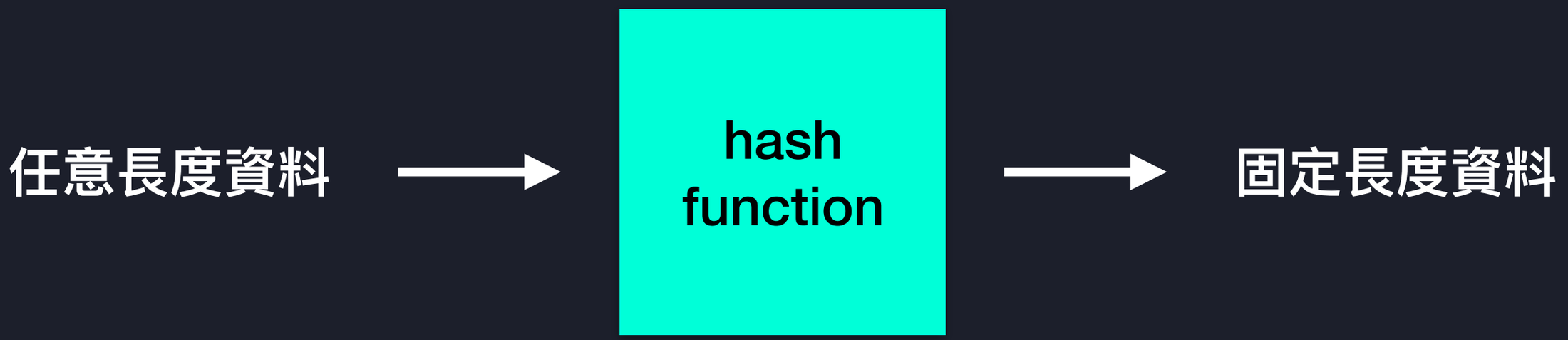
CRYPTO - HASH

OAlienO

目錄

Introduction to Hash

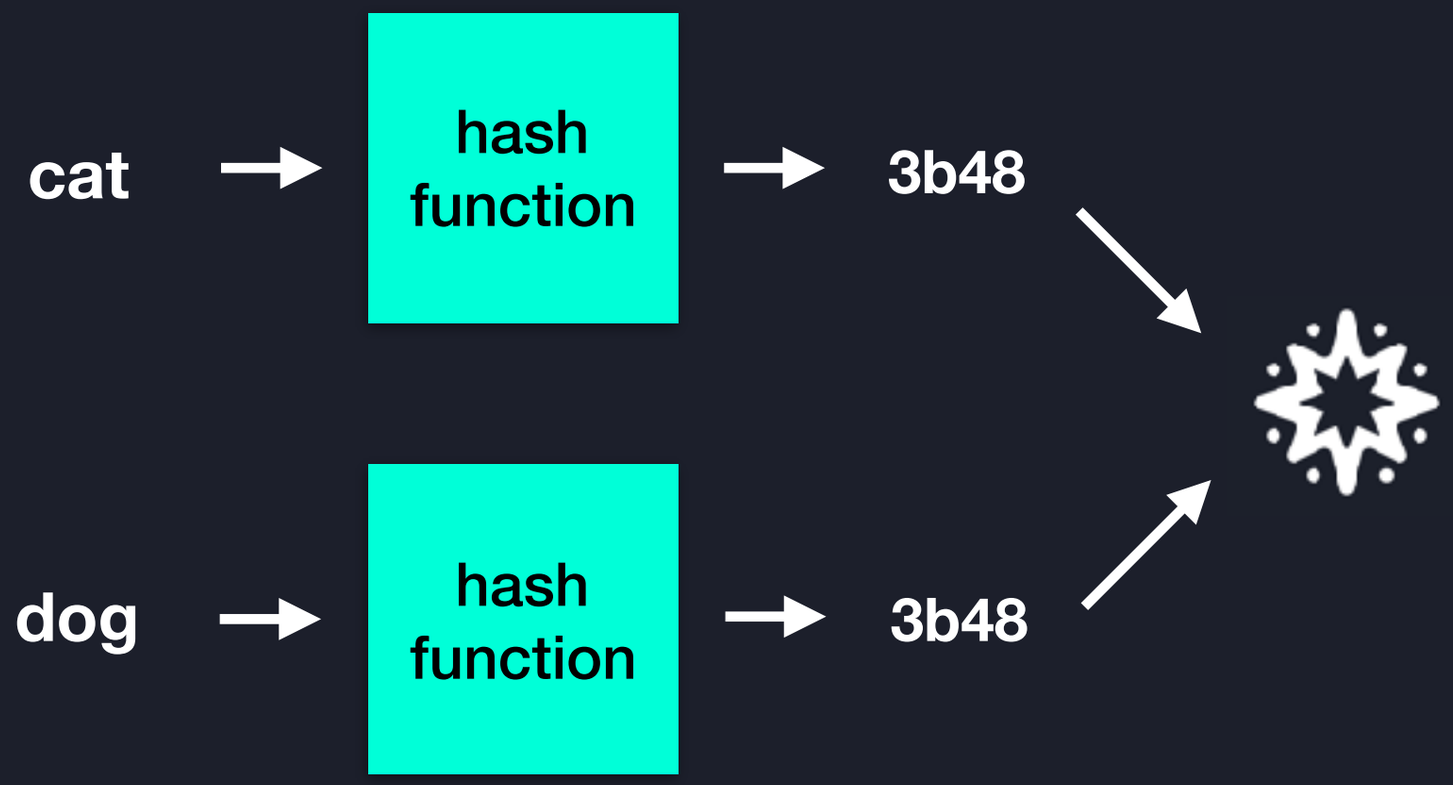
Introduction to Hash



特性：單向函式，不可逆

Introduction to Hash - 碰撞

碰撞 (Collision) :
hash function 是將無限大小的集合映射到有限大小的集合
根據鴿籠原理，產生夠多 hash value 後一定會有重複的
也就是碰撞 (Collision)



