod{p}$$

驗證 $m^{ed} \equiv m \pmod{q}$

By the same argument

RSA 正確性

組起來

- 另 $x = m^{ed}$，現在已知

$$x \equiv m \pmod{p}$$
$$x \equiv m \pmod{q}$$

- 想求 x 模 n 會是多少
- 中國剩餘定理告訴我們在模 $n = pq$ 下存在唯一解
- x 模 n 等於 m 是一個解，那就只能是他了
- 所以 $m^{ed} \equiv m \pmod{n}$，得證

Relation to integer factorization

factor n $\longrightarrow$ obtain private key

- 如果我們可以分解 n
- 就可以順著原本的步驟產生私鑰，進而解密密文

Relation to integer factorization

obtain private key → factor n

- 如果我們有一個很有效率的演算法 f 能找到模 n 下的開方根，那我們就能分解 n

Relation to integer factorization

利用 f 分解 n

- 選一個 x，計算 $y \equiv x^2 \pmod{n}$
- 利用那個演算法 f 找出 y 在模 n 下的開方根 z
- y 的模開方根會有四個解，有 $\frac{1}{2}$ 的機率 z 不是 $\pm x$
- 如果 z 不是 $\pm x$
  - $z^2 \equiv x^2 \pmod{n} \Rightarrow (z+x)(z-x) \equiv 0 \pmod{n}$
  - $1 < \gcd(n, z+x) < n$ 或 $1 < \gcd(n, z-x) < n$ 會成立
  - 那就成功分解 n
- 如果 z 是 $\pm x$，就再選一次 x

Relation to integer factorization

有效率的演算法 f 找模 n 下得開方根

- 選一個 g，計算 $g^{ed-1} \equiv 1 \pmod{n}$
- $ed - 1 = k\varphi(n) = 2^t r$
- 這樣 $g^{2^t r} \equiv (g^{2^{t-1} r})^2 \pmod{n}$ 就會是一組模開方根
- $g^{2^t r} \equiv 1 \equiv (\pm 1)^2 \pmod{n}$，$\pm 1$ 已經是一組模開方根的解
- 如果 $g^{2^{t-1} r} \neq \pm 1$，就可以用剛剛講的方法分解 n
- $g^{2^t r}, g^{2^{t-1} r}, g^{2^{t-2} r}, \cdots g^r$ 都找不到就再選一次 g

Factoring Tools

- http://www.factordb.com/index.php
- https://github.com/DarkenCode/yafu

同態（Homomorphic）

同態的定義

- $f(x * y) = f(x) * f(y)$
- $*$ 可以是任意的一種運算元

RSA 的乘法

- RSA 的乘法有 homomorphic 的特性
- $E(m_1)E(m_2) = m_1^e m_2^e \mod n = (m_1 m_2)^e \mod n = E(m_1 m_2)$
- Leads to chosen ciphertext attack

RSA

What happens if you pick wrong primes p, q

How to pick large primes p, q

- $|p-q|$ 太小 $\rightarrow$ fermat factorization
- $p-1$ 的最大質因數很小 $\rightarrow$ Pollard's p - 1 Algorithm
- $p+1$ 的最大質因數很小 $\rightarrow$ Williams's p + 1 Algorithm
- $r-1$ 的最大質因數很小 $\rightarrow$ Cycling Attack
    - r 是 $p-1$ 的最大質因數

Strong Primes

- 能抵檔針對質因數小的攻擊的質數我們叫他 Strong Primes
- 那建議 p, q 一定要選 Strong Primes … 嗎?
- 其實隨機產生不會比 Strong Primes 還差 [1]

Strong Primes

- p - 1 has a large prime factor, denoted r (Pollard [2])
- p + 1 has a large prime factor (Williams [3])
- r - 1 has a large prime factor (Cycling Attack)

Pollard's p - 1 Algorithm

- $p-1$ 是一個 B-smooth 的數，也就是他最大的質因數是 B
- 如果 B 很小，就可以有效的分解 n

$$
\begin{aligned}
& p-1 \mid 1 \times 2 \times \cdots B \\
\Rightarrow\ & 2^{1\times 2\times\cdots B} = 2^{k(p-1)} \equiv 1 \pmod{p} \\
\Rightarrow\ & \gcd(2^{1\times 2\times\cdots B} - 1, n) > 1
\end{aligned}
$$

Pollard's p - 1 Algorithm

```
def pollard(n):
    a = 2
    b = 2
    while True:
        a = pow(a, b, n)
        d = gcd(a - 1, n)
        if 1 < d < n: return d
        b += 1
```

RSA

What happens if you pick wrong e

How to choose public exponent e

- e 太小 $\rightarrow$ direct eth root, broadcast attack
- e 太大 $\rightarrow$ 加密很慢
- 常見的 e 會選 $2^x + 1$ 這種形式的質數，例如 $2^{16} + 1 = 65537$，這樣在做 Square and Multiply 時只需要 $16 + 1$ 次運算

Square and Multiply

```
def SquareAndMultiply(x, y):
    if y == 0: return 1
    k = SquareAndMultiply(x, y // 2) ** 2
    return k * x if y % 2 else k
```

Direct eth Root

- 如果 m, e 都很小，使得 $m < n^{\frac{1}{e}} \Rightarrow m^e < n$
- 直接在整數下取 eth root 就可以還原 m
- 所以我們需要做 random padding

Broadcast Attack

- 用 e 個不同的 n 加密相同的 m，中國剩餘定理可以直接解回 m
- 以 e = 3 為例

$$\begin{cases} m^3 & \equiv c_1 \pmod{n_1} \\ m^3 & \equiv c_2 \pmod{n_2} \\ m^3 & \equiv c_3 \pmod{n_3} \end{cases}$$

Use CRT to get c, $m^3 \equiv c \pmod{n_1 n_2 n_3}$

$m^3 < n_1 n_2 n_3 \rightarrow m^3 = c \rightarrow$ direct eth root

RSA

What happens if you pick wrong d

How to choose private exponent d

- 基本上 d 也不是我們選的，是從 e 算出來的
- d 太小 $\rightarrow$ Wiener's attack, Boneh-Durfee's attack, $\cdots$

歷年來攻擊小 d 的演進

Bound for d	Assumed Interval for γ	Year	Citation
$d < \frac{1}{3}N^{\frac{1}{4}}$	No γ	1990	[4]
$d < \frac{1}{8}N^{\frac{3}{4}-\gamma}$	$0.25 \leq \gamma < 0.5$	2002	[5]
$d < N^{\frac{1-\gamma}{2}}$	$0.25 \leq \gamma < 0.5$	2008	[6]
$d < N^{\frac{3}{4}-\gamma}$	$0.25 \leq \gamma < 0.5$	2009	[7]
$d < \frac{\sqrt{6\sqrt{2}}}{6}N^{\frac{1}{4}}$	No γ	2013	[8]
$d < \frac{1}{2}N^{\frac{1}{4}}$	No γ	2015	[9]
$d < \frac{\sqrt{3}}{\sqrt{2}}N^{\frac{3}{4}-\gamma}$	$0.25 \leq \gamma < 0.5$	2019	[10]

Table: Comparison of the bounds on d for RSA modulus $N = pq$ [10]

Wiener Attack

Wiener Attack

- $ed = k\varphi(n) + 1$

$$d < \frac{1}{3} n^{\frac{1}{4}}$$
$$\Rightarrow \left| \frac{e}{n} - \frac{k}{d} \right| < \frac{1}{2d^2}$$
$$\Rightarrow \frac{k}{d} \text{ 會在 } \frac{e}{n} \text{ 的收斂連分數裡面}$$

- 遍歷所有 $\frac{e}{n}$ 的收斂連分數，其中一個會是 $\frac{k}{d}$

Wiener Attack

Wiener Attack (cont.)

