Pablo Rauzy <pr@up8.edu>
https://pablo.rauzy.name/teaching/is

Introduction à la sécurité

TP 5 : BellCoRe

Dans ce TP :

— Attaque d'une signature RSA par injection de faute.

Exercice 0.

Démarrage et recommandations.

1. Récupérez les fichiers nécessaires à ce TP sur `https://pablo.rauzy.name/teaching/is/11-tp-bellcore.tgz`.
2. Installez la plateforme SeseLab depuis `https://pypi.org/project/seselab` avec la commande `pip3 install seselab`.

Exercice 1.

Optimisation de RSA avec le CRT.
Cet exercice est à faire en Python.

1. On va commencer par générer une paire de clefs RSA classiques.
   (a) → Générez deux nombres premiers `p` et `q` de 512 bits et assurez-vous qu'ils soient différents.
   (b) → Calculez `n` le module RSA.
   (c) On va utiliser la valeur 65537 pour `e`.
   → Assurez-vous que `e` soit bien premier avec `p-1` et avec `q-1`.
   (d) → Générez maintenant `d`.
   (e) → Générez un nombre aléatoire `m`.
2. Lors d'une signature RSA $s = m^d \bmod n$ on sait que $n = p \times q$ où p et q sont deux nombres premiers.
   Comme on l'a vu en cours, cela permet d'utiliser le théorème des restes chinois (*Chinese remainder theorem* — CRT) pour optimiser le calcul.
   → Comment change la clef privée pour cette optimisation ?
3. → Générez les nouvelles valeurs nécessaires.
4. → En utilisant la fonction `time` de la bibliothèque `time` de Python, mesurez le temps que prend votre code Python pour faire 1000 fois le calcul de la signature de `m` avec RSA classique.
5. → De la même manière, calculez le temps nécessaire au calcul de 1000 signatures du message `m` avec cette fois-ci CRT-RSA.
6. → Quel facteur de vitesse vous fait gagner l'optimisation CRT ? (vérifiez tout de même que le résultat de vos signatures est identiques dans les deux cas).

Exercice 2.

Attaque par injection de faute sur CRT-RSA.

1. On comprend donc pourquoi l'optimisation CRT-RSA est indispensable dans les systèmes contraints en ressources (comme les cartes bancaires par exemple). Le soucis avec CRT-RSA est sa vulnérabilité aux attaques par injection de faute…
   Dans le fichier `crtrsa.asm`, il y a une implémentation de CRT-RSA.
   Voici la clef publique utilisée pour cette implémentation : $(e, n) = (17, 47775493107113604137)$.
   Vous pouvez lancer le calcul avec `seselab -i crtrsa.asm /dev/null` : l'option `-i` active la possibilité de faire des injections de fautes (avec `^C`), et on utilise `/dev/null` comme log de consommation car on ne s'y intéresse pas cette fois-ci.
   Lorsque vous faite `^C` pendant le calcul, vous aurez le choix du type de faute à injecter dans le calcul avant que celui-ci ne reprenne.
   → Essayez d'injecter une ou des fautes de différentes natures de manière à avoir un résultat faux à la fin du calcul.
2. → Factorisez n en p et q à l'aide de l'attaque BellCoRe.
3. → Retrouvez la clef privée.