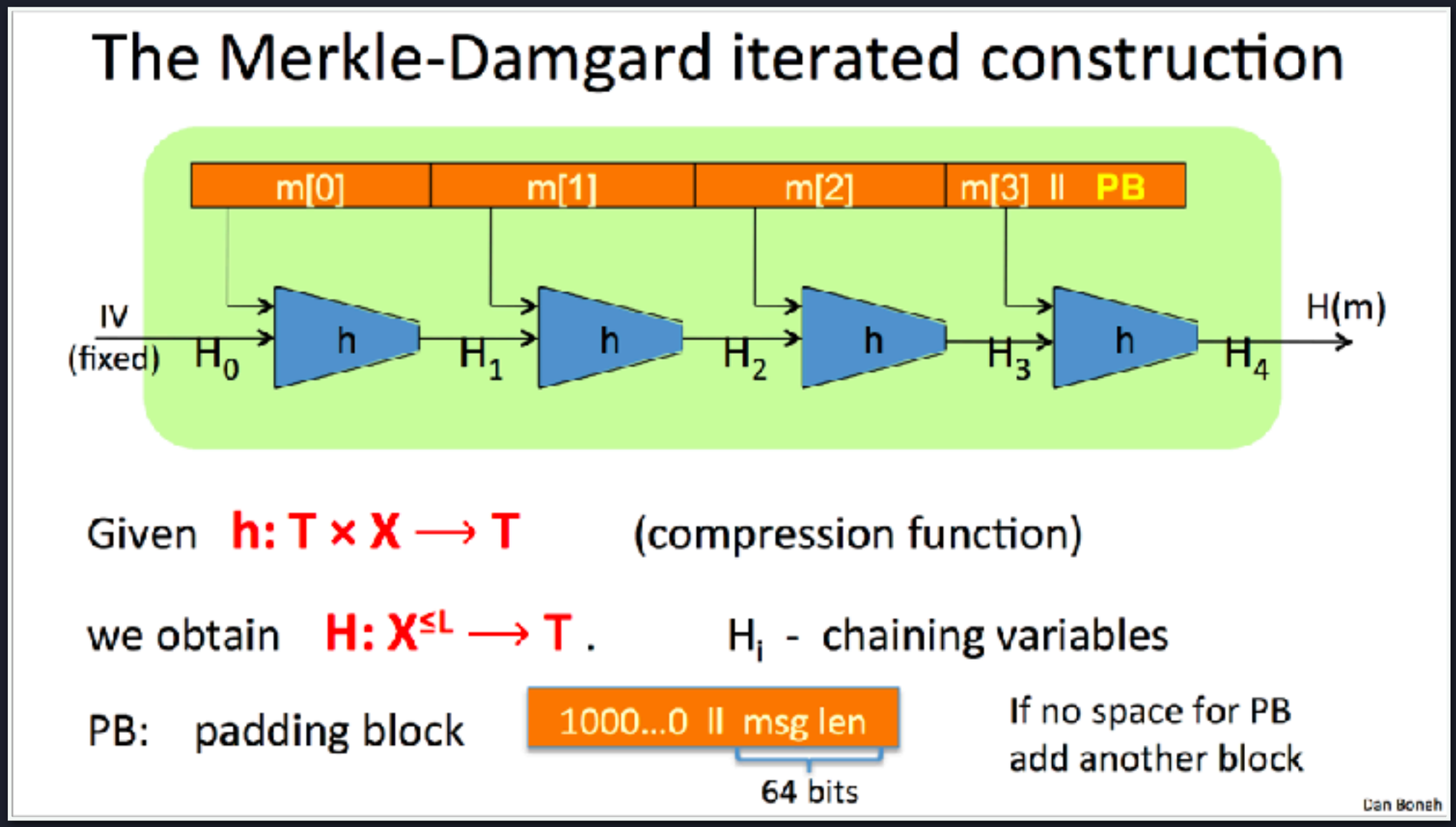
Introduction to Hash - 範例

名稱	輸出長度	區塊長度
MD4	128 bits (16 bytes)	512 bits (64 bytes)
MD5	128 bits (16 bytes)	512 bits (64 bytes)
RIPEMD-160	160 bits (20 bytes)	512 bits (64 bytes)
SHA0	160 bits (20 bytes)	512 bits (64 bytes)
SHA1	160 bits (20 bytes)	512 bits (64 bytes)
SHA256	256 bits (32 bytes)	1088 bits (136 bytes)
SHA512	512 bits (64 bytes)	576 bits (72 bytes)
WHIRLPOOL	512 bits (64 bytes)	512 bits (64 bytes)

Merkle–Damgård Construction

Merkle–Damgård Construction

https://www.youtube.com/watch?v=sawkPgsQPwg



許多的 hash function 是用這種構造方式 (ex: md5, sha1, sha256, …)

Merkle–Damgård Construction

m : *input*
h : *function*
H : *hash function*

$$m = m_0 || m_1 || \cdots || m_{n-1}$$
$$H_o = IV, \forall 1 \leq i \leq n : H_i = h(H_{i-1}, m_{i-1})$$
$$H(m) = H_n$$

不同的 hash function H 的差别是有不同的 function h

Get Your Hands Dirty

Get Your Hands Dirty - command line

```
$ echo 'AAAA' > file
$ md5sum file
ae5b468c7707a1f3d36c49b1fe2ef850  file
$ openssl dgst -md5 file
MD5(file)= ae5b468c7707a1f3d36c49b1fe2ef850
$ sha1sum file
11a62fcbdeda16e2aa22ff8551fee9d3d66985bb  file
$ openssl dgst -sha1 file
SHA1(file)= 11a62fcbdeda16e2aa22ff8551fee9d3d66985bb
```

