LE CHALLENGE DU LOGO ANSSI

MISC n° 073 | mai 2014 | Pierre Bienaimé

Sécurité

Le 3 février 2012, l'agence nationale de la sécurité des systèmes d'information (ANSSI) a publié son nouveau logo et a eu la bonne idée d'y cacher un challenge de sécurité informatique. Des morceaux de solutions furent rapidement trouvés et rendus publics, mais ce challenge a su se faire désirer puisque deux ans plus tard, j'ai finalement été le premier à en voir le bout. Dans cet article, je vous présente ma solution, le raisonnement adopté, les galères rencontrées ainsi qu'un petit bonus, le tout sur un ton assez peu formel.

1. CHRONOLOGIE

Le nouveau logo de l'ANSSI est dévoilé en avant-première le 7 octobre 2011 [1] par Patrick Pailloux lors des Assises de la sécurité. Cependant, ce n'est que le 3 février 2012 que le logo est officiellement publié [2] sur le site web de l'agence. Il est accompagné d'une phrase pour le moins énigmatique : *« Les curieux apprécieront les fonds d'écran qui ont également été réalisés »*. Les jours suivants, des internautes repèrent que ces fonds d'écran renferment un probable challenge de sécurité et le buzz se répand rapidement sur les réseaux sociaux. Certains chercheurs décident de partager leurs découvertes et des bribes de solutions fleurissent un peu partout. Le site **anssi.santo.fr** est alors créé, agrège les avancées connues et devient un site de référence pour tous ceux qui enquêtent sur le challenge. Mais deux des épreuves s'avèrent très complexes, les recherches piétinent et l'engouement de la communauté pour le challenge s'estompe progressivement. À la fin du mois de février 2012, presque plus personne n'en parle (du moins publiquement). De mon côté, j'y consacre à cette époque une grosse semaine, j'avance un peu sur chaque épreuve, mais une fois à court d'idées et de courage, j'abandonne.

Le 3 octobre 2012, une fois de plus aux Assises de la sécurité, Patrick Pailloux annonce à la fin de son discours [3] que *« le challenge glissé dans notre logo sur le site n'est toujours pas résolu »*. À ma connaissance, c'est la seule communication officielle de la part de l'ANSSI qui reconnaît que oui, il y a bien un challenge caché dans leur logo. L'année suivante, lors de son discours aux Assises 2013, le sujet du challenge n'est pas évoqué.

Au début du mois de décembre 2013, je retombe sur ce challenge un peu par hasard. Je constate qu'il n'est toujours pas résolu et qu'aucune avancée significative n'a été dévoilée publiquement depuis tout ce temps. Je me replonge dans les épreuves pour comprendre pourquoi j'avais abandonné et j'ai une sorte de révélation ; une idée désespérée qui – contre toute attente – fonctionne et me débloque. J'apprends ensuite que deux chercheurs planchent actuellement sur le challenge et sont bien décidés à lui faire la peau. Cela créé une petite rivalité très bénéfique, car elle permet de trouver la motivation nécessaire. Après quelques péripéties, je termine finalement le challenge le 13 janvier 2014.

2. PREMIERS PAS

2.1 CE QUI SAUTE AUX YEUX

Figure 1 : Image de départ du challenge.

Le challenge est une image (figure 1) au format PNG qui représente le logo de l'ANSSI sur fond noir. En l'examinant de plus près (figure 2), un œil aguerri ne manque pas de remarquer que le cercle intérieur est agrémenté de chaînes de caractères suspectes, séparées en trois blocs.

fig2

Figure 2 : Zoom sur le cercle intérieur.

Le premier bloc est composé de caractères hexadécimaux.

```
A125894294F6A08D
FDC21B055809130B
D873AD692D563156
F0450B4A33EB5315
E99C64710CD78CDB
AD2EEEF2E168A0EA
8F2320C340CF7BBF
52CC2D94BFA89E01
613094E58C727F72
3ACE254275121653
EE46D39D1103A044
8298EDE384A73E7E
```

Ces 96 octets ne constituent pas un format de fichier connu et ne contiennent pas de texte. Pour comprendre de quoi il s'agit, il suffit de compter chaque octet. Comme ils sont presque uniformément distribués, on conclut que les données sont certainement chiffrées et que l'on ne peut rien en faire pour l'instant. Le deuxième bloc est constitué des trois mots :

```
AUTH : DE9C9C55 : PCA
```

AUTH peut faire penser à *authentication* ou à *author*, mais le reste est assez mystérieux. Je n'ai appris que récemment que le logo est l'œuvre d'un certain Pierre Capillon ; la chaîne PCA correspond donc à ses initiales. Quant à DE9C9C55, cette chaîne permit à quelqu'un **[4]** de remonter (temporairement) jusqu'au CV de Pierre Capillon. J'ai fini par supposer qu'il pouvait s'agir d'un hash. Comme il fait 4 octets, on pense à CRC32. Après quelques essais, le mystère a pu être éclairci. Ça ne sert à rien, mais ça fait quand même plaisir !

```
>>> crc32("Pierre Capillon")
'\xde\x9c\x9cU'
```

Le troisième bloc est celui qui permet d'avancer.

```
TGUgc291cmlyZSBkZSBsYSBKb2NvbmRlIGNhY2hhaXQgYmllbiBkZXMgbXlzdOhyZXMuLi4K
```

On reconnaît un encodage base64. Une fois la chaîne décodée, cela produit la phrase *« Le sourire de la Joconde cachait bien des mystères... »*. Le premier réflexe est de la rechercher sur Google. Elle ne renvoie à rien de particulier, mais *Google Suggest* complète la fin de la phrase, ce qui montre qu'une quantité non négligeable de personnes se sont intéressées au challenge. En se documentant un peu plus sur la Joconde, on apprend que la technique utilisée par Léonard de Vinci pour créer l'effet vaporeux de son sourire s'appelle le *sfumato*. Le principe est de superposer plusieurs fines couches de peinture de différentes couleurs. On peut alors imaginer qu'en jouant avec les couleurs du fond d'écran, il sera possible de révéler un contenu secret.

2.2 DÉCOUVERTE DES COUCHES

Il existe de nombreux moyens permettant de dévoiler les couches incrustées dans l'image. Pour ma part, j'ai ouvert le fichier PNG avec GIMP, puis j'ai poussé la luminosité et le contraste au maximum (figure 3). Comme par magie, des caractères rouges, verts et bleus apparaissent. Ce sont tous des caractères hexadécimaux, à l'exception d'un bloc encodé en base64 qui porte la mention BEGIN PUBLIC KEY. Il est possible de distinguer quatre parties : un bloc vert en haut à droite, un double-bloc vert à gauche, un bloc bleu à gauche et un bloc rouge qui recouvre toute la surface de l'image.

