

Write-Up DM SSHBurned

Pour commencer le DM, on ouvre l'image fournie.

On la fournit ensuite à un site Internet permettant de transformer l'image donnée en texte, comme <https://img2txt.com/en>.

Puis on vérifie qu'il n'y a pas eu d'erreur, et on l'enregistre dans un fichier (qu'on appellera private-key-test.pem par exemple).

Ensuite, on veut analyser le contenu de la clé SSH. On sait qu'une clé SSH est composée de plusieurs éléments :

- q : un nombre premier
- p : un nombre premier
- $N : p \times q$
- e : l'exposant de la fonction contenue dans la clé
- d : un nombre
- d_q : le reste de $d / (q - 1)$
- d_p : le reste de $d / (p - 1)$
- q_{inv} : le reste de q^{-1} / p

On remarque donc qu'ici, les inconnues sont q , p et e . En effet, si on a p ou q , on peut retrouver d . Si on a p et q , on peut retrouver tous les nombres sauf e .

Hors, e est en général égal à 65537 par convention. Donc, il nous faut trouver q et p .

Pour se faire, on va transformer la clé SSH que nous avons en hexadécimal. On utilisera un script python permettant de le faire :

```

decode.py
1  import base64
2
3  with open('private-key-test.pem') as f:
4      tmp = f.read()
5
6  tmp = tmp.encode('ascii')
7  strbytes = base64.b64decode(tmp)
8  for i in range(1, len(strbytes) + 1):
9      end=' '
10     if i%48 == 0:
11         end='\n'
12     if len(hex(strbytes[i-1])) == 3:
13         print('0' + hex(strbytes[i-1])[2:], end=end)
14     else:
15         print(hex(strbytes[i-1])[2:], end=end)
16

```

Puis, pour sauvegarder la sortie, on pourra faire

```

(kali@kali)-[~/Documents/DMCrypto]
$ python3 decode.py > hexkey

```

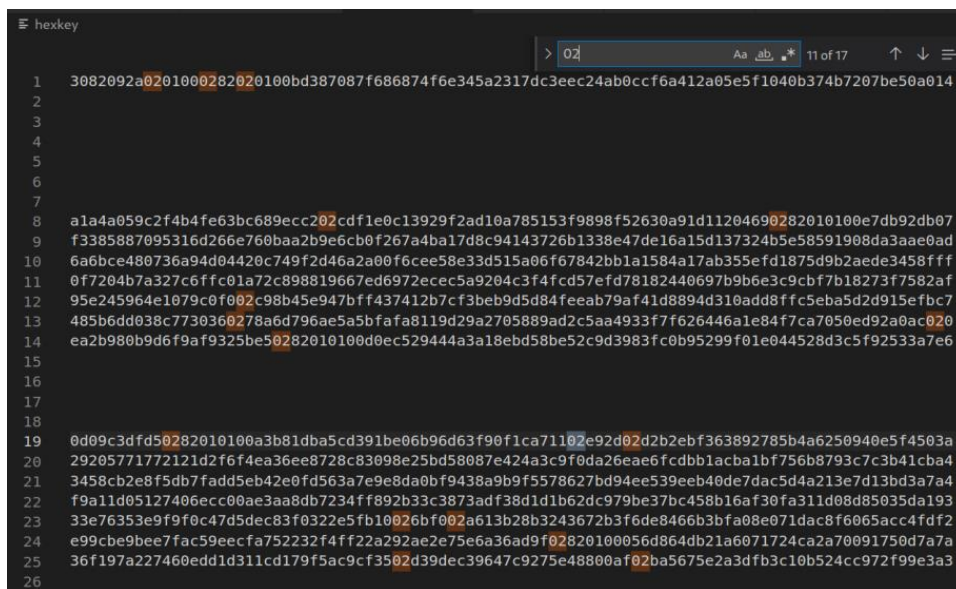
Et on retrouve dans le fichier hexkey l'héxadécimal de notre clé SSH

```

hexkey
1  3082092a201000282020100bd387087f686874f6e345a2317dc3eec24ab0ccf6a412a05e5f1040b374b7207be50a014
2
3
4
5
6
7
8  a1a4a059c2f4b4fe63bc689ecc202cdf1e0c13929f2ad10a785153f9898f52630a91d11204690282010100e7db92db07
9  f3385887095316d266e760baa2b9e6cb0f267a4ba17d8c94143726b1338e47de16a15d137324b5e58591908da3aae0ad
10 6a6bce480736a94d04420c749f2d46a2a00f6cee58e33d515a06f67842bb1a1584a17ab355efd1875d9b2aede3458fff
11 0f7204b7a327c6ffc01a72c898819667ed6972ecce5a9204c3f4fcd57efd78182440697b9b6e3c9cbf7b18273f7582af
12 95e245964e1079c0f002c98b45e947bff437412b7cf3beb9d5d84feeab79af41d8894d310add8ffc5eba5d2d915efbc7
13 485b6dd038c7730360278a6d796ae5a5bfafa8119d29a2705889ad2c5aa49337f7f626446a1e84f7ca7050ed92a0ac020
14 ea2b980b9d6f9af9325be50282010100d0ec529444a3a18ebd58be52c9d3983fc0b95299f01e044528d3c5f92533a7e6
15
16
17
18
19 0d09c3dfd50282010100a3b81dba5cd391be06b96d63f90f1ca71102e92d02d2b2ebf363892785b4a6250940e5f4503a
20 29205771772121d2f6f4ea36ee8728c83098e25bd58087e424a3c9f0da26ae6fcdabb1acba1bf756b8793c7c3b41cba4
21 3458cb2e8f5db7fadd5eb42e0fd563a7e9e8da0bf9438a9b9f5578627bd94ee539eeb40de7dac5d4a213e7d13bd3a7a4
22 f9a11d05127406ecc00ae3aa8db7234ff892b33c3873adf38d1db62dc979be37bc458b16af30fa311d08d85035da193
23 33e76353e9f9f0c47d5dec83f0322e5fb10026bf002a613b28b3243672b3f6de8466b3bfa08e071dac8f6065acc4fd2f
24 e99cbe9bee7fac59eeafa752232f4ff22a292ae2e75e6a36ad9f02820100056d864db21a6071724ca2a70091750d7a7a
25 36f197a227460edd1d311cd179f5ac9cf3502d39dec39647c9275e48800af02ba5675e2a3dfb3c10b524cc972f99e3a3
26