Get Your Hands Dirty - python

使用 hashlib 套件

```
import hashlib
hashlib.md5(b"example").digest()    # b'\x1ay\xa4\xd6\r\xe6q\x8e\x8e[2n3\x8a\xe53'
hashlib.md5(b"example").hexdigest() #  '1a79a4d60de6718e8e5b326e338ae533'
```

使用 pycrypto 套件

```
from Crypto.Hash import MD5
MD5.new(b"example").digest()    # b'\x1ay\xa4\xd6\r\xe6q\x8e\x8e[2n3\x8a\xe53'
MD5.new(b"example").hexdigest() #  '1a79a4d60de6718e8e5b326e338ae533'
```

Get Your Hands Dirty - 線上破解

線上破解各種 HASH
crackstation
hashkiller

線上破解 MD5
CMD5
TTMD5
PMD5

Birthday Attack

Birthday Attack

用**生日悖論**估算暴力尋找**碰撞**需要多少時間

有 n 個人，每個人都可以從大小為 m 的集合中拿出一個元素

$p(n,m)$ 表示有兩個人選擇相同的數字的機率（等同於碰撞的機率）

$\overline{p}(n,m)$ 表示任兩個人選擇的數字都不相同的機率

顯然的，$p(n,m) = 1 - \overline{p}(n,m)$

$$\overline{p}(n,m) = \frac{m}{m} \cdot \frac{m-1}{m} \dots \frac{m-(n-1)}{m} = \frac{m!}{m^n(m-n)!}$$

$$p(n,m) = 1 - \frac{m!}{m^n(m-n)!}$$

Birthday Attack - 估算時間

接下來我們用 First order approximation of Taylor series expansion of e^x 來估算 $p(n, m)$

$e^x \approx 1 + x$

$p(n,m) = 1 - (1 - \frac{0}{m})(1 - \frac{1}{m}) \cdots (1 - \frac{n-1}{m}) \approx 1 - e^{-\frac{0}{m}} e^{-\frac{1}{m}} \cdots e^{-\frac{n-1}{m}} = 1 - e^{-\frac{n(n-1)}{2m}} \approx 1 - e^{-\frac{n^2}{2m}}$