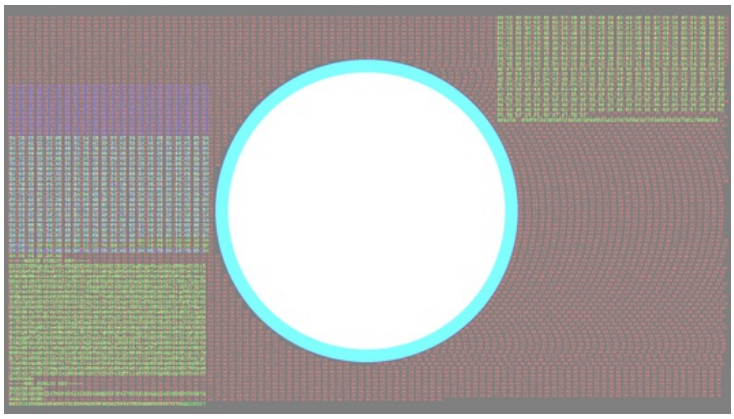

Figure 3 : Découverte des couches.

La couche bleu clair que l'on aperçoit sur la gauche n'existe pas réellement. Il s'agit simplement de caractères de la couche verte et de la couche bleue qui sont superposés. Pourquoi est-ce que ces couches sont révélées lorsque l'on joue avec le contraste ? La réponse est dans les pixels.

```
>>> from PIL import Image
>>> img = list(Image.open("anssi.png").getdata())
>>> img[123456:123464]
[(0, 0, 0), (2, 0, 0), (2, 0, 0), (2, 0, 0), (2, 0, 0), (2, 0, 0), (0, 0, 0), (0, 0, 0)]
```

Ici, les pixels sont codés en RGB (rouge, vert, bleu) sur trois octets. Le pixel (0, 0, 0) est noir tandis que (255, 255, 255) est blanc. Dans l'exemple ci-dessus, en affichant quelques pixels de l'image au hasard, on remarque que certains ont comme valeur (2, 0, 0). Visuellement, ce pixel est noir, mais pourtant il contient une infime touche de rouge. En augmentant le contraste, les écarts de couleur sont accentués et finissent par devenir visibles à l'œil nu. Pour faire les choses plus proprement, j'ai ensuite utilisé un script très pratique pour les challenges de stéganographie. L'idée (qui provient d'ici **[5]**) est de recréer une image en ne conservant que les bits de poids faible des composantes de chaque pixel. Cela permet de constater visuellement qu'une image contient de la stéganographie. Dans le cas du logo, l'image générée par ce script (figure 4) à partir des deux bits de poids faible dévoile directement les couches.

```
>>> lsb_highlight("anssi.png", 2)
```

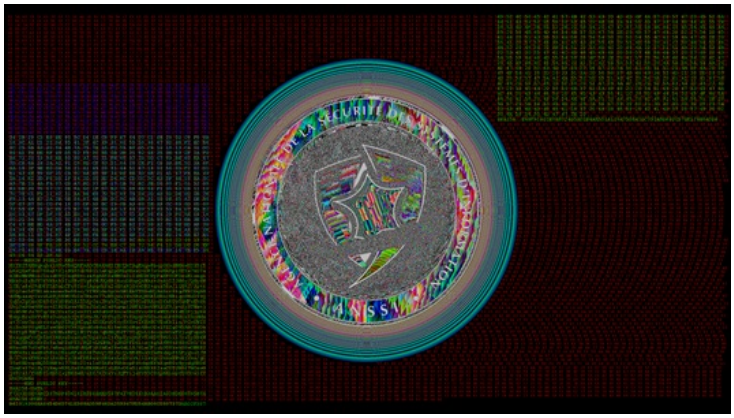

Figure 4 : Image LSB.

L'avantage de cette représentation (outre les couleurs particulièrement chatoyantes) est qu'elle révèle des contours qui sont presque invisibles dans le fond d'écran d'origine. En examinant attentivement le cercle intérieur, deux citations se dessinent : *« La persévérance est la noblesse de l'obstination »* (Adrien Decourcelle) et *« Les moments les plus difficiles sont ceux qui donnent le plus de satisfaction »* (Claude Lelouch). Nous voici avertis que le challenge risque d'être difficile !

2.3 OCR

L'étape suivante consiste à utiliser un logiciel de reconnaissance de caractères (OCR) afin d'extraire les données de chaque couche. Le but de cet OCR est simplement de gagner du temps, car il est possible (mais horriblement rébarbatif) de réaliser cette extraction à la main. Pour cette étape, les concepteurs du challenge sont plutôt cléments puisqu'ils fournissent les sha256 de chaque bloc, ce qui permet de vérifier que l'OCR fait correctement son travail. Après avoir isolé les couches, il convient d'appliquer quelques pré-traitements pour augmenter le taux de réussite de l'OCR (conversion en noir et blanc, découpage propre des zones de texte, etc.).

Pour les caractères hexadécimaux en vert et en bleu, un OCR standard (par exemple `gocr` ou `tesseract-ocr`) donne de bons résultats. Pour la couche rouge, les données étant séparées par un gros trou circulaire, il faut ruser. La solution du bricoleur consiste à appliquer l'OCR par petits morceaux, celle du courageux est de coder un OCR maison qui saura s'adapter à la situation. J'ai opté pour la troisième option – celle du paresseux – qui consiste à attendre que la solution fuite. La couche rouge a pu être extraite grâce au code d'un ami et le base64 a été chapardé sur **anssi.santo.fr**. Voilà, les quatre parties sont à présent sous format numérique et chacune d'entre elles semble contenir une énigme. L'échauffement est terminé, rentrons dans le vif du challenge.

3. FÉVRIER 2012

Pour vous exposer ma solution, il a fallu trouver un compromis entre l'ordre théorique idéal dans lequel le challenge aurait pu être résolu et ce qui s'est réellement passé, à savoir un ordre complètement chaotique, des fausses pistes, des morceaux résolus rapidement et d'autres deux ans plus tard. J'ai opté pour un découpage temporel, en présentant pour chacune des deux périodes l'état d'avancement des quatre parties.

3.1 VERT (DROITE)

La première énigme est le texte ci-dessous :

```
IRLVEBAWKBSC CMOJW WG ERZDQCSRUX SE KT4DP3G9T@DAU.YSMO.UR. XT GGGXTNF ETEFDX XNEMDTKWSGX. EWGI R'T EAD QCUSJX TTP
PTULAYJGE. UA WWFFAECIUI UMX A'PKTPFKW HT NTQ SML IJ WQQJ TSK SE UGALMHEIH XWKTFW WI CZUYJFMUTXXOY. AW VXFJT LG EIKLEVE
AQBTPSBX DZVZTJ MGI PCMYXWVW TPREQQ KYJ XGOTA - #1: PX7MHI4RM! - WM SWGZBJURPUQCL VX AA AGXLI XWI PV RGYJL. PZCD D'MG
SED YTKWSZIH, TQH GSEGJLD EDFX EECCPA, UIDT EOFZDPAX IVTNOZQ DE LIBATVQ. PW VTSEM (#VWY P100000) T CENMEHAXW FXEY BAMW
WI RPNXTPMGG.
```