- $\frac{ed-1}{k} = \varphi(n) = (p-1)(q-1) = n - p - \frac{n}{p} + 1$
- $p^2 + p(\frac{ed-1}{k} - n - 1) + n = 0$
- 求解一元二次方程式可得 p，再驗證 p 是否為 n 的因子即可

Wiener Attack - 什麼是收斂連分數

- $\dfrac{13}{17} = 0 + \dfrac{1}{1 + \frac{1}{3 + \frac{1}{4}}}$
- $\frac{13}{17}$ 的收斂連分數是 $[c_0, c_1, c_2, c_3]$

$$c_0 = 0 = \frac{0}{1}$$

$$c_1 = 0 + \frac{1}{1} = \frac{1}{1}$$

$$c_2 = 0 + \frac{1}{1 + \frac{1}{3}} = \frac{3}{4}$$

$$c_3 = 0 + \frac{1}{1 + \frac{1}{3 + \frac{1}{4}}} = \frac{13}{17}$$

Wiener Attack - 證明

假設 $d < \frac{1}{3}n^{\frac{1}{4}}$

$$
\begin{aligned}
\left|\frac{e}{n} - \frac{k}{d}\right| &= \left|\frac{ed - nk}{nd}\right| \\
&= \left|\frac{1 + k\varphi(n) - nk}{nd}\right| \\
&= \frac{k(n - \varphi(n)) - 1}{nd} < \frac{3k\sqrt{n} - 1}{nd} < \frac{3k\sqrt{n}}{nd} \\
&< \frac{1}{n^{\frac{1}{4}}d} < \frac{1}{2d^2}
\end{aligned}
$$

Wiener Attack - 證明

稍微解釋一下中間的代換

- 假設 $p \approx q \approx \sqrt{n}$

$$
\begin{aligned}
n - \varphi(n) &= n - (p-1)(q-1) \\
&= n - pq + p + q - 1 \\
&= p + q - 1 \\
&< 3\sqrt{n}
\end{aligned}
$$

Wiener Attack - 證明

稍微解釋一下中間的代換 (cont.)

$$
\begin{aligned}
k\varphi(n) &= ed - 1 < ed < \varphi(n)d \\
\Rightarrow & k < d < \frac{1}{3}n^{\frac{1}{4}}
\end{aligned}
$$

Wiener Attack - 證明

稍微解釋一下中間的代換 (cont.)

$$
\begin{aligned}
& d < \frac{1}{3} n^{\frac{1}{4}} \\
\Rightarrow & 2d < 3d < n^{\frac{1}{4}} \\
\Rightarrow & \frac{1}{2d} > \frac{1}{n^{\frac{1}{4}}}
\end{aligned}
$$

Wiener Attack - 證明

Legendre's theorem in Diophantine approximations

給定 $\alpha \in \mathbb{R}, \frac{a}{b} \in \mathbb{Q}$，並且滿足 $\left|\alpha - \frac{a}{b}\right| < \frac{1}{2b^2}$
那麼 $\frac{a}{b}$ 會是 α 的收斂連分數

根據這個定理

- 剛剛已經推得 $\left|\frac{e}{n} - \frac{k}{d}\right| < \frac{1}{2d^2}$
- 對應上述定理 $\alpha = \frac{e}{n}, a = k, b = d$
- 所以 $\frac{k}{d}$ 會是 $\frac{e}{n}$ 的收斂連分數，得證

Wiener Attack - 時間複雜度

- 計算收斂連分數時，是在做輾轉相除法，需要 $O(\log(e))$
- 解一元二次方程式只需要 $O(1)$
- 時間複雜度: $O(\log(e))$

RSA

Chosen Ciphertext Attack

Chosen Ciphertext Attack

情境

- 有一個 oracle 給他密文可以得到明文
- 但是唯獨不能解某個特定的密文 c

解法

- 使用 oracle 解 $2^e c$ 得 $2m$
- $2^{-1} \cdot 2m \equiv m \pmod{n}$

LSB Oracle Attack

情境

- Least Significant Bit Oracle Attack
- 有一個 oracle 給他密文可以得到明文的最低位那個 bit
- 類似 Chosen Ciphertext Attack 但是只能拿 1 bit

LSB Oracle Attack - 解法一

Oracle

$$2^e c \xrightarrow{\text{oracle}} 2m$$

推論

- $$\begin{cases} \lfloor \lfloor 2m \rfloor_n \rfloor_2 = \lfloor 2m \rfloor_2 = 0, & \text{if } m \in [0, \frac{n}{2}) \\ \lfloor \lfloor 2m \rfloor_n \rfloor_2 = \lfloor (2m - n) \rfloor_2 = 1, & \text{if } m \in [\frac{n}{2}, n) \end{cases}$$
- 根據最低位那個 bit 是 0 或 1 推論 m 在 $\frac{n}{2}$ 之前或之後

LSB Oracle Attack - 解法一

Oracle

$$4^e c \xrightarrow{\text{oracle}} 4m$$

推論

- 如果 $m \in [0, \frac{n}{2})$
- $$\begin{cases} \lfloor\lfloor 4m \rfloor_n\rfloor_2 = \lfloor 4m \rfloor_2 = 0, & \text{if } m \in [0, \frac{n}{4}) \\ \lfloor\lfloor 4m \rfloor_n\rfloor_2 = \lfloor (4m - n) \rfloor_2 = 1, & \text{if } m \in [\frac{n}{4}, \frac{2n}{4}) \end{cases}$$
- 根據最低位那個 bit 是 0 或 1 推論 m 在 $\frac{n}{4}$ 之前或之後
- 如果 $m \in [\frac{n}{2}, n)$，同理

LSB Oracle Attack - 解法一

碎碎念

- 每次可以縮小一半的範圍，需要 $O(\log(n))$ 次 oracle
- 有點像是在二分搜尋

LSB Oracle Attack - 解法二

Oracle

$$c \xrightarrow{\text{oracle}} m$$

推論

$$
\begin{aligned}
m &= x_0 + 2y_1 \\
r &\equiv \lfloor x_0 + 2y_1 \rfloor_n \pmod 2 \\
&\equiv x_0 \pmod 2 \\
\Rightarrow x_0 &\equiv r \pmod 2
\end{aligned}
$$

LSB Oracle Attack - 解法二

Oracle

$$(2^{-1})^e c \xrightarrow{\text{oracle}} 2^{-1} m$$

推論

$$
\begin{aligned}
2^{-1} m &= 2^{-1} x_0 + x_1 + 2 y_2 \\
r &\equiv \lfloor 2^{-1} x_0 + x_1 + 2 y_2 \rfloor_n \pmod 2 \\
&\equiv \lfloor 2^{-1} x_0 \rfloor_n + x_1 \pmod 2 \\
\Rightarrow x_1 &\equiv r - \lfloor 2^{-1} x_0 \rfloor_n \pmod 2
\end{aligned}
$$

LSB Oracle Attack - 解法二

Oracle

$$(2^{-2})^e c \xrightarrow{\text{oracle}} 2^{-2} m$$

推論

y_3	x_2	x_1	x_0

$$2^{-2} m = 2^{-2} x_0 + 2^{-1} x_1 + x_2 + 2 y_3$$

$$\begin{aligned} r &\equiv \lfloor 2^{-2} x_0 + 2^{-1} x_1 + x_2 + 2 y_3 \rfloor_n \pmod 2 \\ &\equiv \lfloor 2^{-2} x_0 + 2^{-1} x_1 \rfloor_n + x_2 \pmod 2 \end{aligned}$$

$$\Rightarrow x_2 \equiv r - \lfloor 2^{-2} x_0 + 2^{-1} x_1 \rfloor_n \pmod 2$$

LSB Oracle Attack - 解法二

碎碎念

- x_i 代表 m 的第 i 個 bit
- y_i 代表 m 整段從最高位的 bit 到最低位數來第 i 個 bit
- 每次可以推論一個 bit，需要 $O(\log(n))$ 次 oracle

LSB Oracle Attack - CTF

CTF 考古題

- Google CTF QUALS 2018 - PERFECT-SECRECY
- TokyoWesterns CTF 4th 2018 - mixed-cipher
- HITCON CTF 2018 - Lost-Key

Bleichenbacher 1998 (BB98)

- 介紹一個 Real World 的例子 Bleichenbacher 1998 年的論文
- 也是一種 Chosen Ciphertext Attack，這次的 oracle 是給他密文可以得到明文的最高位那個 byte 是不是 0x0002
- 這個情境很像 Padding Oracle Attack，會告訴你給他的格式正不正確
- 這個格式是 PKCS#1 v1.5

Figure: PKCS #1 v1.5

BB98 本人

Bleichenbacher 1998

Oracle

$$s^e c \xrightarrow{\text{oracle}} sm \text{ 格式對不對}$$

推論

00	02	...	≤ sm % n <	00	03	...

- 他檢查格式的方法是看前面兩個 bytes 是不是 0002
- 所以如果他符合格式的話，代表 $2B \leq \text{sm \% n} < 3B$

Bleichenbacher 1998

推論 (cont.)

- $sm = kn + (sm\%n)$ for some unknown k

$$
\begin{aligned}
& 2B \le sm\%n < 3B \\
\Rightarrow\ & 2B \le sm - kn < 3B \\
\Rightarrow\ & \frac{2B + kn}{s} \le m < \frac{3B + kn}{s}
\end{aligned}
$$

- 雖然我們不知道 k，但是我們可以考慮所有可能的 k

Bleichenbacher 1998

推論 (cont.)