```

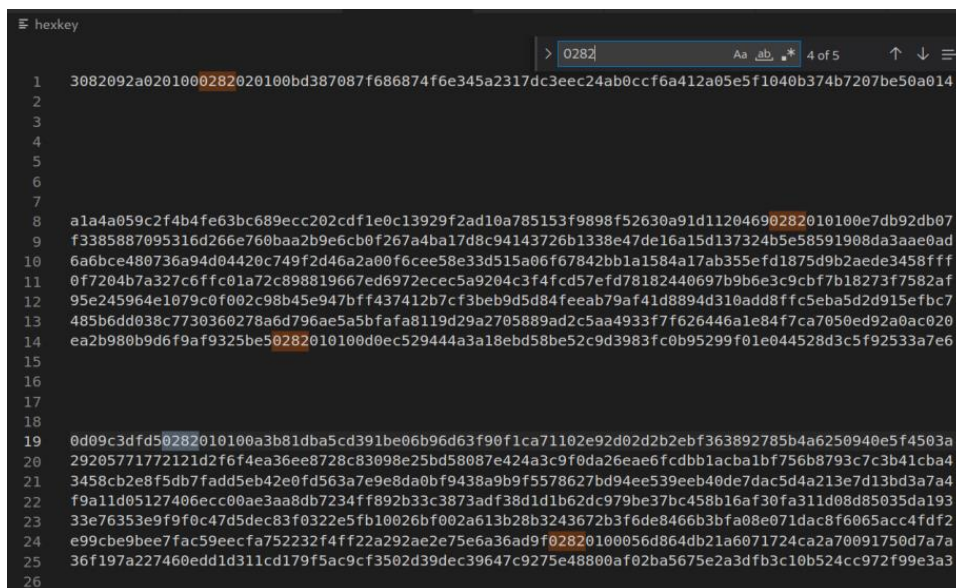
On sait qu'en hexadécimal, les nombres entiers sont stockés de manière caractéristiques, car ils sont composés de 4 chiffres : 02 puis deux chiffres pour indiquer sur combien de bits ils sont codés. Si on cherche 02 sur notre fichier, on a :



```
hexkey
> 02 11 of 17
1 3082092a0201000282020100bd387087f686874f6e345a2317dc3eec24ab0ccf6a412a05e5f1040b374b7207be50a014
2
3
4
5
6
7
8 a1a4a059c2f4b4fe63bc689ecc202cdf1e0c13929f2ad10a785153f9898f52630a91d11204690282010100e7db92db07
9 f3385887095316d266e760baa2b9e6cb0f267a4ba17d8c94143726b1338e47de16a15d137324b5e58591908da3aae0ad
10 6a6bce480736a94d04420c749f2d46a2a00f6cee58e33d515a06f67842bb1a1584a17ab355efd1875d9b2aede3458fff
11 0ff7204b7a327c6ffc01a72c898819667ed6972ecec5a9204c3f4fcd57efd78182440697b9b6e3c9cbf7b18273f7582af
12 95e245964e1079c0f002c98b45e947bff437412b7c3f3beb9d5d84feeab79af41d8894d310add8ffc5eba5d2d915efbc7
13 485b6dd038c7730360278a6d796ae5a5bfafa8119d29a2705889ad2c5aa4933f7f626446a1e84f7ca7050ed92a0ac020
14 ea2b980b9d6f9af9325be50282010100d0ec529444a3a18ebd58be52c9d3983fc0b95299f01e044528d3c5f92533a7e6
15
16
17
18
19 0d09c3dfd50282010100a3b81dba5cd391be06b96d63f90f1ca71102e92d02d2b2ebf363892785b4a6250940e5f4503a
20 29205771772121d2f6f4ea36ee8728c83098e25bd58087e424a3c9f0da26eae6fcdabb1acba1bf756b8793c7c3b41cba4
21 3458cb2e8f5db7fadd5eb42e0fd563a7e9e8da0bf9438a9b9f5578627bd94ee539eeb40de7dac5d4a213e7d13bd3a7a4
22 f9a11d05127406ecc00ae3aa8db7234ff892b33c3873adf38d1d1b62dc979be37bc458b16af30fa311d08d85035da193
23 33e76353e9f9f0c47d5dec83f0322e5fb10026bf002a613b28b3243672b3f6de8466b3bfa08e071dac8f6065acc4dfd2
24 e99cbe9bee7fac59eeca752232f4ff22a292ae275e6a36ad9f02820100056d864db21a6071724ca2a70091750d7a7a
25 36f197a227460edd1d311cd179f5ac9cf3502d39dc39647c9275e48800af02ba5675e2a3dfb3c10b524cc972f99e3a3