Première constatation, les mots ne veulent rien dire, mais la ponctuation est située à des endroits cohérents, ce qui laisse supposer que seules les lettres ont été chiffrées. Le challenge étant organisé par l'ANSSI, il est probable qu'une fois déchiffré, le texte sera en langue française. Pour comprendre quel type de chiffrement a pu être utilisé, le bon réflexe est de faire une analyse fréquentielle, c'est-à-dire compter les lettres puis comparer le résultat avec la fréquence moyenne de la langue française.

```
>>> char_frequency(green)
Counter({'t': 7.775, 'e': 7.239, 'w': 6.434, 'x': 6.434, 'g': 5.362, 'a': 5.094, 'p': 4.826, 'i': 4.558, 'm': 4.558,
'd': 4.021, 's': 4.021, 'u': 3.753, 'c': 3.217, 'j': 3.217, 'q': 3.217, 'f': 2.949, 'l': 2.949, 'v': 2.949, 'k': 2.681,
'r': 2.681, 'y': 2.681, 'z': 2.413, 'b': 1.877, 'h': 1.877, 'o': 1.609, 'n': 1.609})
>>> FRENCH_FREQUENCY
Counter({'e': 17.124, 'a': 8.122, 's': 7.948, 'i': 7.58, 't': 7.244, 'n': 7.095, 'r': 6.553, 'u': 6.369, 'l': 5.456,
'o': 5.387, 'd': 3.669, 'c': 3.345, 'p': 3.021, 'm': 2.968, 'v': 1.628, 'q': 1.362, 'f': 1.066, 'b': 0.901, 'g': 0.866,
'h': 0.737, 'j': 0.545, 'x': 0.387, 'y': 0.308, 'z': 0.136, 'w': 0.114, 'k': 0.049})
```

En français, il y a en moyenne 17% de E, 8% de A et 0.1% de W. Si la fréquence de notre texte est proche de ceci, c'est que le chiffrement est une permutation. Si l'on retrouve la même courbe de répartition, mais avec des lettres différentes, nous sommes en présence d'un chiffrement par substitution monoalphabétique. Ici, nous ne sommes dans aucun de ces deux cas. On constate que la fréquence est lissée, que les extrêmes ont disparu. C'est l'effet d'un chiffrement par substitution polyalphabétique. Il en existe une grande quantité, mais le plus connu est le chiffre de Vigenère et c'est également l'un des moins complexes, c'est donc par là qu'il faut commencer les investigations.

Le principe du chiffre de Vigenère est de choisir une clé (un mot de passe) qu'on utilise de manière cyclique. Pour chaque lettre du texte clair, on réalise une addition modulo 26 avec une lettre de la clé en considérant que A=0 et Z=25. Le chiffre de Vigenère est vulnérable aux attaques par analyse fréquentielle, mais une manière bien plus triviale de le casser est de deviner un morceau du texte clair afin de recalculer directement la clé. Le texte contient justement une chaîne facilement devinable. KT4DP3G9T@DAU.YSMO.UR ressemble à s'y méprendre à une adresse e-mail. Une rapide recherche nous apprend que les adresses e-

mail des agents de l'ANSSI utilisent le domaine **ssi.gouv.fr**, ce qui nous permet de trouver 9 lettres de la clé.

```
>>> vigenere_plaintext("DAU.YSMO.UR", "SSI.GOUV.FR")
'LIMSESTPA'
```

Coup de chance, LIMSESTPA ressemble fortement à un mot que l'on connaît et qui fait (plus ou moins) partie du vocabulaire de la cryptologie : PALIMPSESTE. Ce mot désigne un manuscrit sur lequel on a fait disparaître les inscriptions pour pouvoir y écrire de nouveau. Tenter de déchiffrer tout le texte avec cette clé génère un premier mot cohérent – *transmission* –, mais la suite n'est pas correctement déchiffrée. Je finis par comprendre qu'il s'agit d'une variante maison du chiffre de Vigenère. Normalement, les caractères qui ne sont pas des lettres sont ignorés alors que dans cette variante, une lettre de la clé est consommée à la place. En modifiant la fonction de déchiffrement pour prendre en compte cette petite personnalisation, le texte clair apparaît.

```
>>> vigenere_decrypt(txt, "PALIMPSESTE", False)
"TRANSMISSION RECUE EN PROVENANCE DE CH4LL3N9E@SSI.GOUV.FR. LE CONTENU SEMBLE INTERESSANT. TOUT N'A PAS ENCORE ETE
DECHIFFRE. IL SEMBLERAIT QUE L'ECHANGE DE CLE AIT EU LIEU PAR DE MULTIPLES MOYENS DE COMMUNICATION. LE DEBUT DU MESSAGE
SEMBLAIT DONNER UNE PREMIERE PARTIE SUR TROIS - #1: AX7EVT4NU! - LE DECHIFFREMENT DE LA SUITE EST EN COURS. POUR L'UN
DES MESSAGES, LES CALCULS SONT LANCES, CELA POURRAIT PRENDRE LA SEMAINE. LE RESTE (#REF A100000) A NECESSITE BIEN PLUS
DE REFLEXION."
```

Ce message, je ne l'ai toujours pas réellement compris. Est-ce qu'il m'est adressé ? Ou est-ce une communication que je suis censé avoir interceptée ? Dans tous les cas, nous avons maintenant une adresse e-mail de contact ainsi que quelques indices. Le premier secret est **AX7EVT4NU!** et on sait qu'il y en aura trois. Le deuxième demandera une semaine de calcul et le troisième beaucoup de réflexion. Cette première énigme étant résolue, on peut passer à la suite.

3.2 BLEU

Une fois décodée, la partie bleue est également un texte qui semble chiffré.

```
rsutsrmevoluug rniat Comn defarrerchadeial,n g
Faést po lrthoeuares h sès deuesiuta,
in mieroe taia PDegdpitoesta
Pale osMrut, anienrrue'uecs e, s ivdrhéêvllroï rt ta enue.
br eun a
Illulefcieuea ir astqméQol
pamûnqt uérblen Cirgtux ast s,nto daes sess vealnit E minoialntlesancizé esls inurt l a
inentr s
bouteuienixtsds érAnne.
oquir, myieoos e esntmx dnduccialdepé deanninseeCr lmaeséps,iqha
L'ed phohu nresosra m dcezunchs iur eoe ltpaaTrdlemepieser t
Equis'unltanréeavà om draoi dOul nch g mnt',;
denc ba ca#3hepes-: -ltd ellsalés I aigaen,
esravdeoe mourr re n Ocde entn rgienoétsaln ciefé
Dnds oinou
le"Le.u l el
usv cére l'ant- JMarson ia érdeediHqs",a
eos
```

Comme précédemment, on réalise une analyse fréquentielle :

```
>>> char_frequency(txt)
Counter({'e': 16.418, 'a': 8.396, 's': 8.209, 'n': 8.022, 'r': 7.276, 'i': 6.903, 'l': 6.157, 't': 5.97, 'u': 5.597,
'o': 5.41, 'd': 4.664, 'c': 3.172, 'm': 2.985, 'h': 2.052, 'p': 2.052, 'g': 1.306, 'q': 1.306, 'v': 1.306, 'b': 0.746,
'f': 0.746, 'x': 0.56, 'z': 0.373, 'j': 0.187, 'y': 0.187})
```

Cette fois, la fréquence est très proche de celle de la langue française. Il s'agit donc d'une « simple » permutation, ce qui signifie qu'en rangeant les lettres dans le bon ordre, on obtiendra un texte en français. Mais comment retrouver le bon ordre ? Les algorithmes triviaux (lire un caractère sur deux, un sur trois, un par ligne, etc.) ne fonctionnent pas. Les retours à la ligne sont anarchiques, ce qui montre que la permutation ne s'applique pas uniquement aux lettres. Détail intéressant, le texte contient les caractères **- #3: -**. En effet, en remarquant que le secret 1 est écrit sous la forme **- #1: AX7EVT4NU! -**, on comprend qu'une fois la permutation inversée, le texte révélera le secret 3. Tous ces caractères sont sur la même ligne et ça n'est pas une coïncidence : les permutations sont locales ; un caractère chiffré ne se retrouvera pas trop éloigné de sa position d'origine.