假設我們想要 $p(n, m) = \frac{1}{2}$ (50%)

$\frac{1}{2} = 1 - e^{-\frac{n^2}{2m}}$

$e^{-\frac{n^2}{2m}} = \frac{1}{2}$

$-\frac{n^2}{2m} = -ln(2)$

$n = \sqrt{(2ln(2))m} \approx \sqrt{m}$

假設我們想要 $p(n, m) = \frac{99}{100}$ (99%)，$n = \sqrt{(2ln(100))m} \approx 3\sqrt{m}$

也就是說假設我們有一個長度是 a bits 的 hash function，我們只需要大約 $\sqrt{2^a} = 2^{a/2}$ 個選擇就很有可能找到一組 collision

Rainbow Table

Rainbow Table - 彩虹表的前身

字典攻擊是預先儲存一個很大的資料庫
裡面包含許多字串的 hash value
透過查表的方式找回 hash key
用空間換取時間

aaa	47bce5c7
aab	e62595ee
aac	a9ced3da
aad	c2f7ab46
...	...

Rainbow Table - 建立彩虹表

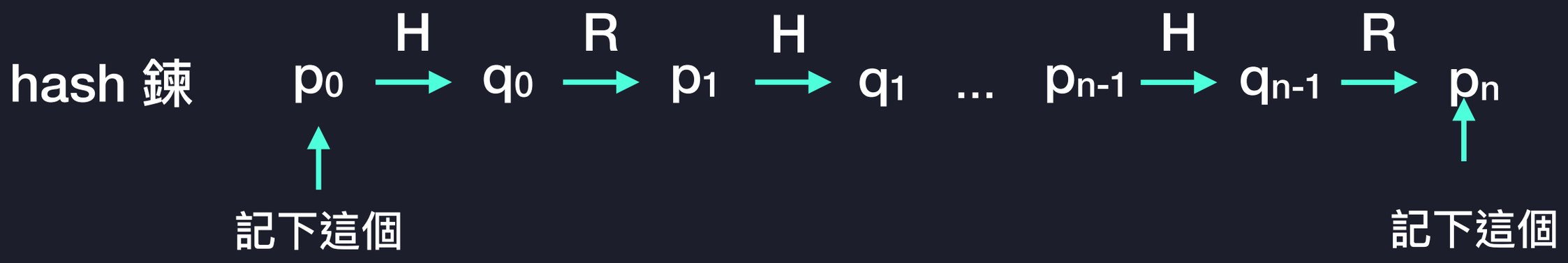
彩虹表和字典攻擊一樣是用空間換取時間，但**彩虹表**多犧牲了一點時間來換去更小的空間

我們有一個 hash function H，輸入任意長度的資料，輸出 n bits 的資料

定義另一個 function R，輸入 n bits 的資料，輸出 m bits 的資料

挑選許多初始值 p_0

計算$\forall 1 \leq i \leq n : q_{i-1} = H(p_{i-1}), p_i = R(q_{i-1})$ 後將 (p_0, p_n) 存入資料庫