26
```

On remarque que le fichier commence par “0201” qui est un petit nombre, ça ne peut donc pas être une des inconnues que l’on cherche.

Le deuxième “02” que l’on trouve est “0282”. Cela indique un grand nombre, et peut être une de nos inconnues. Cherchons maintenant 0282 :



```
hexkey
> 0282 4 of 5
1 3082092a0201000282020100bd387087f686874f6e345a2317dc3eec24ab0ccf6a412a05e5f1040b374b7207be50a014
2
3
4
5
6
7
8 a1a4a059c2f4b4fe63bc689ecc202cdf1e0c13929f2ad10a785153f9898f52630a91d11204690282010100e7db92db07
9 f3385887095316d266e760baa2b9e6cb0f267a4ba17d8c94143726b1338e47de16a15d137324b5e58591908da3aae0ad
10 6a6bce480736a94d04420c749f2d46a2a00f6cee58e33d515a06f67842bb1a1584a17ab355efd1875d9b2aede3458fff
11 0ff7204b7a327c6ffc01a72c898819667ed6972ecec5a9204c3f4fcd57efd78182440697b9b6e3c9cbf7b18273f7582af
12 95e245964e1079c0f002c98b45e947bff437412b7c3f3beb9d5d84feeab79af41d8894d310add8ffc5eba5d2d915efbc7
13 485b6dd038c7730360278a6d796ae5a5bfafa8119d29a2705889ad2c5aa4933f7f626446a1e84f7ca7050ed92a0ac020
14 ea2b980b9d6f9af9325be50282010100d0ec529444a3a18ebd58be52c9d3983fc0b95299f01e044528d3c5f92533a7e6
15
16
17
18
19 0d09c3dfd50282010100a3b81dba5cd391be06b96d63f90f1ca71102e92d02d2b2ebf363892785b4a6250940e5f4503a
20 29205771772121d2f6f4ea36ee8728c83098e25bd58087e424a3c9f0da26eae6fcdabb1acba1bf756b8793c7c3b41cba4
21 3458cb2e8f5db7fadd5eb42e0fd563a7e9e8da0bf9438a9b9f5578627bd94ee539eeb40de7dac5d4a213e7d13bd3a7a4
22 f9a11d05127406ecc00ae3aa8db7234ff892b33c3873adf38d1d1b62dc979be37bc458b16af30fa311d08d85035da193
23 33e76353e9f9f0c47d5dec83f0322e5fb10026bf002a613b28b3243672b3f6de8466b3bfa08e071dac8f6065acc4dfd2
24 e99cbe9bee7fac59eeca752232f4ff22a292ae275e6a36ad9f02820100056d864db21a6071724ca2a70091750d7a7a
25 36f197a227460edd1d311cd179f5ac9cf3502d39dc39647c9275e48800af02ba5675e2a3dfb3c10b524cc972f99e3a3
26
```