J'entreprends alors de reconstituer des mots en mélangeant les caractères à l'échelle d'une ligne. Le texte contient certaines lettres qui sont assez rares en français (û, ê, ï…). Mais après quelques longues prises de tête, je ne trouve rien de concluant et je finis par me résigner.

3.3 ROUGE

La partie rouge est une archive bz2. Une fois décompressée, elle nous donne 256 Kio de données non identifiées. Comme d'habitude, pour se faire une idée de ce que l'on manipule, on compte les octets.

```
>>> char_frequency(red, False)
Counter({'\x00': 95.582, '\xbd': 0.403, '\xff': 0.311, '>': 0.126, '\x99': 0.119, '\xe7': 0.111, '\xdf': 0.101, '2':
0.074, '\xdc': 0.071,'\x07': 0.07, ...
```

Le fichier est composé à 95% d'octets nuls, 0.4% de 0xbd et 0.3% de 0xff. Les autres octets sont présents en faible quantité. Ces proportions sont relativement cohérentes avec la thèse d'un format binaire inconnu, car pour beaucoup d'entre eux, 0x00 et 0xff sont les deux octets les plus fréquents. Par exemple, dans le fichier ELF **`/bin/ls`**, on compte 26% de 0x00 et 5% de 0xff. La commande **`strings`** ne révèle rien de très excitant, à part ce qui pourrait être appelé un alphabet.

```
$ strings red
zyxwvutsrqponmlkjihgfedcba`_^]\\[ZYXWVUTSRQPONMLKJIHGFEDCBA@?>=<;:9876543210/.-,+*
```

En réalité, cet alphabet est la partie émergée de l'iceberg puisqu'il s'agit de la liste décroissante des 256 valeurs possibles d'un octet. Une telle liste est fréquemment rencontrée dans des formats binaires (par exemple dans certaines images PNG). Ce qui est étrange, c'est qu'elle est à l'envers. On tente alors, en vain, de lire le fichier en partant de la fin. Au final, l'opération correcte qui va remettre cet alphabet dans le bon sens est un NOT sur tout le fichier. Le résultat

est une agréable surprise.

```
$ file red.not
'Gameboy ROM: "R", [ROM+MBC1+RAM], ROM: 2Mbit, RAM: 128Kbit'
```

Une ROM Game Boy ! Difficile de ne pas sourire en découvrant ceci. On remarque d'ailleurs que cette cartouche ne contient pas que de la ROM, mais également de la RAM et un MBC. Le *Memory Bank Controller* (MBC) est une puce qui permet d'accéder à plus de mémoire que la limite initialement supportée par une Game Boy. Il s'agit d'un hack assez immonde utilisé pour développer des jeux plus volumineux sans devoir changer la console. C'est un détail qui aura son importance plus tard. Pour l'heure, on lance cette ROM dans un émulateur Game Boy (dans mon cas **gvba**) en salivant de découvrir de quel jeu il sera question. L'effervescence est de courte durée. Une fois le jeu lancé, l'écran affiche *« Action?! »* et reste bloqué. On appuie alors sur toutes les touches, mais rien ne se passe. Pour faire avancer la situation, il va être nécessaire de se plonger dans le code assembleur z80 de la ROM. N'ayant jamais touché à cela de près ou de loin, la mission me semble assez ardue et, à cette époque, je jette rapidement l'éponge.

3.4 VERT (GAUCHE)

Voici la partie la plus sadique. Le double bloc vert situé à gauche du fond d'écran est une clé publique RSA accompagnée de données qui ont a priori été chiffrées avec. La clé RSA contient un module N de 4096 bits ! Pour information, le record de factorisation RSA a été établi sur un N de 768 bits **[6]**. Ici, nous sommes donc très loin de ce qu'il est possible de factoriser avec les avancées mathématiques actuelles. On en conclut qu'il ne va pas falloir casser la clé publique pour retrouver la clé privée, mais plutôt chercher ailleurs. Hum. Ailleurs, il n'y a rien.

L'indice de la première énigme nous parle de calculs qui vont prendre une semaine. Si cela s'applique à la factorisation du module, c'est qu'une étape de la génération de la clé publique a été volontairement mal faite et qu'elle est exploitable. Par où commencer ? Est-ce que N est vraiment un nombre semi-premier ? Dans cette clé, l'exposant de chiffrement E est anormalement grand et n'est pas premier. Il fait 4092 bits alors qu'habituellement on choisit un petit exposant pour accélérer le chiffrement (par exemple 65537). Mais qu'est-ce que tout cela implique concrètement pour la robustesse de la clé publique ? Après avoir vérifié qu'il n'est toujours pas possible d'acheter un ordinateur quantique sur *Le Bon Coin*, je m'avoue vaincu et j'abandonne le challenge en ayant résolu une seule des quatre parties. Nous sommes alors à la mi-février 2012.

4. DÉCEMBRE 2013 - JANVIER 2014

4.1 VERT (DROITE)

Un an et dix mois plus tard, je redécouvre le challenge un peu par hasard. Voulant comprendre le fonctionnement des attaques fréquentielles contre le chiffre de Vigenère, j'entreprends d'implémenter la cryptanalyse décrite sur Wikipédia **[7]**. Pour tester mon code, je cherche des exemples réels et c'est ainsi que l'épreuve verte refait surface.

L'attaque fréquentielle contre Vigenère se fait en deux étapes. La première, le test de Kasiski, permet de déterminer la longueur de la clé. L'idée est de chercher des séquences de lettres (2, 3, 4 voire plus) qui sont dupliquées dans le texte chiffré. Si l'on en trouve, il peut s'agir d'un faux positif, mais le plus probable est que deux morceaux identiques du texte clair aient été chiffrés avec la même sous-partie de la clé. Il convient alors de calculer la distance entre ces deux séquences pour en déduire toutes les longueurs de clé possibles. En corrélant les informations de chaque séquence dupliquée, on détermine la longueur la plus probable, à savoir 11 dans le cas de ce challenge.

La seconde étape est une analyse fréquentielle. Maintenant que la taille de la clé est connue, on découpe le texte en autant de sous-ensembles afin que chaque échantillon soit chiffré par une même lettre de la clé. Il suffit ensuite, pour chaque sous-ensemble, de tester les 26 lettres de l'alphabet et de trouver celle qui permet d'obtenir une fréquence proche de celle de la langue française.

