Q1. SSL/TLS

a. 1

(i) 使用 SSL/TLS 連線會有 key exchange，所以沒有 shared key 就辦法假裝成一個 connected client。

(ii) 因為 SSL/TLS 也有用到 CA 的機制，所以 Attacker 沒辦法假扮成 server，而除非 client 端也有自己的 certificate 可以讓 server 驗證，否則 Attacker 是有可能騙過 server 他的身分，但因為中間人必須要同時騙過 client、server 才能成功在這個連線中，所以只能騙過 server 也沒辦法危害到這個連線。

(iii) SSL/TLS 裡面每一次傳送都會有不同的 session key，所以每個 packet 使用的 share key 也不同，原本 key 加密的 packet 再被重新傳送一次也沒用因為 key 已經又不一樣了。

(iv) 因為 server 每次都會產生不同的 random bytes，所以沒辦法直接 replay，還是必須依照 SSL/TLS 的 protocol 來計算 share key 建立起連線。

b. forward secrecy 就是當破解了一個 cryptography 的 long-term key 時，然仍無法獲得他的 short-term key，或者是說當破解某個密碼系統時仍無法得到其過去所曾經傳送過加密訊息的 plaintext。

因為隨著算力的發展與新的演算法被提出等等，許多的 cryptography 在未來都很可能被破解，要是沒有 forward secrecy 的性質，代表現在正在傳送的秘密訊息未來總有一天也會被知道，這不是我們希望有的情況。

c. Rollback attack 就是若有一個 Attacker 可作為 client-server 的中間人，當一開始 client hello 的時候因為是 plaintext 所以 Attacker 可以任意篡改這個資訊，把 client 聲明可支援的 protocol version 降級成 SSL 2.0，server 以為 client 只支援到 SSL 2.0 的版本，所以之後的連線就變成 SSL 2.0 這個比較不安全的 protocol，尤其在 SSL 2.0 最後 finished stage 並不會比對雙方收送的資訊，所以 client、server 也無法發現有中間人介入。
預防的方法只能讓所有 server 都僅提供 SSL 2.0 之後的版本，因為在 SSL 3.0 之後的版本在 finished stage 時會再確認內容有無被篡改。

d. 因為一般使用者在使用連線的時候不會特別打 https 的網址，瀏覽器一般只會直接自動完成 http 的網址，所以如果連線之中有一個中間人的 Attacker，就可以攔截到 client 發出的 http 網址連線，就算 server 會 redirect 到 https 的安全連線，但只要 Attacker 在中間 forward 資訊就可以

讓使用者一直都在 http 的不安全連線之下暴露所有的傳送訊息。
HSTS：讓 server 告訴使用者的瀏覽器之後都只能用 https 的安全連線來互動，使用方法有兩個。(1) 瀏覽器在第一次連線到這個 server 之後收到 server 的通知，以後連線自動使用 https 安全連線。(2) server 事先跟瀏覽器申請，瀏覽器會有一個 list，連線到 list 裡面的 server 時都自動使用 https。

Q2. BGP

(1)
如果 AS 1 願意幫 AS 1000 轉送封包的話，可以 announce {10.10.12.0/22 , {AS 1→AS 1000}}，然後 withdraw {10.10.12.0/22 , {AS 4→AS 1000}}，這樣接下去 announce 的 BGP advertisement 就會 prepend 在 AS 1 之前，有封包要傳送到 AS 1000 的時候就一定會從 AS 1 這一邊傳送而不會經由 AS 4 與 AS 1000 間的連結。

(2)
就連離 AS 1000 比較近的 AS 都往 Attacker 這邊傳送封包，所以 Attacker 一定不只是對路徑篡改等等而已，因為 AS 1000 的 announcement 為 {10.10.12.0/22 , {AS 1000}}，所以 AS 999 可以 announce {10.10.12.0/23 , {AS 999}}，讓其他 AS 在選擇正確的 IP 時優先權會排在真正的 AS 1000 之前，所有的封包都會往 AS 999 這邊送。

(3)

a. AS 999 可以 announce 擁有一個更長的 prefix，例如：10.10.12.0/23，然後再 prepend AS 2、AS 1、AS 1000 到 AS path 上。
BGP update =>{10.10.12.0/23 , {AS 1000→AS 1→AS 2→AS 999}}。

b. Attacker 必須要有一些 AS 能夠幫他 forward 封包到 victim 那邊才能不被發覺，所以至少要有一條 AS path 到 AS 1000，若是只有直接 announce 一個更 specified 的 IP 會讓所有通往 AS 1000 的 AS path 都只往 AS 999 傳，所以就要利用 AS Path Prepending、Loop prevention。