Rainbow Table - 查找彩虹表

給 Q 我們要找 P 使得 $H(P) = Q$

那我們就找資料庫裡面有沒有 $p_n = R(Q)$，有的話 $P = p_{n-1}$，因為 $H(p_{n-1}) = Q$

沒有的話就繼續找資料庫裡面有沒有 $p_n = R(H(R(Q)))$，有的話 $P = p_{n-2}$，因為 $H(p_{n-2}) = Q$

然後就這樣掃過 hash chain

工具：

ophcrack
rainbowcrack
rtgen

MD5 Collision

MD5 Collision - 碰撞實測

https://www.mathstat.dal.ca/~selinger/md5collision/

將兩串不同的 data 寫入檔案

```
$ echo d131dd02c5e6eec4693d9a0698aff95c2fcab58712467eab4004583eb8fb7f8955ad340609f4b30283e488832571415a085125e8f7cdc99fd91dbdf280373c5bd8823e3156348f5bae6dacd436c919c6dd53e2b487da03fd02396306d248cda0e99f33420f577ee8ce54b67080a80d1ec69821bcb6a8839396f9652b6ff72a70 | xxd -r -p > A
$ echo d131dd02c5e6eec4693d9a0698aff95c2fcab50712467eab4004583eb8fb7f8955ad340609f4b30283e4888325f1415a085125e8f7cdc99fd91dbd7280373c5bd8823e3156348f5bae6dacd436c919c6dd53e23487da03fd02396306d248cda0e99f33420f577ee8ce54b67080280d1ec69821bcb6a8839396f965ab6ff72a70 | xxd -r -p > B
```

用 md5sum 看看他的 hash value

```
~/toolbox/md5-collision # md5sum A B
79054025255fb1a26e4bc422aef54eb4  A
79054025255fb1a26e4bc422aef54eb4  B
```

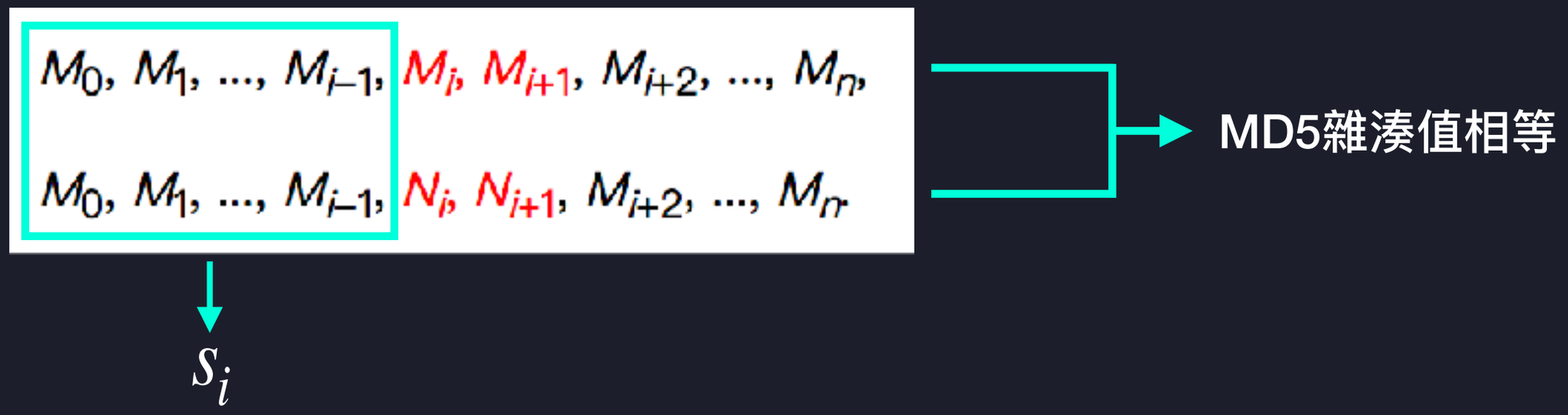
https://asciinema.org/a/hfdPyyIMMtNvEMKsMtv0JGO08

MD5 Collision - 碰撞應用

https://www.mathstat.dal.ca/~selinger/md5collision/

給任意的初始值 s_i 用 method of Wang and Yu 可以找到 M_i, M_{i+1} 和 N_i, N_{i+1} 使得

$$h(h(s_i, M_i), M_{i+1}) = h(h(s_i, N_i), N_{i+1})$$



用這種方法可以構造出下面這樣的程式 (他們的 MD5 相等)