Je constate avec une certaine stupéfaction que mon code fonctionne et qu'il retrouve le mot de passe PALIMPSESTE, sans avoir besoin d'utiliser l'astuce de l'adresse e-mail. Par curiosité, je cherche alors ce qu'il est advenu du challenge ANSSI et si quelqu'un l'a résolu. La réponse est non. Dans un élan de courage insoupçonné, je décide d'analyser une nouvelle fois les énigmes restantes afin de me rappeler les raisons de mon précédent abandon.

4.2 BLEU

Pour rappel, le texte bleu a subi un chiffrement par permutation dont la portée est locale. En relisant le message de la première énigme, je réalise qu'un indice n'a pas encore été exploité.

```
LE RESTE (#REF A100000) A NECESSITE BIEN PLUS DE REFLEXION.
```

Quand on cherche la chaîne A100000 sur Google, le premier lien nous renvoie vers OEIS, l'encyclopédie en ligne des suites de nombres entiers **[8]**. Oui, ce site existe réellement. La séquence d'entiers en question est [3, 6, 4, 8, 10, 5, 5, 7]. Elle correspond aux traits qui sont gravés sur l'os d'Ishango, un os datant de 22000 ans qui a été découvert au Congo. Nous voici maintenant avec un chiffrement par permutation et une suite d'entiers. J'essaye de faire le lien entre les deux, par exemple en décalant le 3ème caractère de 6 places, puis le 4ème de 8 places, etc. Mais aucune de mes tentatives n'aboutit.

C'est alors que me vient une idée. Dans les trois dernières lignes du texte, on trouve deux guillemets, un tiret, et les lettres majuscules LJMH. Deux citations étaient déjà cachées au début du challenge, je fais donc la supposition que les deux guillemets sont ceux d'une nouvelle citation qui va conclure le message déchiffré. Par exemple, avec les lettres de ces trois lignes, on peut former *« L'aventure continue »*, ce qui je trouve sonnait assez bien. Mais non. La présence du tiret me pousse à croire que l'auteur a un prénom composé. En considérant que le L sera la première lettre de la citation, je pars en quête de célébrités ayant les initiales JMH (dans le désordre). Jean-Marc, Jean-Michel, Jean-Marie, Henri-Jean, Marie-José,... José-Marie ?!? En cherchant *« José Marie Citation »* sur Google, les premiers liens me renvoient vers un certain José-Maria de Heredia qui s'avère être un poète français. Les 3 dernières lignes du texte permuté contiennent bien toutes les lettres nécessaires pour reconstruire ce nom. Ma démarche désespérée a donc été payante !

Une fois le nom du poète reconstitué, il ne reste assez de lettres que pour un ou deux mots. Je réalise que j'avais tort, les guillemets n'encadrent pas une citation, mais plutôt le titre d'un poème. Après quelques recherches, je retrouve le texte original : *« Les conquérants »*.

```
Comme un vol de gerfauts hors du charnier natal,
Fatigués de porter leurs misères hautaines,
De Palos de Moguer, routiers et capitaines
Partaient, ivres d'un rêve héroïque et brutal.
Ils allaient conquérir le fabuleux métal
```

```
Que Cipango mûrit dans ses mines lointaines,
Et les vents alizés inclinaient leurs antennes
Aux bords mystérieux du monde Occidental.
Chaque soir, espérant des lendemains épiques,
L'azur phosphorescent de la mer des Tropiques
Enchantait leur sommeil d'un mirage doré ;
Ou penchés à l'avant des blanches caravelles,
Ils regardaient monter en un ciel ignoré
Du fond de l'Océan des étoiles nouvelles.
"Les conquérants",
José-Maria de Heredia
```

En comparant les caractères du texte bleu et du poème, on constate que les seuls caractères qui diffèrent sont `- #3: -`. En clair, le secret 3 est un des mots du poème. Pour savoir lequel, il suffit de retrouver l'algorithme de permutation. Maintenant que nous avons le texte chiffré, le texte clair et la séquence d'entiers, cela va être facile. En fait, pas du tout. Malgré toutes ces informations, je ne suis pas parvenu à retrouver l'algorithme. Néanmoins, on remarque que le secret 1 fait 10 caractères. Plus tard, on apprendra que le secret 2 en fait également 10. Dans ce poème, il n'y a qu'un seul mot de 10 lettres qui se trouve proche des fameux caractères **#3** du texte bleu, il s'agit de **caravelles**. Pour l'instant, on ne peut pas affirmer avec certitude que c'est bien le secret 3, mais c'est en tout cas le meilleur candidat.

Une fois le challenge validé, son concepteur m'a expliqué quel était ce satané algorithme de permutation. Mais je ne vais pas vous le révéler pour autant. À la place, je vais utiliser la botte secrète des auteurs qui veulent parler d'un problème sans avouer qu'ils ont été incapables de le résoudre : *« La détermination de l'algorithme de permutation est laissée en exercice au lecteur »*.

4.3 ROUGE

Nous voici de retour sur la ROM Game Boy. Je découvre que fin février 2012 (peu de temps après mon abandon) quelqu'un a réussi à trouver un Konami code qui débloque un écran secret ! En tapant « Haut, Haut, Bas, Bas, Gauche, Droite, Gauche, Droite, B, A » sur l'émulateur Game Boy, un clavier virtuel apparaît. Nous avons la possibilité de rentrer trois mots, puis la chaîne *« Enjoy! »* s'affiche à l'écran accompagnée de 32 octets sous forme hexadécimale (figure 5). On comprend rapidement que le jeu n'est pas une énigme, mais plutôt une interface dans laquelle il faudra valider les trois secrets des autres épreuves. En tentant des dizaines de combinaisons, on s'aperçoit que la sortie ressemble à un mélange des trois mots passés en entrée. Par exemple, en entrant ABC, abc et 123, la sortie commence par 0x41 0x61 0x31, c'est-à-dire les codes ASCII de la 1ère lettre de chaque secret. La suite est 0x32 0x42 0x62 (2-B-b), la 2ème lettre des secrets, mais cette fois dans le désordre. La fin est 0x70 0x20 0x50 (p-espace-P), des caractères qui ne font pas partie des mots en entrée. Le mélange varie en fonction des entrées d'une manière qui ne semble pas logique. Pour comprendre l'algorithme, il va falloir examiner le code assembleur z80 de la ROM.

fig5

Figure 5 : Test de l'algorithme de mélange de la ROM Game Boy.

À ce stade, voici le plan de bataille : nous avons le secret 1 ainsi qu'un bon candidat pour le secret 3. Le secret 2 semble quant à lui impossible à trouver puisqu'il faut casser une clé RSA de 4096 bits. On peut poser comme postulat que le message généré par cet algorithme de mélange sera porteur de sens (il pourrait commencer par un mot du type *« Félicitations ! »*). Ainsi, une fois l'algorithme identifié, il devrait être possible de retrouver le secret 2 avec une attaque par force brute.