AS Path Prepending 讓 Attacker 可以任意在 AS path 上 prepend 新的 AS，所以 Attacker prepend AS 1、AS 2、AS 1000 在一條 AS path 上，由於 Loop prevention 的關係，AS 1、AS 2、AS 1000 看到了有自己在裡面的 AS path 都不會接受，使得封包仍會往原來 AS 1000 – 10.10.12.0/22 的地方傳，所以之後 Attacker 就能利用保留下來的這條 path 來傳回給

原來的 victim，而所有其他不在 prepended path 裡面的 AS 其封包都匯往 AS 999 傳了。

c.
Advantage：除非 latency 太大，否則 User 不容易發現。
Disadvantage：會犧牲一條 path 上面的 AS，在這裡面傳送給 victim 的封包是 Attacker 沒辦法拿到的。

Q3. SYN Cookies

(1)
因為最原始的 TCP handshake 設計只要有 client 傳送 SYN 給 server，server 就會在自己的 connection table 記錄這筆請求的相關資訊並且等待 client 進一步的 ACK，使得攻擊者可以針對這個 table 作為攻擊的目標，只要攻擊者在短時間內發送多個 SYN 給 server，server 的 table 就會被塞滿而無法容納其他正常使用者的連線請求，攻擊者即利用 server 還沒確定 client 端就為其儲存資料以及 table limited size 來癱瘓 server 的正常連線。

(2)
要是 cookie 不包含 timestamp 的話，Attacker 可以先建立一次連線請求就拿到 cookie，之後的連線請求所使用的 cookie 也都可以直接用這一個，server 也都必須接受。

(3)
因為 server 端完全沒有先儲存任何資料，cookie 中如果有 client IP 也可以防止 Attacker 偷拿別人傳送的資訊來 replay。

(4)
如果 Attacker 可以自己算出跟 server 一樣的 MAC，那就可以自己偽造大量的 cookie 丟給 server，一樣能夠成功塞滿 server 的 connection table。

Q4. NS Protocol Revenge

reference : “Two attacks on Neuman-Stubblebine authentication protocols”

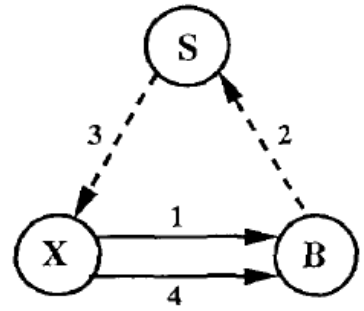
(1) 因為 B 在一開始收到訊息的時候並不會對傳訊息的一方做認證，所以我

們直接宣稱自己是 A 後加上一個 Nonce Nx 傳給 B 他就會直接加到某串 plaintext 用跟 server 的 share key K_BS 加密後傳送。

之後我們會得到 server 傳送過來的訊息，裡面會有用 K_AS 加密的真正 A、B 之間的 share key，但是我們也無法、也不需要知道 K_AS 或是 K_AB，我們只要隨便取一個 key 加密 Nonce B 並且加上之前 B 傳給 Server 的一段 message 就可以冒充有真正的 share key K_AB。

```
b'Hi Alice, here is the flag: BALSN{M1dT3rM_i5_SO_h4rD_QAQ}'
```

(如下圖)



(1) $X \rightarrow B : A, N_x$
(2) $B \rightarrow S : B, \{A, N_x, T_b\}K_{bs}, N_b$
(3) $S \rightarrow X :$ (ignore)
(4) $X \rightarrow B : \{A, N_x, T_b\}K_{bs}, \{N_b\}N_x$

(2)想要通過 B 的認證假裝成 A，主要的部分就是要用 share keyK_AB 對 B 傳來的 Nonce B 加密。但在這之前的通訊中可以發現 B 其實會對我們傳送過去的 Nonce X 先用 share key K_AB 做了一次加密，因此我們可以使用 oracle attack，請求第一次連線時收到 B 希望我們加密的 Nonce N_b'，之後另開第二條連線將 N_b' 丟給 B 加密，於是我們就會有用 K_AB 加密的 N_b'了，之後再回到第一條的連線請求將這個 ciphertext 丟給 B 就完成認證，B 會以為我們真的有 share key K_AB。

```
Oh! Welcome back Alice, this is another flag: BALSN{R3fl3CtloN_4774cK_SO_pOw3RfuL}
```

