Attaques par canal auxiliaire
Mini-projet M1-DID, I231

1 Implantation efficace et sûre de calcul d'exponentiation modulaire

1.1 Objectifs du mini-projet

On donne :
— un dossier `Ressources` contenant des fichiers sources `C` et un `Makefile` ;
— le résumé de cours contenant les algorithmes ;
— la documentation GMP.

L'objectif général de ce projet est d'implanter les primitives de calculs pour le protocole RSA. Ceci se décline de la façon suivante :

1. implanter un générateur de nombres aléatoires de taille arbitraire :
   — implanter le test de primalité utilisant le petit théorème de Fermat ;
   — implanter le test de primalité de Solovay-Strassen ;
   — implanter les générateurs respectifs, pour RSA et DSA, selon les recommandations du NIST FIPS 186-4 (notamment en déterminant un générateur d'ordre grand, respectant les critères de la norme du NIST) ;
2. Monter une attaque *Simple Power Analysis*, et déchiffrer le message secret ;
3. Monter une attaque *Differential Power Analysis*, et déchiffrer le message secret ;

Ces objectifs sont déclinés dans les sections qui suivent.

2 Préliminaires : génération de nombres premiers

2.1 Génération de nombres premiers aléatoires de taille arbitraire

2.1.1 Générateur utilisant le test basé sur le petit théorème de Fermat

Dans ce test, on utilise le petit théorème de Fermat :

Théorème 2.1. Soit p un nombre premier et a un entier tel que $0 < a < p$.
On a

$$a^p \equiv a \bmod p$$

ce qui revient à

$$a^{p-1} \equiv 1 \bmod p$$

2.1.2 Générateur utilisant le test de primalité de Solovay-Strassen

Le test de Solovay-Strassen exploite le critère d'Euler (déterminer si a est résidu quadratique mod$p$).

Théorème 2.2. Soit p un nombre premier impair. Soit $a \in \mathbb{N}$. On a :

— Si a est un résidu quadratique modulo p, alors

$$a^{\frac{p-1}{2}} \equiv 1 \bmod p$$

— Si $a \equiv 0 \bmod p$ (a est un mutiple de p), alors

$$a^{\frac{p-1}{2}} \equiv 0 \bmod p$$

— Si a n'est pas un résidu quadratique modulo p, alors

$$a^{\frac{p-1}{2}} \equiv -1 \bmod p$$

On définit les symoles de Legendre et Jacobi de la façon suivante :

Définition 2.1. Symbole de Legendre On definit le symbole
Si a est un résidu quadratique modulo p, alors

$$\left(\frac{a}{p}\right) = 1$$

Si $a \equiv 0 \bmod p$ (a est un mutiple de p), alors

$$\left(\frac{a}{p}\right) = 0$$

Si a n'est pas un résidu quadratique modulo p, alors

$$\left(\frac{a}{p}\right) = -1$$

Définition 2.2. Symbole de Jacobi
Soit $n = \prod_{i=1}^{k} p_i^{e_i}$, le symbole de Jacobi se définit comme suit :

$$\left(\frac{a}{n}\right) = \prod_{i=1}^{k} \left(\frac{a}{p_i}\right)^{e_i}$$

Le test de Solovay se présente maintenant de la façon suivante :

	n nombre impair dont on veut tester la primalité
$\rightarrow$	$a \leftarrow random(1, n-1)$
$\rightarrow$	si $a^{\frac{n-1}{2}} \equiv \left(\frac{a}{n}\right)$
	alors renvoyer "n est un nombre premier" (prob. $\approx$ 0,5)
$\rightarrow$	sinon
	alors renvoyer "n est factorisable (composite)" (avec certitude)

On peut exploiter ce test qui n'est pas totalement déterministe (à cause des pseudo-premiers d'Euler, voir cours). Une façon de se prémunir de mauvaises surprises est de réitérer le test à plusieurs reprises pour augmenter la confiance dans le résultat. On peut aller de 10 à 80 itérations.

2.2 Travail demandé

En termes d'implantation, on pourra procéder par étape :

— on utilisera pour commencer les fonctions suivantes de la bibliothèque `GMP` (voir documentation) :
  — pour l'exponentiation : `void mpz_powm(out,a,exp,n)`
  — pour le symbole de jacobi : `int mpz_jacobi(a,n)` (le résultat est la valeur de retour de la fonction)

3 Attaque *Simple Power Analysis*

3.1 Scénario

Ève accède au téléphone d'Alice pendant qu'elle envoie un message chiffré à Bob. Cet communication est chiffrée à l'aide d'une clé échangée par un protocole de Diffie-Hellmann (voir rappels). L'objectif d'Ève, pour déchiffrer la correspondance entre Alice et Bob, c'est de connaître la clé secrète échangée.