Le reverse de la ROM m'a demandé plusieurs soirées et j'ai eu du mal à trouver les bons outils. IDA ne désassemble pas le z80 dans sa version de démonstration et j'ai dû me résigner à tout faire au débogueur (BGB en est un excellent). Je me suis heurté à deux principales difficultés. La première est que le code assembleur de l'algorithme de mélange et celui qui gère l'affichage graphique sont entrelacés. Le simple fait d'identifier les morceaux de code intéressants a donc été long. J'ai finalement trouvé 9 routines de mélange qui prennent en entrée un caractère de chaque secret (on appellera leur valeur ASCII c1, c2 et c3) et génèrent trois caractères en sortie. La plupart de ces routines ne font que permuter les caractères, mais d'autres font des XOR. Voici la sortie des 9 routines en question :

```
1. c1, c2, c3
2. c1, c3, c2
3. c2, c1, c3
4. c2, c3, c1
5. c3, c1, c2
6. c3, c2, c1
7. c1^c2, c2^c3, c1^c3
8. c1^c3, c1^c2, c2^c3
9. c1^c2^0x42, c2^c3^0x42, c1^c3^0x42
```

La seconde difficulté a été de déterminer comment est choisie la routine de mélange qui va s'appliquer à chaque triplet de lettres. C'est ici que la puce MBC et le concept de *Bank Switching* rentrent en piste, car le jeu utilise ce hack matériel pour accéder à plus de mémoire. Le code des routines est situé dans 9 banques différentes. Pour changer la banque courante, il faut écrire une valeur dans la ROM sur la plage d'adresse [2000-3FFF]. La ROM étant par définition de la mémoire en lecture seule, je vous laisse méditer sur la teneur philosophique de la phrase précédente. Cette tentative d'écriture est interprétée par la puce MBC qui va charger la banque correspondante de la cartouche. En plaçant un breakpoint en écriture sur cette plage, on peut remonter jusqu'au code qui sélectionne le numéro de routine à appliquer. Ce dernier est assez horrible puisqu'en assembleur z80, il n'y a pas de modulo ; cette opération a été implémentée ici avec toutes sortes de bit-shifts. Mais l'algorithme est en définitive très simple. Le numéro de routine est le résultat du calcul (((c1 + c2 + c3) & 0xff) % 9) + 1, en utilisant toujours le triplet de l'étape N-1 (pour le premier triplet, c'est toujours la routine 1 qui est sélectionnée).

Une fois l'algorithme implémenté en Python avec succès, je code mon attaque par force brute, puis je pleure. En effet, aucun secret 2 ne permet de générer un message de sortie qui a du sens. Damned, il va donc falloir casser cette maudite clé publique RSA.

4.4 VERT (GAUCHE)

En écumant le web à la recherche d'indices sur lesquels me raccrocher, je découvre que le bien nommé blog Cryptobourrin contient plusieurs articles qui traitent de la clé RSA du challenge. Le dernier date de septembre 2013 et s'intitule *« La panne »* **[9]**. L'auteur nous apprend qu'après avoir passé un an et tout essayé, il n'a rien trouvé et a besoin de nouvelles idées. Voici un extrait de cet article.

```
Nous avons essayé successivement :
- Une factorisation en brute force de N notamment en ecm : sans succès
- L'attaque de Wiener : sans succès
- Tester la friabilité de N : sans succès
- Factoriser l'exposant public et lancer l'attaque de Wiener : sans succès
- L'attaque de De Weger : sans succès
```

Ce message est pour le moins décourageant... pour ne pas dire déprimant. En effet, après avoir lu avec grand intérêt toutes les réflexions de l'auteur, je comprends qu'il est bien plus compétent que moi en matière de crypto. Il a testé des attaques dont je ne connaissais même pas le nom. Dans ces conditions, comment puis-je espérer trouver la solution ? Après m'être documenté et avoir testé moi-même (en vain) plusieurs des attaques décrites, je réalise qu'il y a bien un cas que l'auteur n'a pas évoqué (voire même deux **[10]**). Peut-être que le texte clair est très court. Peut être qu'il s'agit simplement de **- #2: secret2 -**. Si le secret 2 est court et qu'aucun padding aléatoire n'a été utilisé avant le chiffrement, alors on peut lancer une attaque par force brute, chiffrer tous les mots de passe possibles (avec plusieurs variantes dans le formatage) et comparer les résultats avec le bloc contenu dans l'image. L'exposant public E étant très gros (4092 bits), les opérations de chiffrement sont incroyablement longues. Après une semaine de calcul, j'en suis toujours aux mots de passe de 4 caractères.

Argh. Je pense alors à abandonner définitivement le challenge. Dommage, je suis peut-être le seul à avoir avancé sur l'énigme bleue. Comme cela fait presque deux ans que le challenge est lancé, je me dis qu'après tout ce temps, les concepteurs seraient peut-être enclins à dévoiler un petit indice. Et puis Noël est dans quelques jours. J'envoie donc un e-mail à l'adresse CH4LL3N9E@ssi.gouv.fr trouvée dans la première énigme. Mais quitte à quémander de l'aide, autant le faire avec style. C'est ainsi que, repensant à cette fameuse énigme bleue, j'ai complètement craqué. J'ai écrit un poème.

```
Cher monsieur CH4LL3N9E,
Je me présente à vous, dans l'espoir avoué
De quérir quelques indices sur la force brute
Permettant de casser le second secret,
Car voilà des semaines que j'erre sans but.
La souffrance de mon CPU devient des plus critique
Et pour paraphraser notre ami José-Maria,
Chaque soir, espérant des lendemains épiques,
Je rêve que le jour suivant m'offre un résultat.
Mais ce mirage doré jamais ne s'exauce,
Je pars donc en quête de nouvelles pistes.
J'ai cuisiné cette clé RSA à toutes les sauces,
Pourtant elle reste muette, éternelle égoïste.
Quatre mille quatre-vingt-seize est un N bien trop grand.
Mais P et Q sont-ils vraiment des premiers de même taille ?
Le gigantisme de E implique des calculs lents,
Est-ce pour décourager les participants, ou est-ce la faille ?
Un padding a-t-il été utilisé avant de chiffrer le message ?
Quelque indice que ce soit serait un bijou,
Car comme mes rimes moisies le présagent
Trop de guessing finit par rendre les gens fous !
"Les cons errants"
```

Je m'attendais à toute sorte de réponses, mais certainement pas à ça : un autre poème !

```
Bien aimé homonyme,
P et son comparse se portent très bien ma foi,
Ormis le temps, il n'y a guère à deviner,
Las de ces efforts, persévérer, mais pourquoi ?
Le grand bleu résiste, pourtant bien plus armé.
Alors me direz-vous, que faire, j'en reste coi,
Regardez, d'autres ont tenté, raté, survolé,
De toute façon, ce E ne vaut pas un clou.
Point de padding et point de difficulté, mais,
Moins complexe, tentez le module ci-dessous,
Un de ses deux précieux, en ligne, vous trouverez.
```

Avec ce poème est fourni un entier de 2049 bits dont je n'ai pas su quoi faire. Au final, ne me voilà pas tellement plus avancé. Pourtant, ce poème renferme bel et bien un précieux indice puisque c'est un acrostiche ! Le début de chaque vers forme le nom « Pollard P Moins Un », qui désigne un algorithme de factorisation d'entiers.