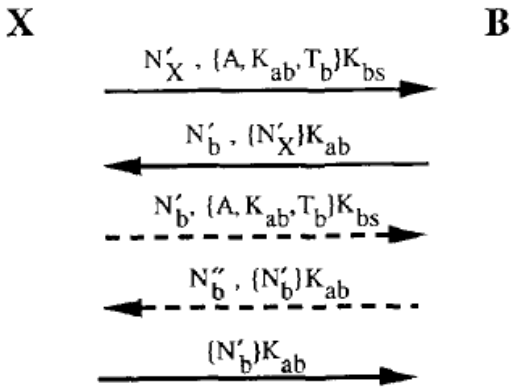
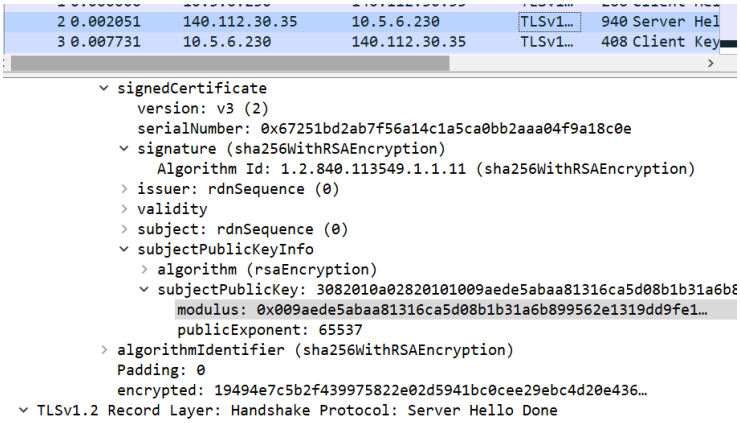


Fig. 3. Parallel session attack.