On remarque donc ici que 5 nombres sont renseignés, dont 2 sont entièrement visibles. Mais on ne sait pas encore à quoi correspondent ces nombres.

[illegible]

```
1 3082092a0201000282020100bd38707f586874f6e345a2317dc3eec24ab0cc6fa412a05e5f1040b374b7207be50a014
2
3 N
4
5
6
7
8 a1a4a059c2f4b4fe63bc689ecc202cdf1e0c13929f2ad10a785153f9898f52630a91d11204690282010100e7db92db07
9 f3385887095316d266e760baa2b9e6cb0f267a4ba17d8c94143726b1338e47de16a15d137324b5e58591908da3aae0ad
10 6a6bce480736a9d40420c749f2d46a2a00f6cee58c33d515a06f67842bb1a1584a17ab355efd1875d9b2aede3458fff
11 0f7204b7a327c6ff01a72c98819667ed6972eced09204c3f4fcd57efd78182440697b9b6e3c9cbf7b18273f7582af
12 95e245964e1079c0f002c98b45e947bff437412b70f3be9b9d5d84feea479af41d8894d310add8f3c5eba5d2d915efbc7
13 485b6dd0c38c7730360278a6d796ae5a5bfa4ba119d29a2705889ad2c5aa4933f7f626446a1e847fca7050ed9a2a0c020
14 ea2b980b9d6f9af9325be50282010100d0ec52944a3a18eb58be52c9d3983fc0b95299f01e044528d3c5f92533a7e6
15
16
17
18
19 00ae30fd50282010100a3b81dba5cd391be06b96d63f90f1ca71102e92d02d2b2ebf363892785b4a6e250940e5f4503a
20 29205771772121d2f6f4ea36ee8728c83898e25bd58087e424a3c9f0da26aeae6fcdabb1acba1bf756b8793c73b41cba4
21 3458cb2e8f5d7fadd5eb42e0fd6563a7e4e6da0bbf9438a9bf5578627bd94ee539eb40de7dac5d4a213e7d13bd3a7a4
22 f9a11d05127406ecc00ae3aa80f7234ff892b33c3873adf38d1d1b62dc979bfe37c3458b16af30fa311d08d850353da193
23 33b76353e9f9f0c47d5dec83f0322e5fb1b0926bf002a613b2b83243672b3f6de8466b3ffa80e071dacbf6665acc4fd42
24 e99cbe9bee7fac59eecfa752232f4ff22a292ae2e756a36ad9f029f0100056d864db21a6071724ca2a70091750d7a7a
25 36f197a227460edd1311cd179f5ac9cf3502d39dec39647c9275e48380a1010005e2a3d3fb3c10b524cc972f99e3a3
26
```

Il nous manque donc un nombre essentiel pour régénérer la clé privée : q .

Pour retrouver q , on va faire appel aux maths et trouver une équation à partir des deux nombres connus à notre disposition.

On regarde donc d_q .

On sait que $d_q = \text{reste de } d / (q - 1)$

Donc $d_q = d - K_1 * (q - 1)$ où K_1 est un nombre.