- 要考慮所有可能的 k，那就要看一下 k 的範圍

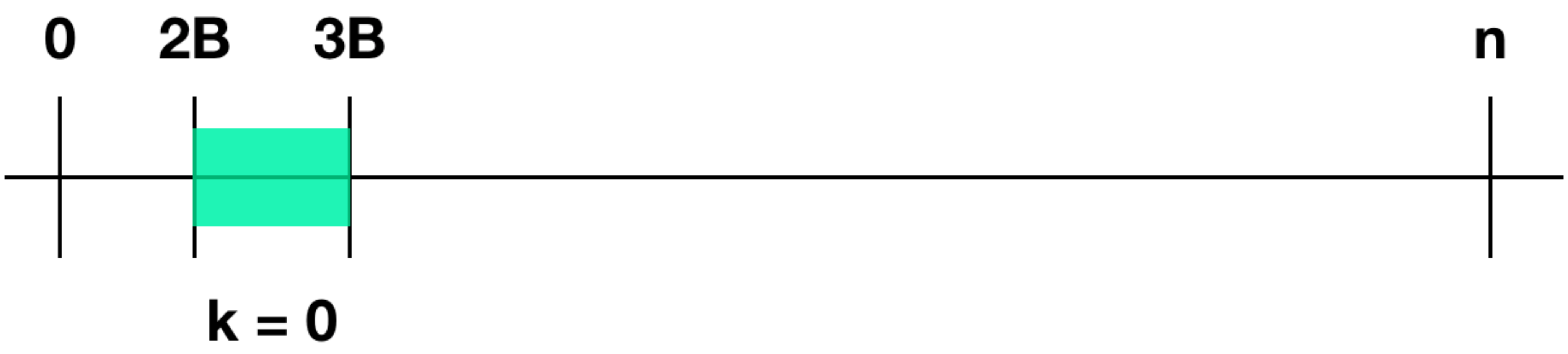
$$2B \leq sm - kn < 3B$$
$$\Rightarrow \frac{sm - 3B}{n} < k \leq \frac{sm - 2B}{n}$$

- 我們知道原本的明文會滿足 $0 \leq m < n$
- 所以就可以用舊的 m 的範圍推論 k 的範圍
- 再用 k 的範圍推論新的 m 的範圍

Bleichenbacher 1998

假設 $s = 1$ 格式正確

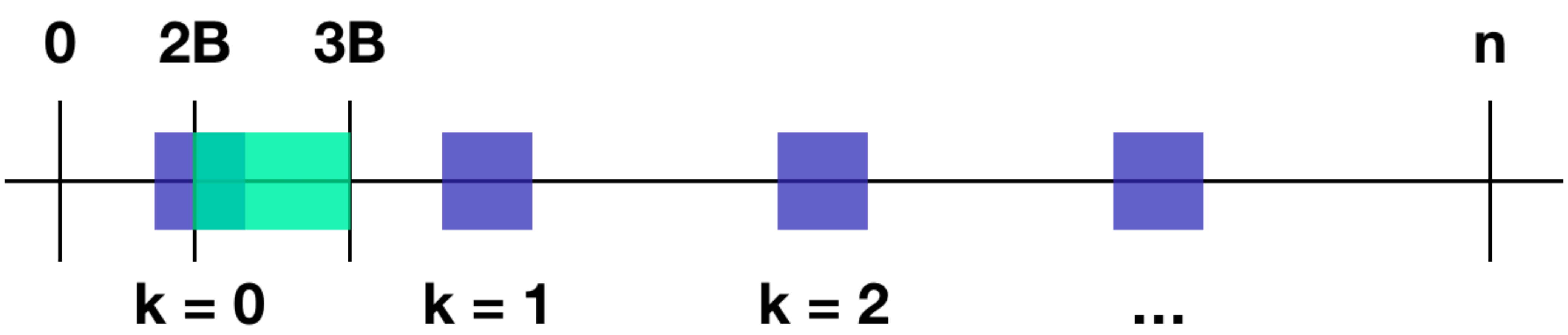
- k 只可能是 0
- 新的 m 的範圍 $2B \leq m < 3B$



Bleichenbacher 1998

假設 $s = 10$ 格式正確

- 綠色是 m 舊的範圍，藍色這次 oracle 得到的新範圍
- 兩個交集的地方就是 m 新的可能的範圍



Bleichenbacher 1998

總結一下

- 每次 oracle 會把這些範圍和舊的範圍交集起來
- m 的範圍就會越來越小，直到只剩一個

Bleichenbacher 1998

後續影響

- 2016 : DROWN: Breaking TLS Using SSLv2 [11]
- 2018 : Return Of Bleichenbacher Oracle Threat (ROBOT) [12]
- 2019 : The 9 Lives of Bleichenbacher's CAT [13]
- 到近幾年還是會看到他的身影

RSA

Coppersmith Method

Coppersmith Method - 問題敘述

Given

- A monic polynomial $f(x) = x^\delta + \cdots$
- An integer N of unknown factorization
- An upper bound X

Goal

- Find all $|x_0| \leq X$ satisfy $f(x_0) \equiv 0 \pmod{b}$
- Where b is a divisor of N, and $b \geq N^\beta$

碎碎念

- 這個演算法就是 find small modular root

Coppersmith Method - 原理

換個問題解

- Let $X_0 = \{ x_0 \mid f(x_0) \equiv 0 \pmod{b},\ |x_0| \leq X \}$
- Find a function g such that $\forall x_0 \in X_0, g(x_0) = 0$

碎碎念

- 計算 mod b 下的根很難，就算根的範圍縮小了，還是不知道要怎麼做啊
- 如果我們能找到一個 $g(x)$，他在整數底下求出來的根和 $f(x)$ 在 mod b 下求出來的根一樣就好，在整數底下求根我會做

Coppersmith Method - 原理

$g(x)$ 從哪裡來

- Pick two integers m, t
- Construct a collection of polynomial from $f(x)$

$$f_i(x) = \begin{cases} g_{i,j}(x) = x^j N^{m-i} f^i(x) & \text{for } 0 \le i < m,\ 0 \le j < \delta \\ h_i(x) = x^i f^m(x) & \text{for } 0 \le i < t \end{cases}$$

- Satisfy $\forall x_0 \in X_0,\ g_{i,j}(x_0) \equiv h_i(x_0) \equiv 0 \pmod{b^m}$

碎碎念

- 生成一堆滿足 mod $b^m = 0$ 的變種 f

Coppersmith Method - 原理

嘗試製造 $g(x)$

- Construct an integer linear combination $g(x)$

$$g(x) = \sum_{i=0}^{n-1} a_i f_i(x), \text{ for } a_i \in \mathbb{Z}$$

Where $n = m\delta + t$

碎碎念

- 把變種 f 看成 coefficent vector，考慮變種 f 生成的 lattice

Coppersmith Method - 原理

如果 $|g(x_0)| < b^m$ 的話

- 我們構造的 $f_i(x)$ 都滿足 $f_i(x_0) \equiv 0 \pmod{b^m}$，$g(x)$ 也滿足
- $\forall x_0 \in X_0$

$$\begin{cases} g(x_0) \equiv 0 \pmod{b^m} \\ |g(x_0)| < b^m \end{cases} \Rightarrow g(x_0) = 0$$

- 這樣我們找 $g(x)$ 上所有根，裡面就會包含我們要的答案 X_0

Coppersmith Method - 原理

再換個問題解

- $\forall |x_0| < X,\ \|g(xX)\| < \frac{b^m}{\sqrt{n}} \Rightarrow |g(x_0)| < b^m$

碎碎念

- 這個小定理幫我們把問題轉成在 lattice 找 short vector
- 覺得很熟悉嗎，登愣，Shortest Vector Problem，那我們就用 LLL 演算法來做搞定了
- 這裡的 $\|g(xX)\|$ 是指 $g(xX)$ 的 coefficient vector 的 norm

Coppersmith Method - 原理

Proof of the previous statement

- Let c_i be the coefficients of $g(x)$, then $\forall |x_0| < X$

$$\begin{aligned}
|g(x_0)| &= \sum_{i=0}^{n} c_i x_0^i \leq \sum_{i=0}^{n} |c_i x_0^i| \\
&\leq \sum_{i=0}^{n} |c_i X^i| = \sqrt{(\sum_{i=0}^{n} |c_i X^i|)^2} \leq \sqrt{n \sum_{i=0}^{n} |c_i X^i|^2} \\
&= \sqrt{n} \|g(xX)\| < b^m
\end{aligned}$$

碎碎念

- 中間用到了柯西不等式，忘記的同學們可以回去複習一下

Coppersmith Method - 原理

LLL 找到的 short vector 夠嗎?

- Let L be the coefficient matrix formed by $f_0(x), f_1(x), \cdots$, and v be the vector found using LLL
- 現在要證明 v 滿足 $\|g(xX)\| < \frac{b^m}{\sqrt{n}}$
- 已知 v 會滿足 $\|v\| \leq 2^{\frac{n-1}{4}}\det(L)^{\frac{1}{n}}$
- 所以我們只要讓 $2^{\frac{n-1}{4}}\det(L)^{\frac{1}{n}} < \frac{b^m}{\sqrt{n}} < \frac{N^{\beta m}}{\sqrt{n}}$ 就可以了

Coppersmith Method - 原理

Compute $\det(L)$

- Notice that the degree of $f_i(x)$ is i, therefore the basis of L forms a lower triangular matrix
- So the $\det(L)$ is simply the product of all entries on the diagonal

$$\det(L) = N^{\frac{1}{2}\delta m(m+1)} X^{\frac{1}{2}n(n-1)}$$

Coppersmith Method - 原理

決定 m, t, X

- 我們其實還沒決定 m, t, X 要是多少
- 基本上我們想要 m, t 越小越好，這樣 lattice 會小一點
- X 則是越大越好，這樣我們能解的東西越多
- 而且最好可以消掉一些變數，比較好化簡
- $m = \left\lceil \frac{\beta^2}{\delta\varepsilon} \right\rceil, t = \left\lfloor \delta m (\frac{1}{\beta} - 1) \right\rfloor, X = \left\lceil \frac{1}{2} N^{\frac{\beta^2}{\delta} - \varepsilon} \right\rceil$
- 這裡引入了一個新的常數 ε，滿足 $0 < \varepsilon \leq \frac{1}{7}\beta$，可以調參