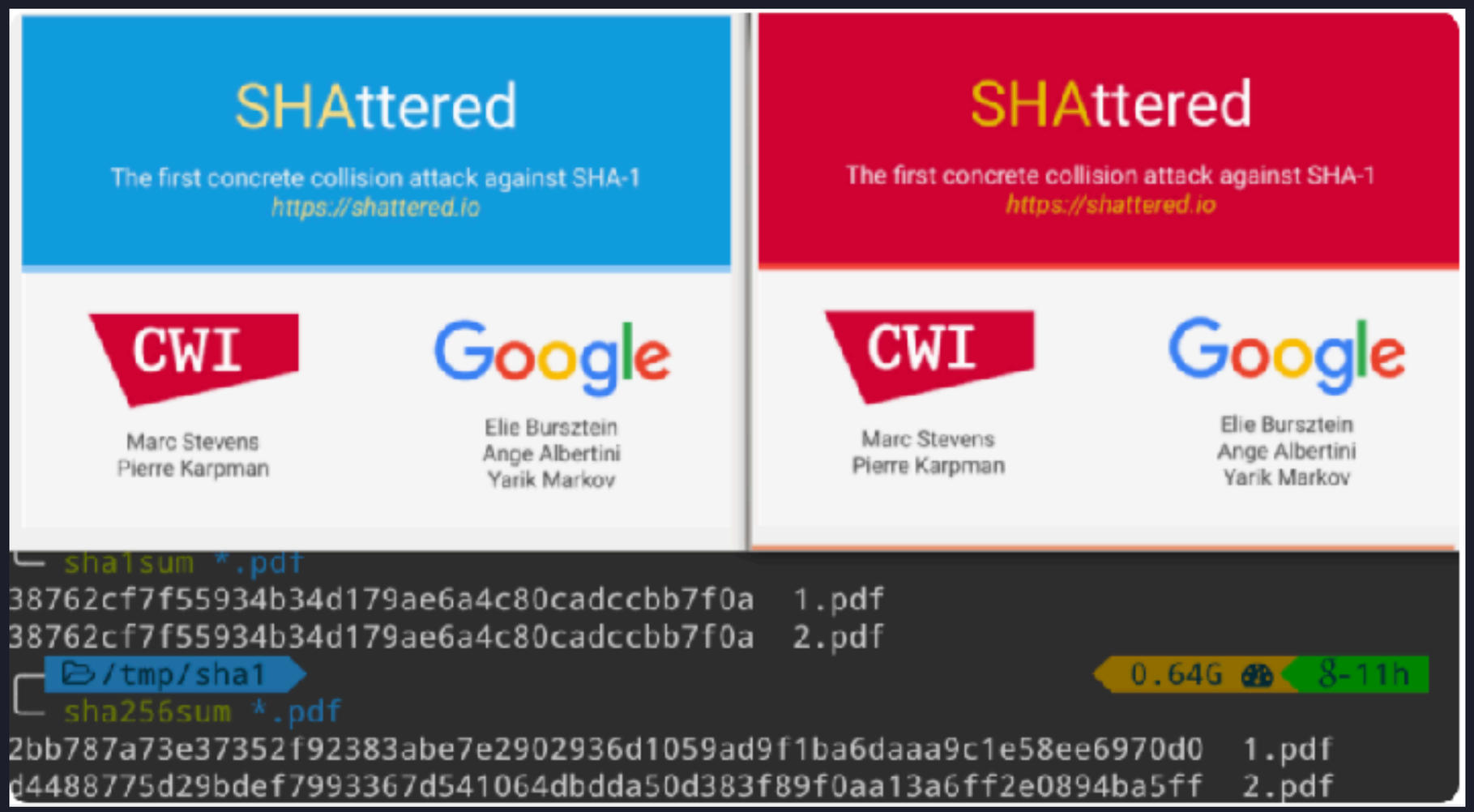
```
Program 1: if (data1 == data1) then { good_program } else { evil_program }
Program 2: if (data2 == data1) then { good_program } else { evil_program }
```

https://asciinema.org/a/oYBze8AXvRA3Xh8eqgjVX1axb

SHA1 Collision

SHA1 Collision

Google 在 2017 年 2 月 23 號
發表了兩份有一模一樣的 SHA1 checksum 的 PDF



https://shattered.io/

SHA1 Collision

Boston Key Party CTF 2017 在 2017 年 2 月 25 號

馬上出了一題 SHA1 collision 的題目

Boston Key Party CTF 2017 - prudentialv2

要讓 name != password 但是 sha1(name) == sha1(password)

直接把 google 找到的那兩個 PDF 傳過去就對了XD

```
if ($_GET['name'] == $_GET['password'])
    print 'Your password can not be your name.';
else if (sha1($_GET['name']) === sha1($_GET['password']))
  die('Flag: '.$flag);
```

SHA1 Collision

Seccon CTF 2017 - SHA-1 is dead 也是 SHA1 collision
上傳兩個檔案滿足：
1.file1 != file2
2.SHA1(file1) == SHA1(file2)
3.2017 KiB < sizeof(file1) < 2018 KiB
4.2017 KiB < sizeof(file2) < 2018 KiB