Q5. TLS



因為 modulus n 的 factor p,q 距離很近，我們可以使用 Fermat factorization 很快

```
(python2) ksc@ksc-All-Series:~/Desktop$ python ./RsaCtfTool-master/RsaCtfTool.py --publickey ~/Deskto
p/key.pub --verbose --private
[*] Performing hastads attack.
[*] Performing factordb attack.
[*] Performing pastctfprimes attack.
[*] Loaded 71 primes
[*] Performing noveltyprimes attack.
[*] Performing smallq attack.
[*] Performing wiener attack.
[*] Performing comfact_cn attack.
[*] Performing fermat attack.
-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEAmu3lq6qBMWyl0IsbMaa4mVYuExndn+GL7fImnFBYFnADNvXS
3jxF3dB9QcsWVA8AXMs00d9Umqwwqo9Nly8JRg7xE9QzwoG+JiQAh7LuwWpJGnQE
jNke+hWENduM3/jdN6e45yOsE/iFHqtLloQdhOpyBH5e7FWbulDkT0IlDo78Xbv2
hiJIEB0KhoifLGfz4fcpNXFQgXusIU+yWgt6NreH/zTlHWTBpX4AcoiGBpRCtTun
zqSp/cJMBbUHdkc1VqDNWP+SC4Kw6ejQ3oBwraiHN9rJI3q+4wvfbhMoE88yP1bk
EIrXN6MRiDuNfukGc3E6rTooYy7Z0CokKOrg/QIDAQABAoIBADF86C4LirUE5XpV
1NvpS3H/azxCuN5HJ3FcNiUcLOcN5Jj7sN24FwEhhcEsY/d5yz3Y0whu9BFsCyeQ
GtK9/ZJZV3xj/hyVYO3B9kFiUI2Lq8o9YO60tAZiNRcXCErKraBHbjK23FKMwcjo
In7zuJIqaT0yOJP5TIMZL7R5hmi9mHho00mGo2dsuuSBpnJHk3FYwuA/t7PwLoyO
j8bqRjb0iAu4eyX8kq4/5yq6ouaWobK8wFOXdor0vW6yO7pQ/TIOFzk+fXVXcvPz
XIClBdmFWtRgaJi0lOMNBgseAqIJw/VXd89zOdbiga38wDB3Jx/qR/RZ2Tsfaytw
0rqrEKkCgYEAxycnrLztshgmLnqWTMZGVWhu+Jsx0lKMK+6UQW425FqpuAM4R90J
cahLxXwAQlIPEBpTjhlQjtkYttqS1p1958phPqPz8YBOzsmWzhsOxou8fSwWG/J8
1+aEiR0TOnlOv7+BTC2fyLsr2oBHqMeUQug4Fu0wefs0NMeI+4eQDLsCgYEAxycn
rLztshgmLnqWTMZGVWhu+Jsx0lKMK+6UQW425FqpuAM4R90JcahLxXwAQlIPEBpT
jhlQjtkYttqS1p1958phPqPz8YBOzsmWzhsOxou8fSwWG/J81+aEiR0TOnlOv7+B
TC2fyLsr2oBHqMeUQug4Fu0wefsuyQAqziSACacCgYAAiz/VffKcQOqhx93vOb9p
bk6XST06mNUfLZikERv/Rawe/4tAzv6+m7oRQQRzKew2nAvMYI9SxI8HuNMb2PRG
FMxtMl9xMmuErxJFX9eLfUg3g05QJzpmEJAeG8E4sKohZfGBdsJv9urovt7zPBBQ
mnfEwEMYojIDTuQ146nmDwKBgGcdeehxpi0ZXeOg62NzIgWk3hhw2roIrbUtgavj
Mh7IZqn1WNW0t+gWeK0XCFdOf6gZyXuA1fowpCBBSdg/kWpLmlBBudQoWCWvsRqk
XeElhwHUqtnrkwKCw8rut/8FwmV79amI0fuwFgO7ed8E/1opPKl6RChUiyPdp0w3
D+hvAoGAFg9j1p053FD+Xa10AkRy3hYA0n8HMjVUvXS5ogw+15rRICJO9tjMFe6W
sYiH7DZaZLLh+8GGWrujMcedftH+55X3c4/Qtzb5bDroZ+lx7i4IceTnFpIky6o1
sGWfQBj9SieZj+t0v3CPvCtdDprw0IxkDwo2dKMY8fi3AC0QRRU=
-----END RSA PRIVATE KEY-----
```

```
(base) ksc@ksc-All-Series:~/Desktop$ cat private.key
-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEAmu3lq6qBMWyl0IsbMaa4mVYuExndn+GL7fImnFBYFnADNvXS
3jxF3dB9QcsWVA8AXMs00d9Umqwwqo9Nly8JRg7xE9QzwoG+JiQAh7LuwWpJGnQE
jNke+hWENduM3/jdN6e45yOsE/iFHqtLloQdhOpyBH5e7FWbulDkT0IlDo78Xbv2
hiJIEB0KhoifLGfz4fcpNXFQgXusIU+yWgt6NreH/zTlHWTBpX4AcoiGBpRCtTun
zqSp/cJMBbUHdkc1VqDNWP+SC4Kw6ejQ3oBwraiHN9rJI3q+4wvfbhMoE88yP1bk
EIrXN6MRiDuNfukGc3E6rTooYy7Z0CokKOrg/QIDAQABAoIBADF86C4LirUE5XpV
1NvpS3H/azxCuN5HJ3FcNiUcLOcN5Jj7sN24FwEhhcEsY/d5yz3Y0whu9BFsCyeQ
GtK9/ZJZV3xj/hyVYO3B9kFiUI2Lq8o9YO60tAZiNRcXCErKraBHbjK23FKMwcjo
In7zuJIqaT0yOJP5TIMZL7R5hmi9mHho00mGo2dsuuSBpnJHk3FYwuA/t7PwLoyO
j8bqRjb0iAu4eyX8kq4/5yq6ouaWobK8wFOXdor0vW6yO7pQ/TIOFzk+fXVXcvPz
XIClBdmFWtRgaJi0lOMNBgseAqIJw/VXd89zOdbiga38wDB3Jx/qR/RZ2Tsfaytw
0rqrEKkCgYEAxycnrLztshgmLnqWTMZGVWhu+Jsx0lKMK+6UQW425FqpuAM4R90J
cahLxXwAQlIPEBpTjhlQjtkYttqS1p1958phPqPz8YBOzsmWzhsOxou8fSwWG/J8
1+aEiR0TOnlOv7+BTC2fyLsr2oBHqMeUQug4Fu0wefsuyQAqziSACacCgYAAiz/VffKcQOqhx93vOb9p
bk6XST06mNUfLZikERv/Rawe/4tAzv6+m7oRQQRzKew2nAvMYI9SxI8HuNMb2PRG
FMxtMl9xMmuErxJFX9eLfUg3g05QJzpmEJAeG8E4sKohZfGBdsJv9urovt7zPBBQ
mnfEwEMYojIDTuQ146nmDwKBgGcdeehxpi0ZXeOg62NzIgWk3hhw2roIrbUtgavj
Mh7IZqn1WNW0t+gWeK0XCFdOf6gZyXuA1fowpCBBSdg/kWpLmlBBudQoWCWvsRqk
XeElhwHUqtnrkwKCw8rut/8FwmV79amI0fuwFgO7ed8E/1opPKl6RChUiyPdp0w3
D+hvAoGAFg9j1p053FD+Xa10AkRy3hYA0n8HMjVUvXS5ogw+15rRICJO9tjMFe6W
sYiH7DZaZLLh+8GGWrujMcedftH+55X3c4/Qtzb5bDroZ+lx7i4IceTnFpIky6o1
sGWfQBj9SieZj+t0v3CPvCtdDprw0IxkDwo2dKMY8fi3AC0QRRU=
-----END RSA PRIVATE KEY-----
```