Coppersmith Method - 原理

最後的最後

- 最後就剩驗證 $2^{\frac{n-1}{4}}\det(L)^{\frac{1}{n}} < \frac{N^{\beta m}}{\sqrt{n}}$
- 把 $m, t, X, \det(L)$ 代進去喇一喇就行了

Coppersmith Method - 總結

Given

- A monic polynomial $f(x) = x^\delta + \cdots$
- An integer N of unknown factorization
- A rational number β

Coppersmith Method

- Find all $|x_0| \leq \frac{1}{2}N^{\frac{\beta^2}{\delta}-\varepsilon}$ satisfy $f(x_0) \equiv 0 \pmod{b}$
- Where b is a divisor of N, and $b \geq N^\beta$

Coppersmith Method - 總結

Push the bound

- 我們可以很輕易的把 X 推廣到 $cN^{\frac{\beta^2}{\delta}}$ for some constant c
- 選 $\varepsilon = \frac{1}{\log N}$ 那 $X = \frac{1}{4}N^{\frac{\beta^2}{\delta}}$
- 接著把 $[-cN^{\frac{\beta^2}{\delta}}, cN^{\frac{\beta^2}{\delta}}]$ 切成 4c 個區間
- 假設 x_i 是每個區間的中點，那把所有 $f(x - x_i)$ 的根收集起來就是答案了
- 可以平行去跑所有區間

Coppersmith Method - 總結

Given

- A monic polynomial $f(x) = x^{\delta} + \cdots$
- An integer N of unknown factorization
- A rational number β

Coppersmith Method

- Find all $|x_0| \leq cN^{\frac{\beta^2}{\delta}}$ satisfy $f(x_0) \equiv 0 \pmod{b}$
- Where b is a divisor of N, and $b \geq N^{\beta}$

Coppersmith Method - 演算法步驟

Step 1 - 選參數

- 選 $m = \left\lceil \frac{\beta^2}{\delta\epsilon} \right\rceil, t = \left\lfloor \delta m(\frac{1}{\beta} - 1) \right\rfloor$，計算 $X = \left\lceil N^{\frac{\beta^2}{\delta} - \epsilon} \right\rceil$

Step 2 - 計算變種 f

- 計算

$$
\begin{aligned}
g_{i,j}(x) &= x^j N^{m-i} f^i(x) && \text{for } 0 \le i < m,\ 0 \le j < \delta \\
h_i(x) &= x^i f^m(x) && \text{for } 0 \le i < t
\end{aligned}
$$

Coppersmith Method - 演算法步驟

Step 3 - LLL

- $g_{i,j}(xX), h_i(xX)$ 的 coefficient vetors 組成的 lattice basis B
- 對 B 做 LLL algorithm

Step 4 - 還原 g

- 假設 v 是 shortest vector in LLL reduced basis
- 那 v 是某個 $g(xX)$ 的 coefficient vector，從 v 還原 $g(x)$

Coppersmith Method - 演算法步驟

Step 5 - 找根

- 找出 $g(x)$ 的所有根
- 檢查根 x_0 是否滿足 $\gcd(N, f(x_0)) \geq N^\beta$，沒有滿足的丟掉

Stereotyped messages

- 已知大部分的訊息，比如某個固定格式
- 明文 $m = \tilde{m} + x_0$，已知 $\tilde{m}$，未知 x_0 滿足 $x \leq N^{\frac{1}{e}}$
- 密文 $c \equiv m^e \equiv (\tilde{m} + x_0)^e \pmod{N}$
- x_0 會是 $f(x) \equiv (\tilde{m} + x)^e - c \pmod{N}$ 的小根

Coppersmith Method

- 參數設定為 $\beta = 1, \delta = e, c = 1, X = N^{\frac{1}{e}}$
- 用 Coppersmith Method 可以找回 x_0，再計算出明文 m

Known High Bits of p

- $N = pq$，質數 $p = \tilde{p} + x_0$，已知 $\tilde{p}$，未知 x_0 滿足 $|x_0| < N^{\frac{1}{4}}$
- x_0 會是 $f(x) = \tilde{p} + x \pmod{p}$ 的小根

Coppersmith Method

- 參數設定為 $\beta = \frac{1}{2}, \delta = 1, c = 1, X = N^{\frac{1}{4}}$
- 用 Coppersmith Method 可以找回 x_0，再計算出質數 p

Coppersmith Method

Real World Example

- 2013 : Factoring RSA Keys from Certified Smart Cards [14]
- 2017 : The Return of Coppersmith's Attack (ROCA) [15]

Elliptic Curves over Finite Fields

- Elliptic Curves 就是 degree 最多是 3 的方程式們
- 我們用 $E(K)$ 來表示 equation E over field K 所形成的 group
- 大多數密碼學的應用是用 Elliptic Curves over Finite Fields

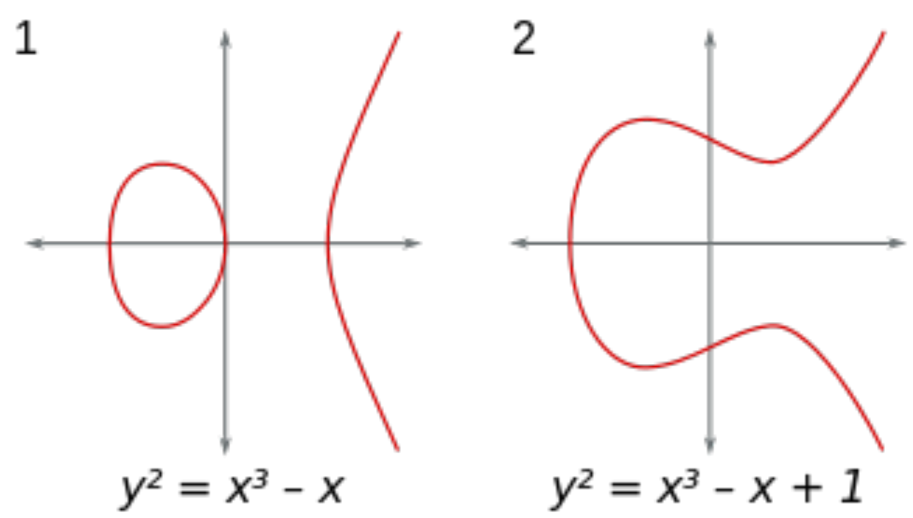


Figure: Elliptic Curves over Real Numbers [16]

Elliptic Curves Cryptography

- 我們以最常看到的 $y^3 = x^2 + ax + b$ 這個形式的 equation 做例子
- 考慮這個方程式所有解的集合 $\{(x, y) | y^3 = x^2 + ax + b\}$
- 接著我們只要再定義一個加法就可以得到一個 group 了
  - 當然還需要驗證他是否滿足群的性質

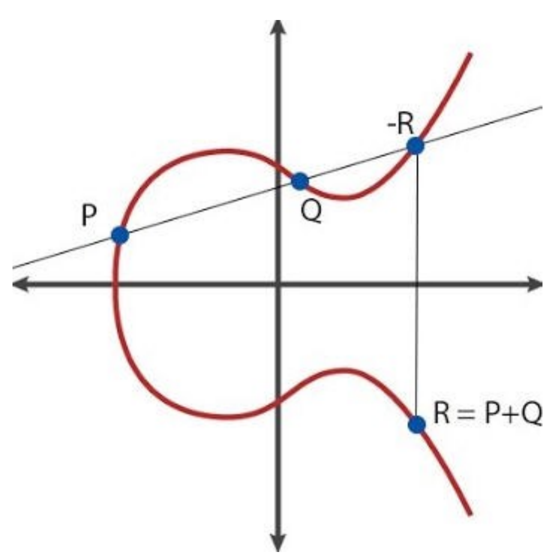


Figure: Addition on Elliptic Curves

Elliptic Curves Discrete Logarithm Problem (ECDLP)

- 給一個在 Elliptic Curve 上的點 P
- 給 $sP = \underbrace{P + P + \cdots + P}_{\text{s times}}$，求 s
- 這是很難的問題，沒有辦法有效率的解決
- 其實就是把原本 Discrete Logarithm Problem 的 group 換成 Elliptic Curves 上的 group

Elliptic Curves Cryptography

- 把之前用到 DLP 的密碼系統都改成 ECDLP
- Diffie-Hellman $\rightarrow$ Elliptic Curves Diffie-Hellman
- ElGamal $\rightarrow$ ECElGamal
- DSA $\rightarrow$ ECDSA
- ...
- 就得到一堆新的系統啦

Elliptic Curve

General Attack

General Attack on ECDLP

- n 是 $E(K)$ 的 order，l 是 n 最大的質因數

Attack	Expected Running Time
Exhaustive Search	$O(n)$
Baby-Step, Giant-Step	$O(\sqrt{n})$
Pollard's	$O(\frac{\sqrt{\pi n}}{2})$
Pohlig Hellman	$O(\sqrt{l})$

