Injection de faute sur RSA-CRT

Kévin Boyeldieu
Titouan Tanguy
TLS-SEC

Mars 2018

Abstract

Le but de ce challenge est de montrer dans une version simplifiée comment une injection de faute sur un cryptosystème peut le compromettre si aucune attention particulière n'a été portée à ce problème. Pour ce faire nous prendrons l'exemple de l'attaque Bellcore sur l'algorithme RSA-CRT que nous avons implémenté sur une carte FPGA. Ici l'injection de faute sera faite de façon *artificielle* puisque pour éviter de dégrader le matériel, et par manque de moyen, il suffira d'appuyer sur un bouton pour fauter l'algorithme.

Description du tutoriel

Lors de ce challenge vous êtes présenté à une carte FPGA Xilinx Nexys2 sur laquelle est implémenté l'algorithme RSA-CRT.
Cette carte sert d'oracle de signature à message imposé. C'est à dire que vous pouvez lui demander de signer un message choisit à l'avance, par un autre que vous. Dans le cas présent, le message qui sera signé est toujours le même l'attaque n'exploitant pas le fait de pouvoir signer plusieurs messages différents.
Sur la carte fournie, seuls quelques composants seront utiles :

- **Bouton B18** qui servira à remettre à zéro tous les modules de la carte (pas nécessaire pour réaliser le tutoriel)
- **Bouton D18** qui permet de commencer la signature du message
- **Bouton E18** qui permet d'injecter la faute lorsque cela est possible
- **LED J14** qui indique que la signature est finie et que la carte est prête pour signer à nouveau
- **LED J15** qui indique que si vous appuyez sur le Bouton E18, l'algorithme sera fauté
- **Port P9** qui permet la communication entre la carte et le programme python fourni

Une fois la signature (fautée ou non) faite, la carte communiquera à l'ordinateur le résultat qu'elle obtient. Pour comprendre comment exploiter la faute la description suivante est nécessaire :

```
Parametre publique utile pour l'attaque
n = pq (p et q, les secrets du proprietaire de la carte,
        sont des nombres premiers)

Constantes
d_p = d mod (p-1)
d_q = d mod (q-1)
i_q = q^{-1} mod p

Exponentiations
s_p = m^{d_p} mod p
s_q = m^{d_q} mod q

Recombinaison de Garner
s = s_q + q (i_q (s_p - s_q ) mod p)
```

Lorsque nous avons dit que le **Bouton E18** permet de fauter l'algorithme, il faut comprendre qu'appuyer sur ce bouton au bon moment permet de rendre le calcul de s_p faux, en substituant le résultat attendu par une valeur aléatoire. Le but pour vous est alors de retrouver les paramètres p et q qui sont les secrets de RSA-CRT et qui permettent, s'ils sont connus, de signer les messages de votre choix en usurpant l'identité du propriétaire de la carte.

Solution du tutoriel

Comme expliqué dans la description du tutoriel, le but pour l'utilisateur est de retrouver les paramètres p et q qui sont les secrets de RSA-CRT.
Pour ce faire la marche à suivre est la suivante :
1- Dans un premier temps il faut demander à notre oracle de signer le message et récupérer la signature non fautée que l'on notera *signature* dans la suite de la solution. Pour cela il suffit donc d'appuyer sur le **Bouton D18** et d'attendre de voir le résultat apparaître à l'écran grâce au programme python fourni.
2- Ensuite il faut demander à notre oracle de signer le message, et le pousser à la faute. On récupère ainsi la signature où la composante s_p est fausse. On notera cette signature fautée *signatureF* dans la suite de la solution. Pour cela il suffit d'appuyer sur le **Bouton D18** et lorsque l'on voit la **LED J15** s'allumer, appuyer sur le **Bouton E18**, puis d'attendre de voir le résultat apparaître à l'écran comme précédemment.
3- Maintenant que nous disposons de *signature* et *signatureF* il nous faut calculer leur différence. On obtient ainsi :

```
S = signature - signature F
  = s_q + q (i_q (s_p - s_q ) mod p)
        - s_q + q (i_q (s_p_faut  - s_q ) mod p)


  = q ((i_q (s_p - s_q ) mod p)
        - (i_q (s_p_faut  - s_q) mod p))
```

La propriété intéressante ici est qu'on sait que notre S est un multiple de q.
4- Enfin, maintenant que l'on dispose d'un multiple de l'un des facteurs premier du paramètre publique n, il nous suffit de calculer le plus grand commun diviseur entre S et n. La complexité de l'algorithme de calcul de $PGCD$ étant bien moindre que celui du calcul de facteurs premiers d'un grand nombre, il devient réalisable sur les données que nous avons. Ainsi nous obtenons $PGCD(S, n) = q$ puis tout aussi facilement $n/q = p$, le propriétaire de la carte n'a donc plus de secret que nous ne possèdons pas.

Procédure d'installation

Pour mettre en place le challenge, il faut avoir une carte FPGA Xilinx Nexys2, ainsi qu'un ordinateur sous Linux avec un port COM et un câble permettant la liaison entre notre FPGA et le port COM de l'ordinateur.
En premier lieu il faut programmer la carte FPGA avec *nexys2_rsa_crt.bit*. Le plus simple pour ce faire est de connecter le port micro-usb de la carte à un port USB de l'ordinateur et d'utiliser la commande :

```
djtgcfg -d Nexys2 prog -i 0 -f nexys2_rsa_crt.bit
```

Dans le cas où l'on voudrait que le programme reste sur la carte après avoir éteint cette dernière il faut utiliser le fichier *nexys2_rsa_crt.mcs* et la commande suivante :

```
djtgcfg -d Nexys2 prog -i 1 -f nexys2_rsa_crt.mcs
```

Ces deux commandes nécessitent l'installation de Digilent Adept software.

Une fois la carte prête, la **LED J14** devrait s'allumer. Il ne suffit plus alors qu'à lancer le programme python *uart_receiver.py* sur la machine Linux (cela peut nécessiter les droits administrateurs). Ce programme permet de récupérer les signatures calculées et envoyées par la carte.
Pour le reste du tutoriel, à savoir le calcul de la différence et du $PGCD$ nous ne fournissons pas de sources, mais une console python par exemple fonctionnera très bien.