Pour ce faire, Ève mesure le rayonnement électromagnétique du téléphone d'Alice pendant l'échange de clé, et l'enregistre dans le fichier `SPATrace.dat`.

l'échange de clé se fait dans un corps fini premier (voir rappels) avec un nombre premier de taille 2048 bits.

L'algorithme utilisé pour le calcul est le *Right-to-left Square-and-multiply* (algorithme 1).

Algorithm 1 *Right-to-left Square-and-multiply*

Require: Les entiers $p \in \mathbb{P}$, un générateur g et $e = (e_{t-1}, \dots e_0)_2 \in [1, \dots, p[$, avec $e_{t-1} = 1$.
Ensure: $X = g^e \bmod p$

```
1: X ← 1
2: for i = 0 to t − 1 do
3:    if e_i = 1 then
4:       X ← X · g mod p
5:    end if
6:    g ← g^2 mod p
7: end for
8: return (X = g^e mod p)
```

Tous les paramètres sont donnés dans le fichier `SPAparameters.txt`.

Saurez-vous retrouver la clé privée d'Alice et ainsi calculer la clé secrète partagée entre Alice et Bob ?

3.2 Travail à réaliser

Vous rédigerez un compte-rendu répondant au questions suivantes, en fournissant vos scripts et les commandes (en particulier `openssl`) que vous avez utilisées.

1. Visualiser la trace et les valeurs enregistrées ? Que constatez vous ?
2. À l'aide des paramètres de complexité et de ratio carré/multiplication données dans le fichier `SPAparameters.txt`, évaluer le nombre de points d'enregistrement par carré et multiplication.

3. Diviser la trace en motif correspondant aux carrés et multiplications, et en déduire la clé privée d'Alice ayant servi à l'échange de clé de Diffie-Hellmann. Ce travail sera automatisé par un script (Python par exemple).
4. Implanter en Python le carré, la multiplication et l'exponentiation modulaires (le Python implante le calcul entier en multi-précision, ce qui simplifie le travail). Ceci vous permet de calculer la clé échangée par Alice et Bob. À l'aide de cette clé, vous pourrez déchiffrer le message envoyé par Alice à Bob qu'Éve a intercepté dans le fichier `outut.cph`, connaissant le protocole utilisé (voir dans `SPAparameters.txt`). Dans votre compte-rendu, vous fournirez les scripts Python et les commandes `openssl` permettant le déchiffrement du message d'Alice, que vous fournirez également.

4 Attaque *Differential Power Analysis*

4.1 Scénario

Alice soupçonne le piratage de ses communications avec Bob. Elle lui a proposé de refaire un échange de clé de Diffie-Hellmann en utilisant un nouvel algorithme d'exponentiation modulaire, le *Right-to-left Square-and-multiply-always* présenté algorithme 2.

Algorithm 2 *Right-to-left Square-and-multiply-always*

Require: Les entiers $p \in \mathbb{P}$, un générateur g et $e = (e_{t-1}, \dots e_0)_2 \in [1, \dots, p[$, avec $e_{t-1} = 1$.
Ensure: $X = g^e \bmod p$
1: $X_0 \leftarrow 1, X_1 \leftarrow 1$
2: **for** $i = 0$ to $t-1$ **do**
3: $\quad X_{e_i} \leftarrow X_{e_i} \cdot g \bmod p$
4: $\quad g \leftarrow g^2 \bmod p$
5: **end for**
6: **return** $(X_1 = g^e \bmod p)$

À l'aide de cette nouvelle clé secrète, Alice échange à nouveau un message avec Bob qu'Ève veut déchiffrer. Le travail pour Ève se complique car elle doit cette fois monter une attaque plus sophistiquée (voir rappels). Elle doit en effet solliciter plusieurs échanges de clé à Alice, auxquels Alice répondra en utilisant le même exposant secret a.