Sur les pages Wikipédia consacrées à l'algorithme P-1 de Pollard, il est question de friabilité et du petit théorème de Fermat. N'étant pas resté en bons termes avec les mathématiques, je décide qu'au lieu d'essayer de comprendre la théorie, je vais plutôt chercher la bonne implémentation. Les articles évoquent l'outil **GMP-ECM**, qui inclue une implémentation efficace du P-1 de Pollard. De plus c'est un logiciel français, développé par l'INRIA. Pour le lancer, il suffit de lui passer le module N de 4096 bits et une borne B. Trouver cette borne est un problème. J'ai utilisé un nombre que j'ai incrémenté au fur et à mesure des essais, en repartant toujours des résultats intermédiaires. Pour une raison qui j'ignore encore, 10 jours plus tard, la factorisation n'avait pas fonctionné.

Entre temps, mes rivaux dont l'indice est parvenu aux oreilles ont réussi à factoriser la clé RSA grâce à une implémentation *custom* de l'algorithme P-1 de Pollard. Saperlipopette (pour être poli). Il va donc falloir que je comprenne rapidement la théorie et que je code ma propre implémentation. On trouve beaucoup d'articles expliquant le fonctionnement de l'algorithme P-1 de Pollard appliqué à la factorisation RSA et proposant des preuves de concept. Cependant, personne n'a jamais l'idée de s'attaquer à des modules aussi imposants.

Dans une clé publique RSA, le module N est le produit de 2 grands nombres premiers P et Q qui sont secrets. Si l'on parvient à factoriser N, on peut recalculer la clé privée. Le principe de l'algorithme P-1 de Pollard est de trouver un très (très très très) grand multiple de P-1 (ou de Q-1). Une fois ce multiple trouvé, en appliquant le petit théorème de Fermat et en calculant un PGCD, on retrouve P. Mais comment trouver un grand multiple d'un nombre qu'on ne connaît pas ? C'est ici qu'intervient la friabilité. Si le nombre P-1 est friable, il n'a que des facteurs premiers qui sont inférieurs à une borne B (on dit qu'il est B-lisse). Dans ce cas, factorielle B est très probablement un grand multiple de P-1. Mais si la borne est arbitrairement grande, calculer factorielle B est coûteux. Un article **[11]** parle d'une optimisation qui consiste à n'utiliser que des nombres primaires. Un nombre est dit *primaire* s'il peut s'écrire sous la forme *nombre premier* puissance *entier positif*. Si P-1 est B-lisse, alors le produit de tous les nombres primaires compris entre 0 et B a de bonnes chances d'être un multiple de P-1.

Pour mon implémentation, j'ai utilisé le logiciel de calcul Sage. Au final, 10 lignes de Python et 4 heures de calcul suffisent pour factoriser N.

```
from sage.all import *
N = 5346...[4096 bits]
M = prime_powers(2**30)
i = count = 0
a = 2
while True:
 a = pow(a, M[i], N)
 i += 1
 count += 1
 if count == 100000:
 count = 0
 r = gcd(a-1, N)
 if r != 1:
 break
print r
```

La fonction **`prime_powers(2**30)`** pré-calcule tous les nombres primaires compris entre 0 et 2**30. Cela ne prend que 10 secondes, mais consomme 6Go de RAM. On obtient ainsi 54,4 millions de nombres primaires, soit 20 fois moins que pour factorielle 2**30. Le code simpliste ci-dessus part du principe que B vaut 2**30. L'implémentation plus réaliste utilise la fonction **`next_prime_power()`** afin de poursuivre les calculs tant que la factorisation n'est pas un succès. Deux autres optimisations sont utilisées. Tout d'abord, tous les calculs sont effectués modulo N, sinon les nombres deviennent rapidement trop grands pour être manipulables. Ensuite, le PGCD n'est calculé qu'une fois toutes les 100 000 itérations, car si un multiple de P-1 est trouvé, il le restera 100 000 opérations plus tard. Une fois la clé privée recalculée, on peut enfin lire le bloc chiffré.

```
ACCUSEZ RECEPTION DU MESSAGE CHIFFRE A ch4ll3n9e@ssi.gouv.fr
ENVOYEZ SON EMPREINTE POUR ACQUITTEMENT.
CODES ATTENDUS POUR VOTRE LIVRAISON.
- #2: yh%Jc/!23B -
```

À nouveau, j'ai du mal à saisir le sens du message. M'est-il adressé ou l'ai-je intercepté ? L'empreinte de quel fichier faut-il envoyer ? De quels codes et de quelle livraison est-il question ? Qu'importe. Le message contient le secret 2, je peux donc avancer.

5. ADFGVX

Le secret 1 est **AX7EVT4NU!**, le deuxième est **yh%Jc/!23B** et le dernier est probablement **caravelles**. Lorsqu'on les rentre dans le jeu Game Boy, la sortie n'a rien d'exceptionnel.

```
AychXa\x12WEMif5\x15 /TeX\x15MlN23UeBs!\x00\x00
```

Très bien, mais qu'est-ce qu'on fait maintenant ? En passant en revue le challenge, je constate que la seule chose qui n'a pas encore été utilisée est la chaîne chiffrée du tout début, affichée autour du logo. De plus, le premier message parle d'un échange de clé. Peut-être que la mystérieuse chaîne que nous venons de récupérer est la clé qui va permettre de déchiffrer ces données. Elle fait 32 octets de long et l'algorithme le plus connu qui utilise des clés de 256 bits est AES. Bingo. Le déchiffrement AES-CBC du bloc de données génère une archive gzip datée du 26 octobre 2011. Nous connaissons donc avec exactitude le moment où le challenge a été inséré dans le logo.

```
$ file data
data: gzip compressed data, from Unix, last modified: Wed Oct 26 15:55:10 2011
```

Une fois l'archive décompressée, nous obtenons le fichier texte suivant :

```
FGAXAXAXFFFAFVAAVDFAGAXFXFAFAGDXGGXAGXFDXGAGXGAXGXAGXVFVXXAGXDDAXGGAAFDGGAFFXGGXXDFAXGXAXVAGXGGDFAGGGXVAXVFXGVFFGGAXDGAXFDVGGA
```

Il s'agit d'un chiffre ADFGVX, un système de chiffrement utilisé par les allemands pendant la première guerre mondiale. Il est relativement robuste, car il combine un chiffrement par substitution (utilisant une table aléatoire) et un chiffrement par permutation (utilisant un mot de passe). Il peut être cassé en mélangeant analyse fréquentielle et force brute, mais sur un échantillon aussi court, j'ai passé une longue nuit sans obtenir de résultat. Mon analyse fréquentielle ne risquait pas de fonctionner, car le message clair n'est pas du français, mais de l'allemand. En effet, en cherchant mieux, cette chaîne s'avère être un exemple qui a réellement existé **[12]**. Ce message porte le nom de *Radiogramme de la Victoire* et a joué un rôle notable lors de la première guerre mondiale. Il a été déchiffré en 1918 par le français Georges Painvin. Le texte clair allemand est *« Munitionierung beschleunigen Punkt soweit nicht eingesehen auch bei Tag »* et sa traduction française est *« hâtez l'approvisionnement en munitions, le faire même de jour tant qu'on n'est pas vu »*. Me voilà en possession d'un message en allemand. Que suis-je censé en faire ? Cela dit, il s'appelle le Radiogramme de la *Victoire*, donc peut-être ai-je terminé le challenge ? J'ai posé la question. On m'a répondu oui. Pfiou !