Chapter II - Digital Signature

RSA

How RSA Digital Signature works

- RSA 也可以用來做數位簽章
- 明文 m
- 簽章 $s = m^d \mod n$
- 驗證 $v = s^e \mod n$ 是否等於 m
- 因為 $m < n$，所以實務上會先做 hash 然後 padding

RSA

Signature Forgery

Random Signature Forgery

- RSA 乘法的同態性質讓我們能夠偽造隨機的簽章
- 給 (m, s) 可以偽造出 (m^k, s^k) for some k
- $s^k = (m^d)^k = (m^k)^d$ 所以簽章是合法的
- 但沒辦法控制偽造出什麼明文
- 只對 textbook RSA 有用，有 hash 有 padding 就沒用了

Bleichenbacher 2006 (BB06)

- 介紹一個 Real World 的例子 Bleichenbacher 2006 年的論文
- 針對 PKCS#1 v1.5 (RFC 2313) 格式的 Signature Forgery
- 後續有一系列的衍生的攻擊
  - 2016：RSA Signature Forgery in python-rsa [17]
  - 2019：A Decade After Bleichenbacher '06, RSA Signature Forgery Still Works [18]

BB06 本人

PKCS

PKCS

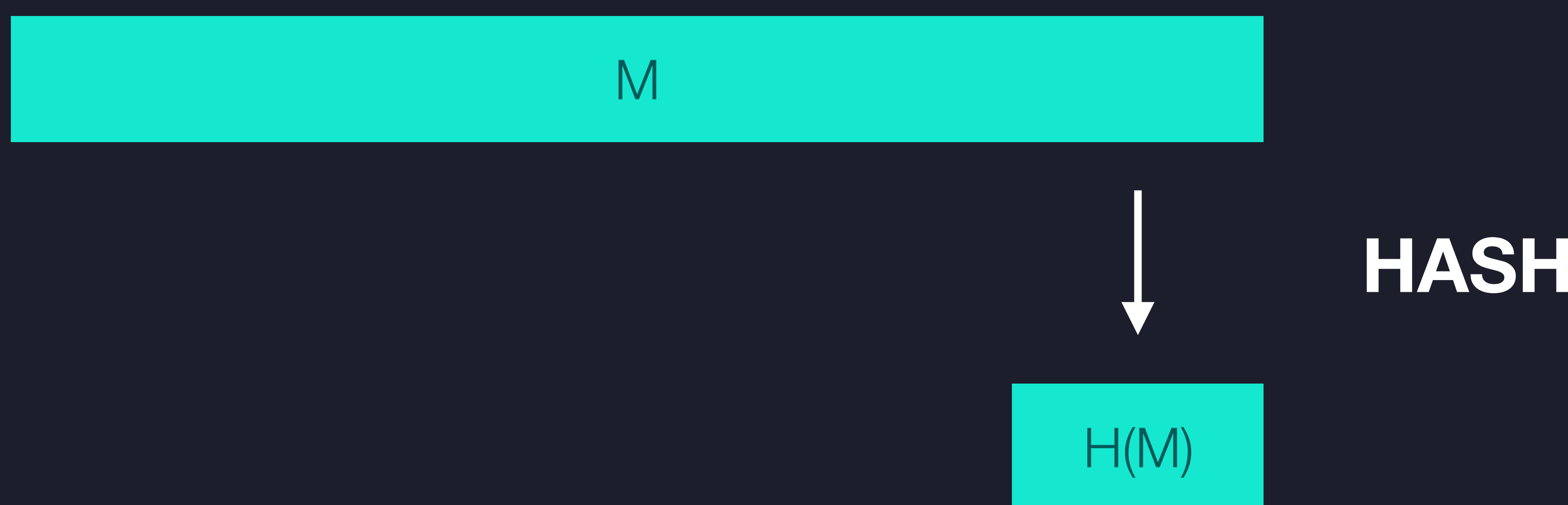
- PKCS (Public Key Cryptography Standards) 是公鑰密碼標準
- 制定了一系列從 PKCS#1 到 PKCS#15 的標準
- 其中 PKCS#1 是 RSA Cryptography Standard

ASN.1

- ASN.1 是高階的抽象標準
- 具體的實作編碼規則有：BER, CER, DER, PER, XER

PKCS#1 1.5 Signature

https://tools.ietf.org/html/rfc2313



Sign

Step 1 : Message Digest

PKCS#1 1.5 Signature

https://tools.ietf.org/html/rfc2313

- **ASN.1 是編碼數據的格式，這裡紀錄了使用的 hash 演算法**

D	=	00	01	FF	…	FF	00	ASN.1	H(M)

padding

Sign

Step 2 : Data Encoding

PKCS#1 1.5 Signature

https://tools.ietf.org/html/rfc2313

$$\boxed{D}^{\,d} \; \% \; n = \boxed{S}$$

Sign

Step 3 : RSA encryption

PKCS#1 1.5 Signature

https://tools.ietf.org/html/rfc2313

$$S^{e} \% n = D$$

Verify

Step 1 : RSA decryption

PKCS#1 1.5 Signature

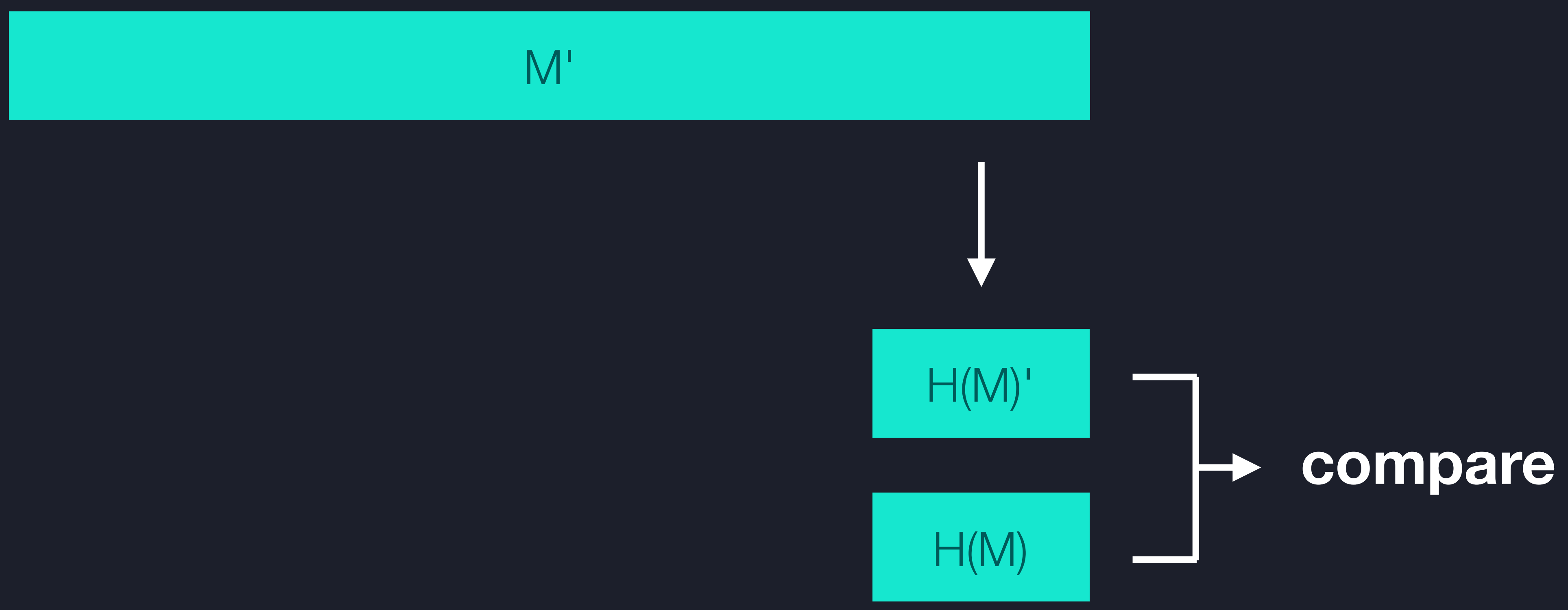
https://tools.ietf.org/html/rfc2313

- 需要 parse 這個格式取出 H(M)
- 這個標準沒有說要怎麼 parse
- 如果 e 太小且沒有正確的 parse，就有機會偽造簽章

D	=	00	01	FF	...	FF	00	ASN.1	H(M)

PKCS#1 1.5 Signature

https://tools.ietf.org/html/rfc2313



Verify

Step 3 : Message digesting and comparison

Bleichenbacher RSA Signature Forgery (2006)

Bleichenbacher RSA Signature Forgery (2006)