On connaît également e et on le multiplie de part et d'autre de l'équation donc

$$e * d_q = e (d - K_1 * (q - 1))$$

$$= e * d - e * K_1 * (q - 1)$$

$$\text{Donc } e * d = 1 * (p - 1) (q - 1)$$

$$\text{Donc } e * d = K_2 * (p - 1) (q - 1) + 1 \text{ avec } K_2 \text{ un nombre}$$

$$\text{Donc } e * d_q = K_2 * (p - 1) (q - 1) + 1 - e * K_1 * (q - 1)$$

$$= (q - 1) (K_2 * (p - 1) - e * K_1) + 1$$

On peut donc maintenant définir que K_3 est un nombre où

$$K_3 = (K_2 * (p - 1) - e * K_1)$$

$$\text{Donc } e * d_q = (q - 1) * K_3 + 1$$

On a donc l'équation de q :

$$q = (e * d_q - 1) / K_3 + 1$$

On veut maintenant trouver K_3 pour n'avoir qu'une seule inconnue. Pour se faire, on va tout d'abord encadrer K_3 puis nous allons ensuite le bruteforce.

On sait déjà que

$0 \leq (q - 1) * K_3 + 1 \leq e * (q - 1)$ car $e * d_q = (q - 1) * K_3 + 1$ et d_q ne peut donc pas dépasser $(q - 1)$, car d_q est un reste.

Donc

$$-1 \leq (q - 1) * K_3 < e * (q - 1) \text{ car } (q - 1) * K_3 < (q - 1) * K_3 + 1$$

Et pour $q > 1$

$$-1 / (p - 1) \leq K_3 < e$$

Et donc pour $q > 2$

$$0 \leq K_3 < e$$

Maintenant que nous avons trouvé l'équation, et encadré K_3 , on va bruteforce K_3 grâce à ce script python

```
bruteforce.py
1  from sympy import isprime
2
3  p = 0x00e7db92db07f3385887095316d266e760baa2b9e6cb0f267a4ba17d8c9
4  dq = 0x00a3b81dba5cd391be06b96d63f90f1ca71102e92d02d2b2ebf3638927
5  e = 65537
6
7  for k3 in range(1, e):
8      if (dq*e%k3 == 1):
9          q = (dq * e - 1) // k3 + 1
10         if (isprime(q)):
11             print(q)
12
```

On obtient donc q :

```
(kali㉿kali)-[~/Documents/DMCrypto]
$ python bruteforce.py
26374102437938552598977775622060483564941500314335674804443257510
66445076586481679113841644050194447873723787162461543012337002755
00792401391613297064940589286914213118584678411153945438814897268
59388606527864266014327364474064223329857483595456677811551699807
04116724508610553934484006583
```

Il ne nous suffit plus que d'utiliser la library PyCryptoDome pour générer la clé SSH :

```

recover.py
1  from Crypto.PublicKey import RSA
2
3  q = 2637410243793855259897777562206048356494150031433567480444325
4  p = 0x00e7db92db07f3385887095316d266e760baa2b9e6cb0f267a4ba17d8c9
5  dq = 0x00a3b81dba5cd391be06b96d63f90f1ca71102e92d02d2b2ebf3638927
6  e = 65537
7
8
9  N = p*q
10 d = pow(e, -1, (p-1)*(q-1))
11
12
13 key = RSA.construct((N,e,d,p,q))
14 pem = key.exportKey('PEM')
15 print(pem.decode())

```

Puis sauvegarder la clé complète dans le fichier privatekey.pem

```

(kali@kali)-[~/Documents/DMCrypto]
$ python recover.py | > privatekey.pem

```

Changer les permissions de la clé privée SSH pour pouvoir l'utiliser plus tard

```

(kali@kali)-[~/Documents/DMCrypto]
$ chmod go= privatekey.pem

```

Et enfin se connecter en SSH au serveur

```

(kali@kali)-[~/Documents/DMCrypto]
$ ssh -i privatekey.pem admin@ctf.isen-cyber.ovh -p 6022

```

Plus qu'à recopier le flag !

```
(kali㉿kali)-[~/Documents/DMCrypto]
$ ssh -i privatekey.pem admin@ctf.isen-cyber.ovh -p 6022
SSH_BURNED
ISEN{SSH_Keyp4ir_C3rtific4te_Burn3d}
This account is not available
```

Write-Up de Thomas SEIGNOUR au DM SSHBurned pour M. Aymeric Deliencourt (et un peu pour avoir les 50 points sur le scoreboard du site...)