的進行質數分解，這邊直接使用一個 RSACtfTool 來得到 RSA 的 private key 並存成 private.key。

便可以用 openssl 來看到詳細的資訊，其中 prime1、prime2 就是 n 分解出來的質數 p、q。

```
(python2) ksc@ksc-All-Series:~/Desktop$ openssl rsa -in private.key -text -noout
RSA Private-Key: (2048 bit, 2 primes)
modulus:
    00:9a:ed:e5:ab:aa:81:31:6c:a5:d0:8b:1b:31:a6:
    b8:99:56:2e:13:19:dd:9f:e1:8b:ed:f2:26:9c:50:
    58:16:70:03:36:f5:d2:de:3c:45:dd:d0:7d:41:cb:
    16:54:0f:00:5c:cb:34:d1:df:54:9a:ac:30:aa:8f:
    4d:97:2f:09:46:0e:f1:13:d4:33:c2:81:be:26:24:
    00:87:b2:ee:c1:6a:49:1a:74:04:8c:d9:1e:fa:15:
    84:35:db:8c:df:f8:dd:37:a7:b8:e7:23:ac:13:f8:
    85:1e:ab:4b:96:84:1d:84:ea:72:04:7e:5e:ec:55:
    9b:ba:50:e4:4f:42:25:0e:8e:fc:5d:bb:f6:86:22:
    48:10:1d:0a:86:88:9f:2c:67:f3:e1:f7:29:35:71:
    50:81:7b:ac:21:4f:b2:5a:0b:7a:36:b7:87:ff:34:
    e5:1d:64:c1:a5:7e:00:72:88:86:06:94:42:b5:3b:
    a7:ce:a4:a9:fd:c2:4c:05:b5:07:76:47:35:56:a0:
    cd:58:ff:92:0b:82:b0:e9:e8:d0:de:80:70:ad:a8:
    87:37:da:c9:23:7a:be:e3:0b:df:6e:13:28:13:cf:
    32:3f:56:e4:10:8a:d7:37:a3:11:88:3b:8d:7e:e9:
    06:73:71:3a:ad:3a:28:63:2e:d9:d0:2a:24:28:ea:
    e0:fd
publicExponent: 65537 (0x10001)
privateExponent:
    31:7c:e8:2e:0b:8a:b5:04:e5:7a:55:d4:db:e9:4b:
    71:ff:6b:3c:42:b8:de:47:27:71:5c:36:25:1c:2c:
    e7:0d:e4:98:fb:b0:dd:b8:17:01:21:85:c1:2c:63:
    f7:79:cb:3d:d8:d3:08:6e:f4:11:6c:0b:27:90:1a:
    d2:bd:fd:92:59:57:7c:63:fe:1c:95:60:ed:c1:f6:
    41:62:50:8d:8b:ab:ca:3d:60:ee:b4:b4:06:62:35:
    17:17:08:4a:ca:ad:af:07:6e:32:b6:dc:52:8c:c1:
    c8:e8:22:7e:f3:b8:92:2a:69:3d:32:38:93:f9:4c:
    83:19:2f:b4:79:86:68:bd:98:78:68:d3:49:86:a3:
    67:6c:ba:e4:81:a6:72:47:93:71:58:c2:e0:3f:b7:
    b3:f0:2e:8c:8e:8f:c6:ea:46:36:f4:88:0b:b8:7b:
    25:fc:92:ae:3f:e7:2a:ba:a2:e6:96:a1:b2:bc:c0:
    53:97:76:8a:f4:bd:6e:b2:3b:ba:50:fd:32:0e:17:
    39:3e:7d:75:57:72:f3:f3:5c:80:a5:05:d9:85:5a:
    d4:60:68:98:b4:94:e3:0d:06:0b:1e:02:a2:09:c3:
    f5:57:77:cf:73:39:d6:e2:81:ad:fc:c0:30:77:27:
    1f:ea:47:f4:59:d9:3b:1f:6b:2b:70:d2:ba:ab:10:
    a9
prime1:
    00:c7:27:27:ac:bc:ed:b2:18:26:2e:7a:96:4c:c6:
    46:55:68:6e:f8:9b:31:d2:52:8c:2b:ee:94:41:6e:
    36:e4:5a:a9:b8:03:38:47:dd:09:71:a8:4b:c5:7c:
    00:42:52:0f:10:1a:53:8e:19:50:8e:d9:18:b6:da:
    92:d6:9d:7d:e7:ca:61:3e:a3:f3:f1:80:4e:ce:c9:
    96:ce:1b:0e:c6:8b:bc:7d:2c:16:1b:f2:7c:d7:e6:
    84:89:1d:13:3a:79:4e:bf:bf:81:4c:2d:9f:c8:bb:
    2b:da:80:47:a8:c7:94:42:e8:38:16:ed:30:79:fb:
    34:34:c7:88:fb:87:90:0c:bb
prime2:
    00:c7:27:27:ac:bc:ed:b2:18:26:2e:7a:96:4c:c6:
    46:55:68:6e:f8:9b:31:d2:52:8c:2b:ee:94:41:6e:
    36:e4:5a:a9:b8:03:38:47:dd:09:71:a8:4b:c5:7c:
```