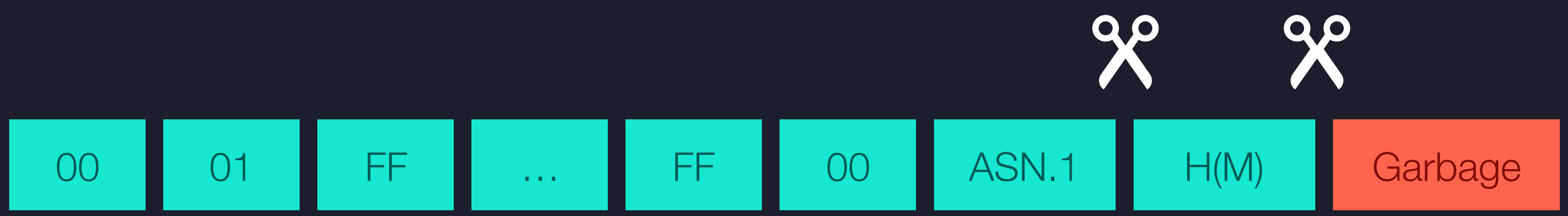
https://mailarchive.ietf.org/arch/msg/openpgp/5rnE9ZRN1AokBVj3VqblGlP63QE

- 又稱作 BB06
- 針對 PKCS#1 1.5 (RFC 2313)
- RSA 簽章偽造

Bleichenbacher RSA Signature Forgery (2006)

https://mailarchive.ietf.org/arch/msg/openpgp/5rnE9ZRN1AokBVj3VqbIGIP63QE

- 實作缺陷：可以有多餘的字元在後面
- parse 的時候直接取出後面固定長度的 H(M)
- 沒有檢查後面還有沒有東西



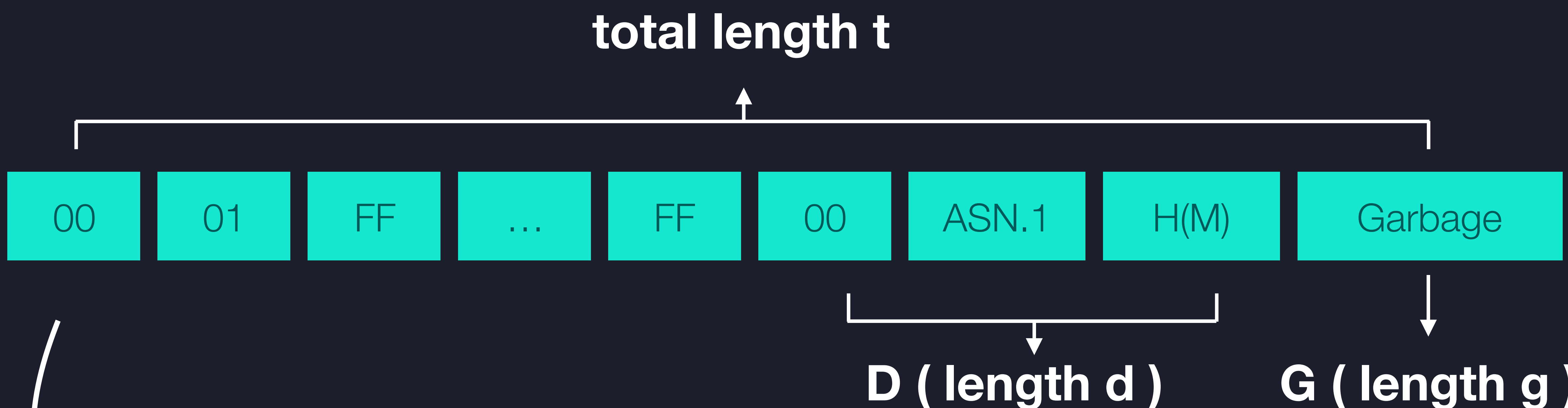
Bleichenbacher RSA Signature Forgery (2006)

https://mailarchive.ietf.org/arch/msg/openpgp/5rnE9ZRN1AokBVj3VqblGlP63QE

- 在 e = 3 的情況下可以 forge signature
- 嘗試構造 ED 讓 ED 的三次方不超過 n 且滿足以下格式

Bleichenbacher RSA Signature Forgery (2006)

https://mailarchive.ietf.org/arch/msg/openpgp/5rnE9ZRN1AokBVj3VqbIGIP63QE



$$\boxed{2^{t-15}} \quad \boxed{-2^{d+g} + 2^{g} \cdot D} \quad \boxed{+\, G}$$

$$(x+y)^3 = \boxed{x^3} \quad \boxed{+\, 3x^2y} \quad \boxed{+\, 3xy^2 + y^3}$$

Bleichenbacher RSA Signature Forgery (2006)

https://mailarchive.ietf.org/arch/msg/openpgp/5rnE9ZRN1AokBVj3VqblGlP63QE

$$x = 2^{\frac{t-15}{3}}$$

$$y = \frac{(D - 2^d) \cdot 2^g}{3 \cdot 2^{\frac{2(t-15)}{3}}}$$

Bleichenbacher RSA Signature Forgery (2006)

https://mailarchive.ietf.org/arch/msg/openpgp/5rnE9ZRN1AokBVj3VqblGlP63QE

- 假設
  - Key 長度為 3072 bit
  - Garbage 長度為 2072 bit
  - 使用 SHA-1 的話，D 的長度是 288 bit
- 最後 ED = x + y 就是我們構造出的合法簽章

$$x = 2^{1019}$$

$$y = \frac{(D - 2^{288}) \cdot 2^{34}}{3}$$

RSA Signature Forgery in python-rsa (2016)

CVE-2016-1494

RSA Signature Forgery in python-rsa (2016)

https://blog.filippo.io/bleichenbacher-06-signature-forgery-in-python-rsa/

- 實作缺陷： padding bytes 可以是任意字元

```
...
# If we can't find the signature  marker, verification failed.
if clearsig[0:2] != b('\x00\x01'):
    raise VerificationError('Verification failed')

# Find the 00 separator between the padding and the payload
try:
    sep_idx = clearsig.index(b('\x00'), 2)
except ValueError:
    raise VerificationError('Verification failed')

# Get the hash and the hash method
(method_name, signature_hash) = _find_method_hash(clearsig[sep_idx+1:])
message_hash = _hash(message, method_name)
...
```

直接取第二個 0x00 沒有檢查中間的 padding bytes

RSA Signature Forgery in python-rsa (2016)

https://blog.filippo.io/bleichenbacher-06-signature-forgery-in-python-rsa/

- 在 e = 3 的情況下可以 forge signature
- 嘗試構造 ED 讓 ED 的三次方不超過 n 且滿足以下格式
  - ED^3 的後綴是 ASN.1 + H(M)
  - ED^3 的前綴是 \x00\x01

S^3 % n

=	00	01	??	...	??	00	ASN.1	H(M)

RSA Signature Forgery in python-rsa (2016)

https://blog.filippo.io/bleichenbacher-06-signature-forgery-in-python-rsa/

S	0	0	0	1
S^3	0	0	0	1
目標	1	1	0	1

match

RSA Signature Forgery in python-rsa (2016)

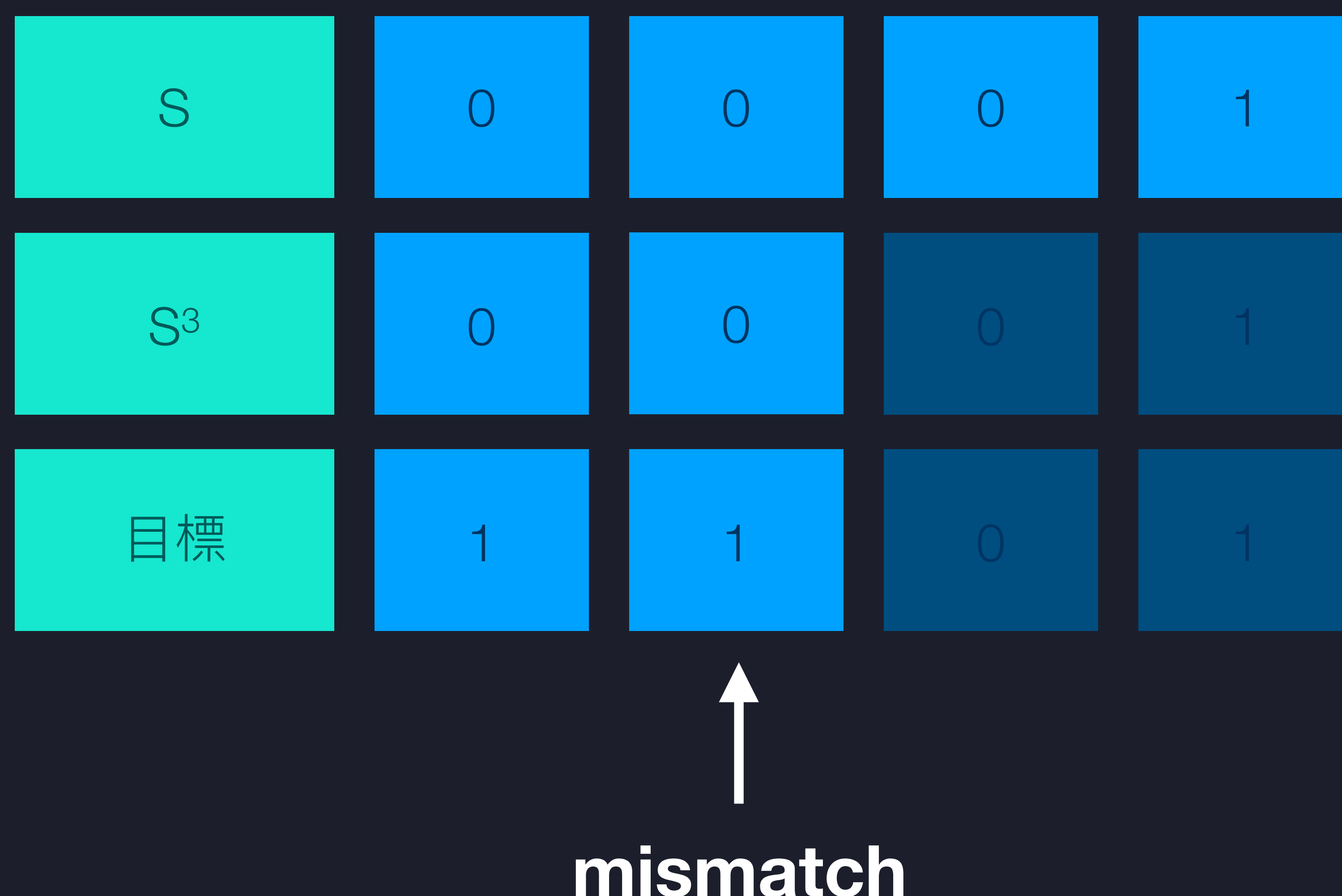
https://blog.filippo.io/bleichenbacher-06-signature-forgery-in-python-rsa/

