Stéganographie - DualEC - 150 points

Kévin DUVERGER

Table des matières

1 Résolution DualEC :

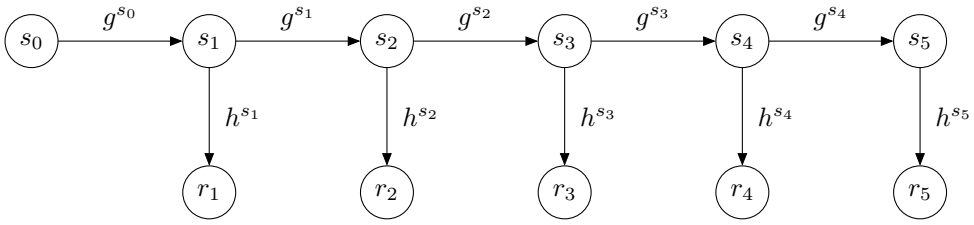
C'est le challenge où nous allons devenir la NSA et tenter de faire de la surveillance de masse.

On peut encore une fois tenter de se connecter au serveur `146.59.227.136` sur le port `23414` :

```
Dear PRG standardization organism, please give me the parameters to use in my randomness generation.
p = ?
g = ?
h = ?
```

Le serveur semble donc nous demander des valeurs p, g, h à utiliser dans le générateur d'aléatoire.

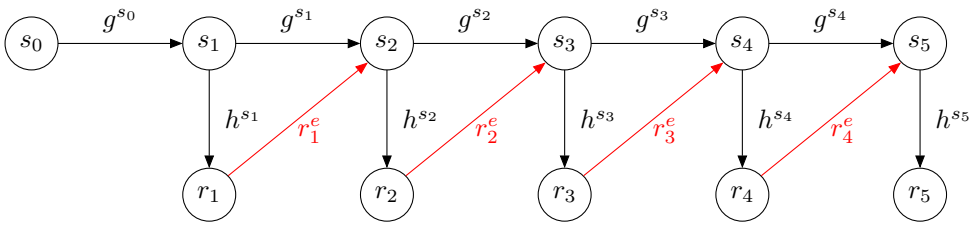
Explorons donc maintenant comment ce générateur d'aléatoire fonctionne, voici l'idée :



On part donc d'un état s_0 aléatoire et qui sera donné par le programme / utilisateur qui se sert du générateur. Pour passer à l'état s_{i+1} à partir de l'état s_i, il suffit de calculer $s_{i+1} = g^{s_i}$. Finalement, la i-ième valeur aléatoire est donnée par $r_i = h^{s_i}$ avec h un autre paramètre public du schéma.

L'attaquant que nous sommes ici a donc le droit de maniupler p, g, h pour faire apparaître une backdoor. Le but de cette backdoor est de pouvoir récupérer un état interne s_i à partir d'un aléatoire qui est parti sur le réseau r_j (qui est facile d'accès).
L'idée la plus simple serait donc de trouver un moyen de passer de r_i à s_{i+1}, ce que l'on peut faire en prenant par exemple $g = h$. Le problème de cela est que les états internes vont sortir en clair sur le réseau et nous ne pourrons plus être les seuls à utiliser notre backdoor. On pourrait donc ensuite tenter $g = h^2$ et en calculant $r_i^2 = (h^{s_i})^2 = (h^2)^{s_i} = g^{s_i} = s_{i+1}$ on a l'état suivant, mais le fait que ce soit 2 fait encore que nous ne sommes pas les seuls à pouvoir utiliser la backdoor. L'idée est donc que nous (la NSA) prenions un e aléatoire et gardé secret tel que $g = h^e$, ce qui nous permet de récupérer s_{i+1} en calculant r_i^e, et nous serons les seuls à pouvoir l'utiliser car nous sommes les seuls à connaître e et le logarithme discret est difficile !

Sous une forme un peu plus schématique, voici ce que cela rajoute (retenez bien que e est secret) :



On peut donc maintenant repasser au challenge pour le résoudre !

On peut tenter d'envoyer une petite valeur de p mais le programme nous répond :

```
The p value you gave is not a 512-bit one !
```

Donc on peut en envoyer un de 512 bits, mais vous devriez recevoir le message suivant :

```
p is not a safe prime (this means that (p - 1) / 2 is not a prime while it should be) !
```

On peut donc modifier le programme pour avoir un premier qui vérifie cela (ça va prendre un peu de temps). Une fois avoir fait cela, le programme réalise quelques tests pour vérifier que h et g ne sont pas faibles.
Finalement, en faisant en sorte que $g = h^e$, soit le programme vous dira que votre valeur de e est trop petite, soit il vous demandera une randomness et vous devrez envoyer le prochain état interne en calculant r_i^e !

Voici un programme en Python qui permet de respecter toutes ces conditions :

```python
import socket, os, sys
import random
from Crypto.Util import number

server = "146.59.227.136"
```

```
port = 23414

ma_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
try:
    ma_socket.connect((server, port))
except Exception as e:
    print("Problème de connexion", e.args)
    sys.exit(1)

# Step 1 = generating parameters
print("Generating parameters (this might take some time)")
p = 0
iteration = 0
while True :
    p = number.getPrime(512)
    iteration += 1
    print("Testing prime " + str(iteration))
    if number.isPrime((p - 1) // 2) :
        break
h = random.randint(2, p - 1)
gExpo = random.randint(2, p - 2)
g = pow(h, gExpo, p)

# Step 2 = sending them
data = ma_socket.recv(4096).decode("utf-8")
ma_socket.sendall((str(p) + "\n").encode("utf-8"))
data = ma_socket.recv(4096).decode("utf-8")
ma_socket.sendall((str(g) + "\n").encode("utf-8"))
data = ma_socket.recv(4096).decode("utf-8")
ma_socket.sendall((str(h) + "\n").encode("utf-8"))

# Step 3 = getting the randomness
data = ma_socket.recv(4096).decode("utf-8")
randomValue = int((data.split("\n")[0]).split(" : ")[1])

# Step 4 = predicting the next state
nextState = pow(randomValue, gExpo, p)
ma_socket.sendall((str(nextState) + "\n").encode("utf-8"))
data = ma_socket.recv(4096).decode("utf-8")
print(data)

# Finally, closing the socket
ma_socket.close()
```