之後到 wireshark > preference > protocol > SSL，RSA key list 新增 server 的 IP 以及選擇 private.key，這樣 wireshark 就可以使用這個找到的 private key 替我們 decrypt 所有的 packet 資訊。
但即使完全不使用套件也可以直接用 Fermat Factorization 直接做質因數分解來找到 private key，存成 private.key 後一樣可以給 wireshark 來用。

```
0000  42 41 4c 53 4e 7b 43 48  4f 4f 53 45 5f 43 49 50   BALSN{CH OOSE_CIP
0010  48 45 52 5f 53 55 49 54  5f 43 41 52 45 46 55 4c   HER_SUIT _CAREFUL
0020  4c 59 7d                                          LY}
```

Q6. Eve's Revenge

```
def integer_slot_collisions(max=1000):

    print 'Slot collision attack:'

    # Fill the dict slots starting at (hash) position 1
    d = {1:1}
    i = 1
    perturb = i
    print 'seeding the dict:'
    for x in xrange(max):
        i = ((i << 2) + i + perturb + 1) & 0xffffffffffffffff
        #print 'adding %i with hash %i' % (i, hash(i))
        d[i] = 1
        perturb >>= 5

    # Add new keys
    print 'generating slot collisions:'
    # cause a hash collision on the first try to enter the
    # prepared slot collision path
    i = 1*(2**64 - 1) + 1
    for x in xrange(max):
        #print 'adding %i with hash %i' % (i, hash(i))
        d[i] = 1
    print 'generated dict with %i items' % len(d)
    return d
```

我們的目標是產生一個大量 collision 的字典，參考網路上的 slot collision，製造一個 key 範圍在 2**30 內的字典，所以我們必須把其中的程式碼修改一下，只保留最後 30 bits。

所以修改

a. & 0xffffffffffffffff → & 0x3fffffff

b. i = 1*(2**64-1) + 1 → i = 1*(2**30-1) + 1

之後便會產生一個 dictionary，我們就可以把 dict.keys() 丟給 server，必須特別提到的是我們一開始聲明的 dict size 是 50000 但生成的 dict size 只設定在 20000 左右，剩下的都丟 0 給 server，所以總共仍會丟 50000 個 keys 給 server。

```
Oh no my resource is exhausted! Here is the flag:

BALSN{Py7h0n_4lg@r!thmic_Comp13Xity_Att4ck}
```