S	0	0	0	1
S^3	0	0	0	1
目標	1	1	0	1

match

RSA Signature Forgery in python-rsa (2016)

https://blog.filippo.io/bleichenbacher-06-signature-forgery-in-python-rsa/



RSA Signature Forgery in python-rsa (2016)

https://blog.filippo.io/bleichenbacher-06-signature-forgery-in-python-rsa/

S	0	1	0	1
S^3	1	1	0	1
目標	1	1	0	1

match

RSA Signature Forgery in python-rsa (2016)

https://blog.filippo.io/bleichenbacher-06-signature-forgery-in-python-rsa/

RSA Signature Forgery in python-rsa (2016)

https://blog.filippo.io/bleichenbacher-06-signature-forgery-in-python-rsa/

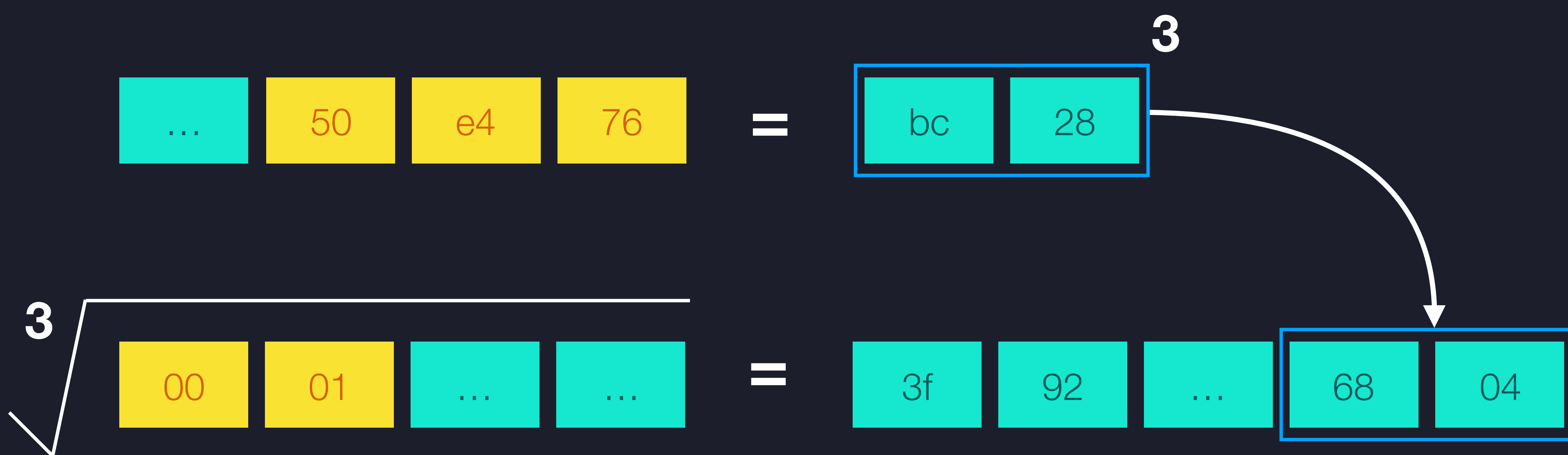
S	0	1	0	1
S^3	1	1	0	1
目標	1	1	0	1

$$0101^3 = 1111101$$

RSA Signature Forgery in python-rsa (2016)

https://blog.filippo.io/bleichenbacher-06-signature-forgery-in-python-rsa/

- 要讓 ED^3 的前綴是 \x00\x01 只要把 \x00\x01... 開三次方
- 最後再把開完三次方的值的後綴換成前面算出來的後綴
- 就可以成功自己構造合法簽章了



RSA Signature Forgery in python-rsa (2016)

https://blog.filippo.io/bleichenbacher-06-signature-forgery-in-python-rsa/

A Decade After Bleichenbacher '06, RSA Signature Forgery Still Works (2019)

A Decade After Bleichenbacher '06, RSA Signature Forgery Still Works (2019)

https://i.blackhat.com/USA-19/Wednesday/us-19-Chau-A-Decade-After-Bleichenbacher-06-RSA-Signature-Forgery-Still-Works.pdf

- 整個格式固定是 n 這麼長
- 用 Symbolic Execution 去找到可以任意亂塞的部分有多長

Name - Version	***Overly lenient***	*Practical exploit under small e*
axTLS - 2.1.3	YES	YES
BearSSL - 0.4	No	-
BoringSSL – 3112	No	-
Dropbear SSH – 2017.75	No	-
GnuTLS – 3.5.12	No	-
LibreSSL – 2.5.4	No	-
libtomcrypt – 1.16	YES	YES
MatrixSSL – 3.9.1 (Certificate)	YES	No
MatrixSSL – 3.9.1 (CRL)	YES	No
mbedTLS – 2.4.2	YES	No
OpenSSH – 7.7	No	-
OpenSSL – 1.0.2l	No	-
Openswan – 2.6.50 *	YES	YES
PuTTY – 0.7	No	-
strongSwan – 5.6.3 *	YES	YES
wolfSSL – 3.11.0	No	-

A Decade After Bleichenbacher '06, RSA Signature Forgery Still Works (2019)

https://i.blackhat.com/USA-19/Wednesday/us-19-Chau-A-Decade-After-Bleichenbacher-06-RSA-Signature-Forgery-Still-Works.pdf

Openswan 2.6.50	CVE-2018-15836

- 實作缺陷 : padding bytes 可以是任意字元

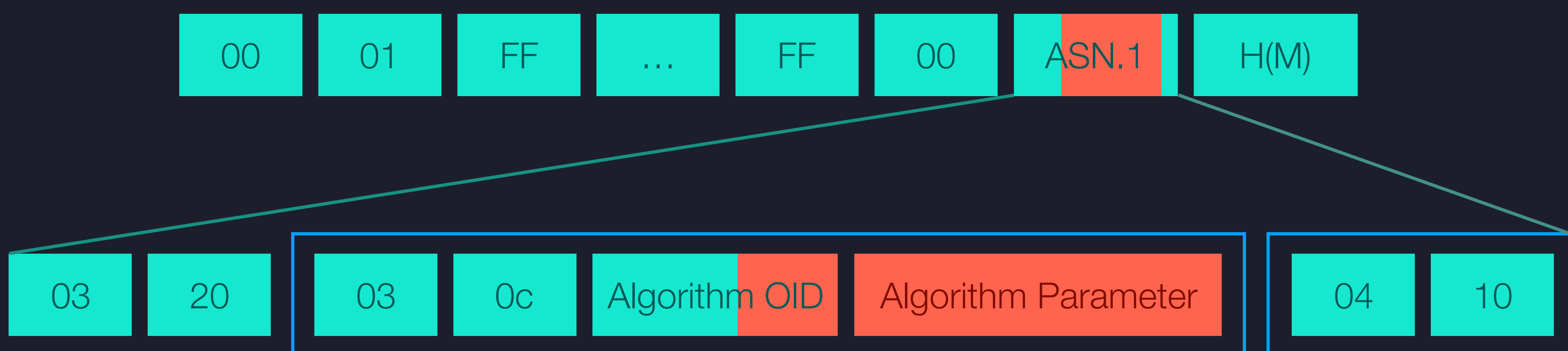
00	01	??	...	??	00	ASN.1	H(M)