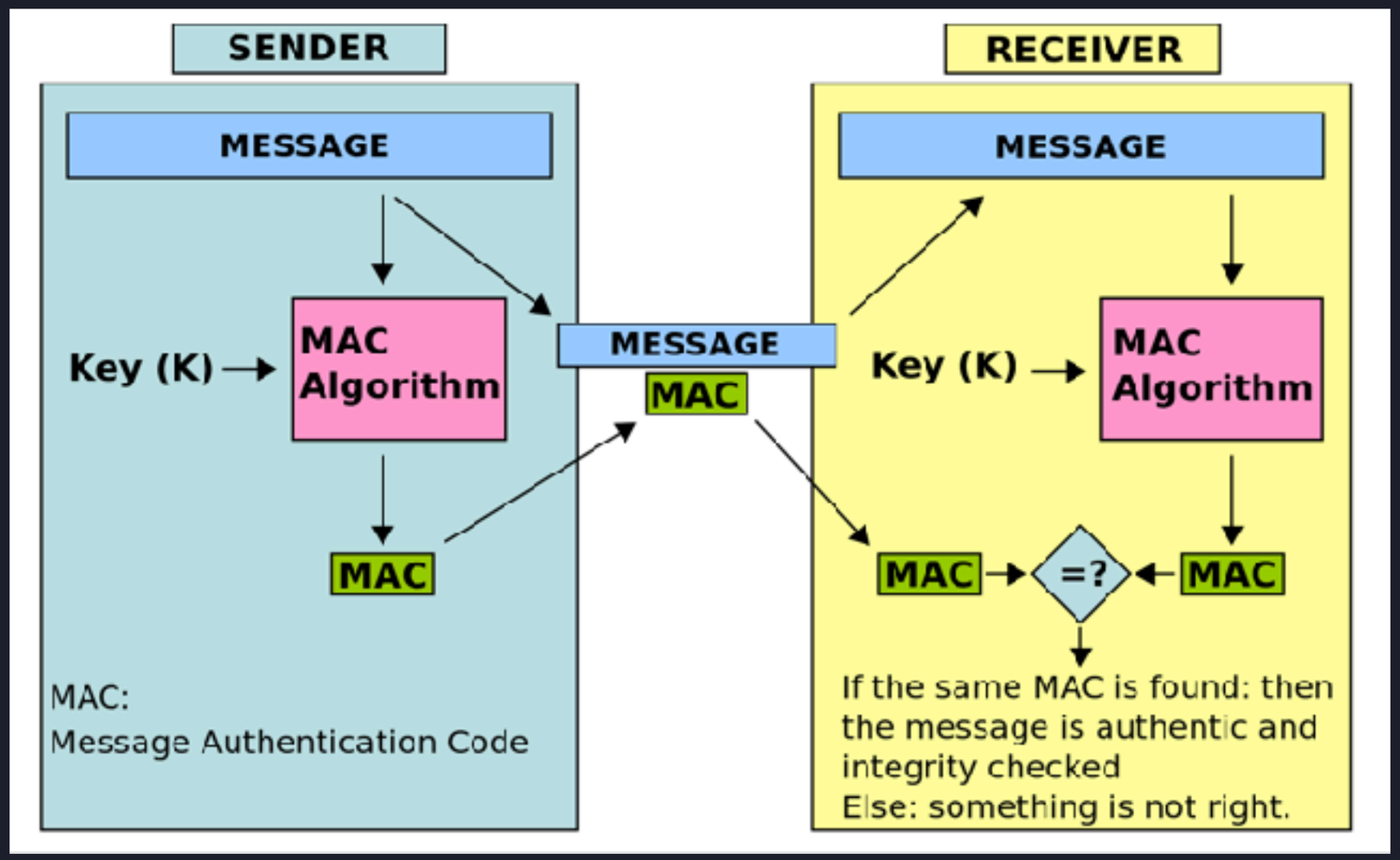
KiB = 1024 bytes
KB = 1000 bytes

```
$ python3 -c "print('A' * (2017 * 1024 - 422435 + 1), end = '')" >> shattered-1.pdf
$ python3 -c "print('A' * (2017 * 1024 - 422435 + 1), end = '')" >> shattered-2.pdf
```

Message Authentication Code

Message Authentication Code

https://en.wikipedia.org/wiki/Message_authentication_code



驗證訊息的完整性 (integrity)

keyed-Hash Message Authentication Code (HMAC)

$$HMAC(K, m) = H((K' \oplus opad)||H((K' \oplus ipad)||m))$$

m 是明文

K 是密鑰

K' 是 padding 過的密鑰

$opad, ipad$ 是 constant

需要密鑰的 MAC，驗證訊息的 integrity 和 authentication

Length Extension Attack

Length Extension Attack - 目的

當我們的 MAC Algorithm 是 $H(salt||message)$
而我們的 hash function H 是 Merkle–Damgård Construction
我們就可以繞過 MAC 的驗證機制，對明文串接部分可控資料