CONCLUSION

Il est maintenant temps de dresser un bilan de ce challenge. Tout d'abord, il faut reconnaître qu'avoir caché autant d'énigmes dans un seul fond d'écran est une belle prouesse. Le début du challenge est très plaisant et l'idée d'utiliser un jeu Game Boy comme interface de validation est à la fois originale, surprenante et instructive. Cependant, le challenge n'est pas exempt de tout reproche. Il a résisté deux ans à la communauté, ce qui est particulièrement long. Il s'agit pourtant d'un choix délibéré de l'ANSSI qui souhaitait que son challenge ne soit pas limité dans le temps. Ainsi, elle a opté pour des épreuves offrant de nombreux chemins à explorer, tout en étant avare sur les indices permettant de se guider. La conséquence inéluctable est que cela introduit une certaine dose de guessing, ce qui a le don d'exaspérer les participants.

En définitive, le challenge du logo ANSSI est relativement éloigné des habituels challenges de recrutement ou de ceux organisés en marge des conférences de sécurité, car ceux-ci sont conçus pour être résolus dans des délais raisonnables. Dans l'esprit, il s'inspire davantage de la sculpture cryptographique Kryptos **[13]**, située dans l'enceinte du quartier général de la CIA et inaugurée en 1990. À ce jour, une des quatre énigmes qu'elle renferme n'a toujours pas été percée. Certains auront d'ailleurs peut-être noté le clin d'œil des concepteurs du challenge ANSSI, puisque la clé de déchiffrement de la première épreuve – *palimpseste* – est également celle du première message de Kryptos.

Quoi qu'il en soit, malgré quelques moments de rage, ce challenge reste une expérience extrêmement positive. Mon seul regret est certainement de m'être

résigné à mendier un indice pour la clé RSA, car avec du recul, je constate que ça n'était pas si sorcier. Mais bon, sans cette demande j'aurais raté une franche rigolade ! Pour conclure, si des curieux souhaitent savoir ce qu'il y avait exactement à gagner... et bien, sachez que les détails de cette information sont placés sous licence Beerware :)

REMERCIEMENTS

J'ai une jolie ribambelle de personnes à remercier. BLue, Trou et tous les agents de l'ANSSI ayant participé à la conception du challenge. gg et geekou pour le savoureux mélange de rivalité et de coopération. L'auteur du merveilleux blog Cryptobourrin. alonetrio pour avoir créé **anssi.santo.fr**. RoDGeR pour son OCR de folie. pjalaber pour sa connaissance des subtilités de la Game Boy. Et enfin chok0bn, Jiss, clem1, plo et chouchou pour le coup de main.

RÉFÉRENCES

[1] **http://www.ssi.gouv.fr/fr/anssi/evenements/discours-de-patrick-pailloux-lors-de-la-conference-de-cloture-des-assises-de-la.html**

[2] **http://www.ssi.gouv.fr/fr/anssi/mediatheque/ecran.html**

[3] **http://www.ssi.gouv.fr/fr/anssi/evenements/discours-de-patrick-pailloux-directeur-general-de-l-agence-nationale-de-la.html**

[4] **http://anssi.santo.fr/rip-from-somewhere/00-info.txt**

[5] **http://hack-and-fun.blogspot.fr/2011/02/detection-de-lsb.html**

[6] **http://en.wikipedia.org/wiki/RSA_Factoring_Challenge**

[7] **http://fr.wikipedia.org/wiki/Test_de_Kasiski**

[8] **http://oeis.org/A100000**

[9] **http://cryptobourrin.wordpress.com/2013/09/01/la-panne/**

[10] La littérature propose une méthode rudement efficace pour contrer un chiffrement RSA 4096 bits : **http://xkcd.com/538/**

[11] **http://sage.csuohio.edu/home/pub/21/**

[12] **http://fr.wikipedia.org/wiki/Georges_Painvin**

[13] **http://en.wikipedia.org/wiki/Kryptos**

SOMMAIRE

PAR LE MÊME AUTEUR

6 BONNES FAÇONS DE RENDRE VOTRE PROGRAMME VULNÉRABLE

MSC n° 065 | janvier 2013 | Pierre Bienaimé

Sécurité

LES MÉCANISMES D'AMPLIFICATION

MSC n° 072 | mars 2014 | Pierre Bienaimé

Sécurité

PETIT PLAIDOYER EN FAVEUR DES CHALLENGES DE SÉCURITÉ

MSC n° 073 | mai 2014 | Pierre Bienaimé

Sécurité Société

CANYOUFINDIT.CO.UK : LE NOUVEAU CHALLENGE DU GCHQ

MSC n° 071 | janvier 2014 | Pierre Bienaimé

Sécurité

CANYOUCRACKIT.CO.UK : ANALYSE D'UN BUZZ ET SOLUTION DU CHALLENGE

MSC n° 060 | mars 2012 | Pierre Bienaimé

Sécurité

D'AUTRES ARTICLES PEUVENT VOUS INTÉRESSER

Dossier : Comprendre les attaques sur le WiFi

GNU/Linux Magazine HS n° 099 | novembre 2018

Sécurité

Nous utilisons de plus en plus de périphériques Wi-Fi : ordinateurs bien sûr, mais également tablettes, smartphones et toute une série de...

Lire l'extrait

Entretien avec Olivier Levillain, expert en sécurité informatique

GNU/Linux Magazine HS n° 099 | novembre 2018 | gapz

Témoignage

Olivier Levillain est un expert en sécurité informatique, avec notamment une expérience importante sur TLS et la sécurité des langages de...

Lire l'extrait

Traçage Wi-Fi : qu'en est-il en pratique ?

GNU/Linux Magazine HS n° 099 | novembre 2018 | Célestin Matte

Sécurité

Les smartphones sont parfois désignés comme des « mouchards de poche », leurs risques pour la vie privée étant dénoncés. Les problèmes de...

Lire l'extrait

U-Boot : à la découverte du « démarrage vérifié »

GNU/Linux Magazine n° 221 | décembre 2018 | Pierre-Jean TEXIER

Embarqué

Sécuriser le processus de démarrage est la première étape afin de garantir qu'un système Linux embarqué est fiable. Cette technique,...

Lire l'extrait

Est-il si simple de mettre en place une attaque Wi-Fi ?

GNU/Linux Magazine HS n° 099 | novembre 2018 | Tristan Colombo

Sécurité

Nous sommes de plus en plus connectés et, pour la plupart d'entre nous, nous laissons le Wi-Fi activé sur nos smartphones 24h/24. En effet,...

Lire l'extrait

Gérez vos listes avec le framework IPset et le pare-feu Netfilter

GNU/Linux Magazine n° 221 | décembre 2018 | Laurent Butti

Sécurité

La brique pare-feu Linux (Netfilter) n'est plus à présenter. Le framework IPset est relativement méconnu bien qu'utilisable conjointement avec...

Lire l'extrait