Théorie des codes - TP 4
ZZ3 F5 - Réseaux et Sécurité Informatique
Cryptographie asymétrique : attaques par canneaux détournés sur les signature électroniques RSA et DSA

Rendu : au plus tard le 30 Octobre, par mail à charles.olivier-anclin@uca.fr.

Partie 1 : Injection de faute sur la signature RSA

Afin d'améliorer l'efficacité de la signature RSA, l'algorithme de signature est souvent implémenté en se basant sur un théorème mathématique : le *Théorème des Restes Chinois* (*Chinese Remainder Theorem*, abrégé CRT).

Theorem 1. *Soit $n = p \times q$, en supposant que p et q sont premiers entre eux, et si a_1 et a_2 sont des entiers quelconques, alors le système*

$$\begin{aligned} x &\equiv a_1 \pmod{p} \\ x &\equiv a_2 \pmod{q}, \end{aligned}$$

a une unique solution modulo n. En algèbre abstraite, le théorème est souvent reformulé ainsi : si p et q sont premiers entre eux, la fonction

$$f\colon x \bmod p \mapsto (a_1 = x \bmod p,\ a_2 = x \bmod q)$$

définit un isomorphisme $\mathbb{Z}/n\mathbb{Z} \cong \mathbb{Z}/p\mathbb{Z} \times \mathbb{Z}/q\mathbb{Z}$. Ce qui équivaut à une bijection entre les éléments des deux ensembles qui préserve les opérations :

$$f(a+b) = f(a) + f(b) \text{ et } f(a \cdot b) = f(a) \cdot f(b).$$

Étant donné a_1 et a_2, et en considérant $n_1 = p$ et $n_2 = q$, la fonction inverse est donnée par $x = a_1 \cdot e_1 + a_2 \cdot e_2$, où $e_i = k_i(k_i^{-1} (\bmod\, n_i))$ et $k_i = n/n_i$.

Ainsi cela permet de réaliser l'exponentiation pour un modulo plus petit. On obtient l'algorithme suivant pour RSA version CRT.

Generation de clef : Comme RSA normal avec les clefs e et d, on précalculera aussi $A = q(q^{-1} \bmod\ p)$ et $B = p(p^{-1} \bmod\ q)$.

Signature : On calcule $M_p = M \bmod\ p$, $M_q = M \bmod\ q$ et $\sigma_1 = M_p^d (\bmod\, p)$, $\sigma_2 = M_q^d (\bmod\ q)$. On fini par calculer $\sigma = A \cdot \sigma_1 + B \cdot \sigma_2$.

Vérification : Comme pour RSA standard, cette phase n'est pas accélérable car p et q sont secrets.

Attaque par injection de faute. Une attaque par injection de faute (Fault Injection Attack) est un type d'attaque physique qui consiste à perturber délibérément le fonctionnement normal d'un système cryptographique en y provoquant des fautes ou des erreurs. Pour cela on everra par exemple une surchagre dans le processeur à un moment très précis du calcul. L'objectif est d'extraire des informations sensibles, comme des clés cryptographiques, en exploitant une erreur potentiel du processeur.

Attaque de Bellcore. C'est une attaque par injection de faute. En pratique on perturbe le processeur durant l'execution d'une partie du calcul en injectant un courant qui va perturbé les calculs. Le même message est signé deux fois. La première execution s'effectue normalement, la seconde sera perturbé sur la valeur σ_1, ainsi transformé en σ_1^*. On obtient donc :

$$\sigma_1 \neq \sigma_1^* \quad ; \quad \sigma_2 = \sigma_2^*.$$

Cela engendre une différence $\sigma - \sigma^*$ non nul modulo n, en effet :

$$\sigma - \sigma^* = \sigma_1 - \sigma_1^* \neq 0 \pmod{p}$$
$$\sigma - \sigma^* = \sigma_2 - \sigma_2^* = 0 \pmod{q},$$

Alors q divise $\sigma - \sigma^*$ et p ne divise pas $\sigma - \sigma^*$ Donc $\mathsf{PGCD}(\sigma - \sigma^*, n = p \times q) = q$ On a n, σ et σ^*, on en déduit q, on en déduit p. On a tout !

1. Écrivez une fonction signant un message aléatoire avec RSA basé sur le CRT et la technique décrite plus haut.
2. Modifiez votre fonction afin de pouvoir engendrer une erreur sur le calcul de σ_1 (la génération de l'erreur peut être effectuée de quelque façon que ce soit).
3. Implémentez l'attaque de Bellcore et retrouvez p et q à partir de deux signatures et des paramètres publics.

Partie 2 : Fuite d'aléatoire sur DSA

Cette partie est indépendante de la partie précédente. Vous êtes invité à utiliser un nouveau fichier source. Il est conseillé de tirer parti du code déjà produit pour l'implémentation de RSA.

La signature *Digital Signature Algorithm* (DSA) est un ancien standard américain de signature électronique [1]. Toutefois, l'algorithme DSA peut encore être utilisé pour vérifier les signatures précédemment générées. Il a plus récemment été décliné sous des versions nommées ECDSA et EdDSA, pour *Elliptic Curve DSA* et *Edwards-curve DSA*, qui sont les standards actuels. La spécification algorithmique de ECDSA et EdDSA est à peu près la même que pour DSA, mais ces deux derniers sont définis sur ce qu'on appelle les *courbes elliptiques*. Les courbes d'Edwards ne sont qu'une façon analogue de les représenter découverte par le mathématicien Harold Edwards. En pratique, cette représentation permet d'accélérer certains calculs. Un troisième standard existe aussi : RSA-PSS (PSS pour *Probabilistic Signature Scheme*).

Voici l'algorithme de la signature DSA qui se base sur une fonction de hachage H [2] :

Signature :

- Choisir un nombre aléatoire k tel que $k \in \{1 \dots q-1\}$.
- Calculer $r = \left(g^k \bmod p\right) \bmod q$.
- Tirer un nouveau k dans le cas (très peu probable) où $r = 0$.
- Calculer $s = \left(k^{-1}\left(H(m) + xg^k\right)\right) \bmod q$.
- Tirer un nouveau k dans le cas (très peu probable) où $s = 0$.

La signature est (r, s).

Vérification :

- Rejeter la signature si $0 < s < q$ ou $0 < r < q$ n'est pas vérifié.
- Calculer $w = s^{-1} \bmod q$.
- Calculer $u_1 := H(m) \cdot w \bmod q$.
- Calculer $u_2 := r \cdot w \bmod q$
- Calculer $v := (g^{u_1} y^{u_2} \bmod p) \bmod q$

La signature est valide si $v = r$.

L'objectif de cette seconde partie du TP est de programmer l'algorithme DSA décrit ci-dessus ainsi qu'une attaque basée sur une fuite lors de l'exécution de l'algorithme. L'attaque suppose

1. https ://csrc.nist.gov/pubs/fips/186-5/final
2. `https://docs.openssl.org/master/man3/SHA256_Init/` `https://docs.openssl.org/master/man7/EVP_MD-SHA3/`

que l'élément aléatoire utilisé pour produire la signature (l'élément k) aurait fuité.

1. Écrivez les fonctions de génération de clés KeyGen, la fonction de signature Sign, et la fonction de vérification des signatures Verify.
2. Modifiez votre fonction de signature afin qu'elle retourne l'aléa utilisé lors de la signature.
3. Trouvez comment il est possible de retrouver la clé secrète à partir d'une signature et de l'aléa associé à la signature.
4. Implémentez cette attaque et testez-la en vérifiant si votre attaque réussit à retrouver la clé secrète.