A Decade After Bleichenbacher '06, RSA Signature Forgery Still Works (2019)

https://i.blackhat.com/USA-19/Wednesday/us-19-Chau-A-Decade-After-Bleichenbacher-06-RSA-Signature-Forgery-Still-Works.pdf

strongSwan 5.6.3	CVE-2018-16152

- 實作缺陷：
  - Algorithm Parameter 可以是任意字元
  - Algorithm OID 後面可以有多餘的字元

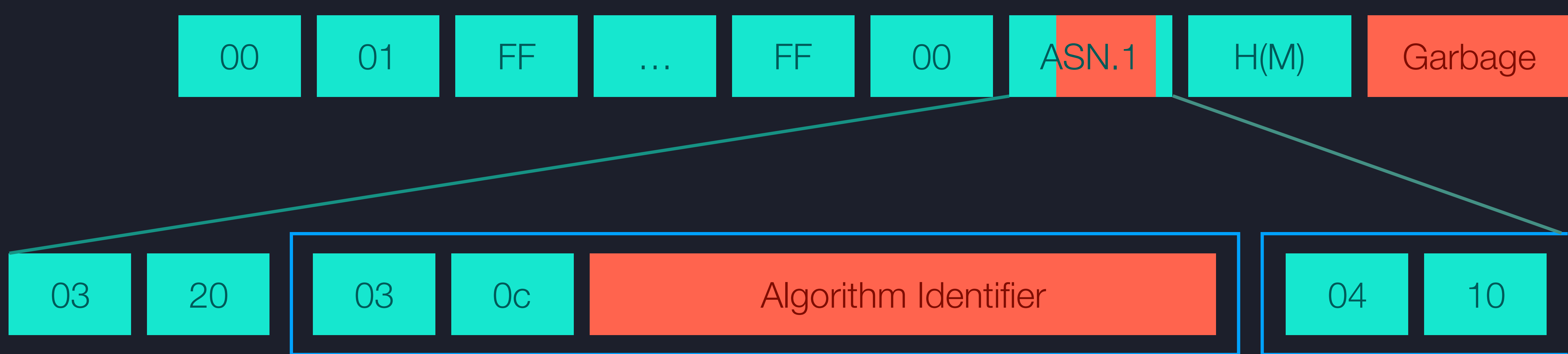


A Decade After Bleichenbacher '06, RSA Signature Forgery Still Works (2019)

https://i.blackhat.com/USA-19/Wednesday/us-19-Chau-A-Decade-After-Bleichenbacher-06-RSA-Signature-Forgery-Still-Works.pdf

axTLS 2.1.3 | CVE-2018-16150

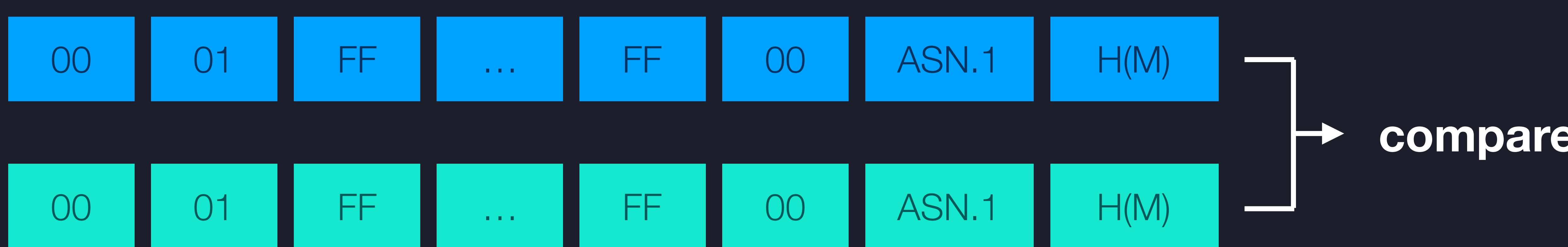
- 實作缺陷：
  - 可以有多餘的字元在後面
  - Algorithm Identifier 可以是任意字元



Defense against RSA Signature Forgery

How to defense?

- 用其他的簽章演算法，比如說 ECDSA
- 用更大的 e，比如 65537
- ~~parsing based~~ → comparison based



DSA

Parameter Generation

- 選一個 Hash Function H
- 選一個 N-bit prime q
- 選一個 L-bit prime p，使得 $q|p-1$
- 選一個 multipicative order 是 q 的 $g \in \{2, \cdots, p-1\}$
- 這些參數 (p, q, g) 可能是很多使用者很多平台共用的

Key Generation

- 選一個 private key $x \in \{1, \cdots, q-1\}$
- 計算 public key $y = g^x \mod p$

Sign

- 選一個 $k \in \{1, \cdots, q-1\}$
- 計算 $r = (g^k \mod p) \mod q$
- 計算 $s = k^{-1}(H(m) + xr) \mod q$
- (r, s) 就是簽章

Verify

- 計算 $w = s^{-1} \mod q$
- 計算 $u_1 = H(m) \cdot w \mod q$
- 計算 $u_2 = r \cdot w \mod q$
- 計算 $v = (g^{u1} y^{u2} \mod p) \mod q$
- 驗證 v 是否等於 r

DSA

Nonce Attack

Public Nonce

- 如果 Nonce k 被知道了，那就可以直接算出 private key x
- $x = r^{-1} \cdot (sk - H(m)) \mod q$
- 所以 Nonce k 必須保密

Repeated Nonce

- 如果重複使用 Nonce k，就可以算出 k，然後算出 x
- 以下計算都是在 modular q 下

$$
\begin{aligned}
s_1 &= k^{-1}(H(m_1) + xr) \\
s_2 &= k^{-1}(H(m_2) + xr) \\
\Rightarrow s_1 - s_2 &= k^{-1}(H(m_1) - H(m_2)) \\
\Rightarrow k &= (H(m_1) - H(m_2))(s_1 - s_2)^{-1}
\end{aligned}
$$

- 所以除了必須保密，還需要不能重複，也就是用隨機產生

- R. L. Rivest and R. D. Silverman, "Arestrong'primes needed for rsa?," in *The 1997 RSA Laboratories Seminar Series, Seminars Proceedings*, 1997.
- J. M. Pollard, "Theorems on factorization and primality testing," in *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 76, pp. 521–528, Cambridge University Press, 1974.
- H. C. Williams, "A p + 1 method of factoring," *Mathematics of Computation*, vol. 39, no. 159, pp. 225–234, 1982.
- M. J. Wiener, "Cryptanalysis of short rsa secret exponents," *IEEE Transactions on Information theory*, vol. 36, no. 3, pp. 553–558, 1990.
- B. De Weger, "Cryptanalysis of rsa with small prime difference," *Applicable Algebra in Engineering, Communication and Computing*, vol. 13, no. 1, pp. 17–28, 2002.

S. Maitra and S. Sarkar, "Revisiting wiener's attack–new weak keys in rsa," in *International Conference on Information Security*, pp. 228–243, Springer, 2008.

C.-Y. Chen, C.-C. Hsueh, and Y.-F. Lin, "A generalization of de weger's method," in *2009 Fifth International Conference on Information Assurance and Security*, vol. 1, pp. 344–347, IEEE, 2009.

A. Nitaj, "Diophantine and lattice cryptanalysis of the rsa cryptosystem," in *Artificial Intelligence, Evolutionary Computing and Metaheuristics*, pp. 139–168, Springer, 2013.

M. Asbullah, *Cryptanalysis on the Modulus N= p2q and Design of Rabin-like Cryptosystem Without Decryption Failure.*
PhD thesis, PhD thesis, Universiti Putra Malaysia, 2015.

M. Kamel Ariffin, S. Abubakar, F. Yunos, and M. Asbullah, "New cryptanalytic attack on rsa modulus n= pq using small prime difference method," *Cryptography*, vol. 3, no. 1, p. 2, 2019.

N. Aviram, S. Schinzel, J. Somorovsky, N. Heninger, M. Dankel, J. Steube, L. Valenta, D. Adrian, J. A. Halderman, V. Dukhovni, *et al.*, "{DROWN}: Breaking {TLS} using sslv2," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pp. 689–706, 2016.

H. Böck, J. Somorovsky, and C. Young, "Return of bleichenbacher's oracle threat ({ROBOT})," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pp. 817–849, 2018.

E. Ronen, R. Gillham, D. Genkin, A. Shamir, D. Wong, and Y. Yarom, "The 9 lives of bleichenbacher's cat: New cache attacks on tls implementations," in *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 435–452, IEEE, 2019.

D. J. Bernstein, Y.-A. Chang, C.-M. Cheng, L.-P. Chou, N. Heninger, T. Lange, and N. Van Someren, "Factoring rsa keys from certified smart cards: Coppersmith in the wild," in *International Conference on the Theory and Application of Cryptology and Information Security*, pp. 341–360, Springer, 2013.

M. Nemec, M. Sys, P. Svenda, D. Klinec, and V. Matyas, "The return of coppersmith's attack: Practical factorization of widely used rsa moduli," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1631–1648, ACM, 2017.

Wikipedia, "Elliptic curve — Wikipedia, the free encyclopedia." `http://en.wikipedia.org/w/index.php?title=Elliptic%20curve&oldid=927921391`, 2019.

[Online; accessed 15-December-2019].

F. Valsorda, "Bleichenbacher'06 signature forgery in python-rsa," May 2016.

S. Y. Chau, "A decade after bleichenbacher'06, rsa signature forgery still works,"