Length Extension Attack - 限制

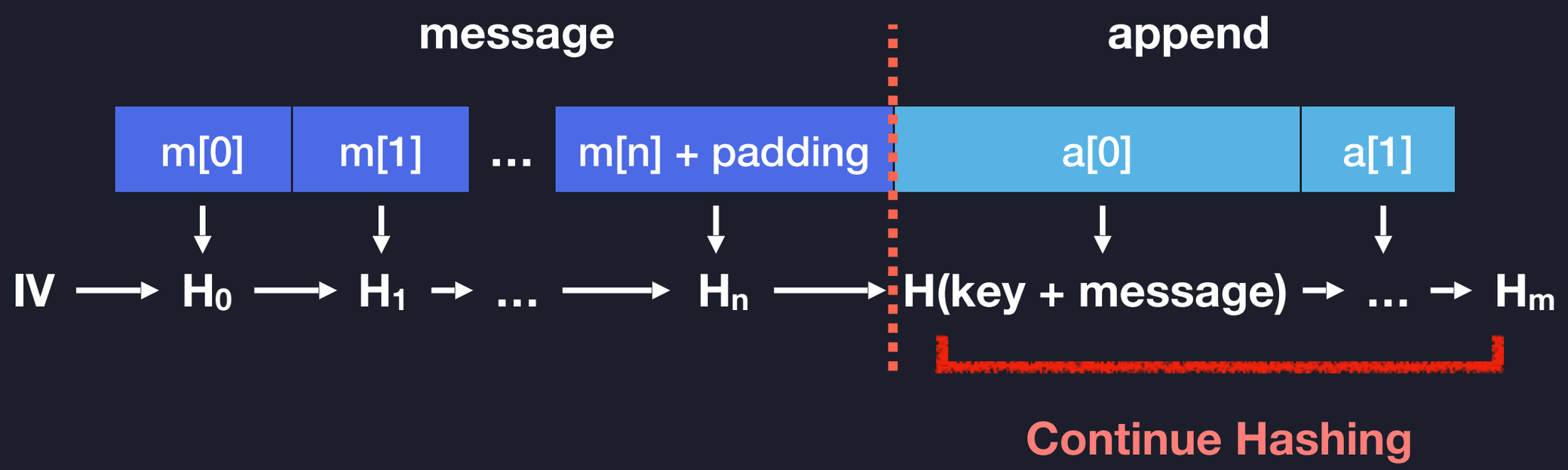
需要知道 $message$ 和 $H(salt||message)$ 和 $salt||message$ 的長度

Length Extension Attack - 原理

我們要繞過 MAC 驗證機制，其實就是在改動明文後重新計算驗證碼

我們把 $H(salt||message)$ 當作新的 IV' 把要串接的 $append$ 當作新的明文算下去就好

我們就可以成功算出新的明文 $message||padding||append$ 的驗證碼



Length Extension Attack - 工具

https://github.com/bwall/HashPump

--data : message

-s : H(salt || message)

-a : append

```
oalieno@oalieno  ~/HashPump  master  hashpump -s '6d5f807e23db210bc254a28be2d6759a0f5f5d9
9' --data 'count=10&lat=37.351&user_id=1&long=-119.827&waffle=eggo' -a '&waffle=liege' -k 14
0e41270260895979317fff3898ab85668953aaa2
count=10&lat=37.351&user_id=1&long=-119.827&waffle=eggo\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
0\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
02(&waffle=liege
```

new hash: 0e41270260895979317fff3898ab85668953aaa2

new message: count=10&lat=37.351&user_id=1&long=-119.827&waffle=eggo … &waffle=liege

-k : salt 長度

這邊的 \x80\x00... 就是 padding 填充字元

不用自己指定要用哪個 Hash Function
他會自動根據 H(salt || message) 這個 hash 的長度判斷

Length Extension Attack - CTF

TUM CTF Teaser - bad_apple : 裸 LEA

RuCTF Quals 2014 - MD5 lext : 裸 LEA

Teaser CONFidence CTF 2015 - Mac hacking : HMAC 的 LEA

BAMBOOFOX CTF 2018 - baby-lea : 裸 LEA