— pour chacun des échanges de clé qu'elle sollicite, Ève connaît la base de l'exponentiation qu'elle lui envoie pour le calcul de la clé secrète, et connaît la clé secrète puisqu'elle peut la calculer avec son propre exposant secret et la base que lui a envoyé Alice. Ces données sont contenues dans le fichiers `PTCT.dat`. Chaque ligne de ce fichier contient deux valeurs : la base C_i qu'Ève a envoyé à Alice, et le résultat que doit obtenir Alice qu'Ève a calculé, soit $A_i^c \bmod p$, c étant l'exposant privé d'Ève et $A_i = g^a \bmod p$.
— durant les calculs qu'effectue Alice sur son téléphone, Ève enregistre les rayonnements électromagnétiques. Elle obtient ainsi 1000 traces (les fichiers `DPATrace`i`.dat`, dont le tracé est fourni par les fichiers correspondants `DPATrace`i`.png`, avec $0 \leq i < 1000$).
— elle compte sur vous pour exploiter ces traces en montant l'attaque *Differential Power Analysis* !

4.2 Travail à réaliser

Vous rédigerez un compte-rendu répondant au questions suivantes et en fournissant vos scripts et les commandes (en particulier `openssl`) que vous avez utilisées. Comme précédemment,

on se reportera au fichier texte `DPAparameters.txt` pour les grandeurs caractéristiques nécessaire au montage de l'attaque.

1. Estimation du deuxième bit de la clé :

   On dispose de l'information suivante : le premier bit de l'exposant secret d'Alice est à 1. En examinant les traces fournies, quelles différences remarquez vous par rapport à la trace SPA précédente ? Connaissant la longueur de la clé et l'algorithme utilisé, on utilise les mêmes hypothèses sur les nombres de points d'enregistrements correspondant aux carrés et multiplications que pour l'attaque SPA. On appelle ℓ le nombre de points correspondant à un tour de la boucle (un multiplication suivie d'un carré) :

   — écrire un script qui évalue le quatrième bit $D(g_i, s)$ du résultat de l'opération de l'étape 3 de l'algorithme 2 (une multiplication modulaire, vous pouvez utiliser la fonction correspondante que vous avez écrite pour la partie précédente). Que représente ce bit, et quelles hypothèses sur la clé doit-on faire ?

   — estimez, pour tous les points j tels que $\ell \leq j < 2 \cdot \ell - 1$, la différentielle suivante :

$$\Delta_D[j] = \frac{\sum_{i=1}^{m} D(g_i, s) \cdot \mathbf{TraceDPA}_i[j]}{\sum_{i=1}^{m} D(g_i, s)} - \frac{\sum_{i=1}^{m} (1 - D(g_i, s)) \cdot \mathbf{TraceDPA}_i[j]}{\sum_{i=1}^{m} (1 - D(g_i, s))}$$

   — après avoir tracé toutes les différentielles $\Delta_D[j]$ pour $\ell \leq j < 2 \cdot \ell - 1$, que peut-on dire ? Avez vous identifié les deux premiers bits probables de l'exposant secret d'Alice ?

   — automatisez ce processus pour les bits suivants de l'exposant d'Alice. Après avoir retrouvé cet exposant, déchiffrez le message envoyé par Alice à Bob, en tenant compte des paramètres données dans le fichier texte `DPAparameters.txt`. Joignez votre résultat de déchiffrement à votre compte-rendu.

Bonne chance !

5 Travail attendu

Le travail demandé comporte donc les tâches suivantes :

- l'implantation du générateur de nombres premiers de taille arbitraire avec Fermat, puis Solovay-Strassen, avec l'exponentiation modulaire de `GMP`, puis votre implantation quand elle sera disponible ;
- La mesure de performance, en nombre de cycles d'horloge du processeur :
  - la taille des nombres RSA : 1024 bits, 2048 bits et 4096 bits (respectivement 16, 32 et 64 mots de 64 bits, constante `SIZE` du code source, définie dans le fichier `fonctions.h`) ;
  - les options de compilation (modifier le `Makefile` comme il convient) : `-O0`, `-O1 -O2`, `-O3`, combinées éventuellement avec `-funroll-all-loops`, `-finline-functions`, ou d'autres selon votre curiosité ;
  - une analyse des différences, qui pourra s'appuyer sur un examen des fichiers exécutables avec la commande `objdump -D %code_exec | less`, la redirection vers `less` étant nécessitée par le fait qu'`objdump` affiche le résultat sur la sortie standard ; on s'attachera à bien observer la présence ou non des sauts indiquant la structure des boucles, la présence ou non des appels de fonctions (voire la présence de ces dernières dans les fichiers objets...)
- pour les deux parties sur les attaques, il est demandé :
  - l'élaboration des scripts exploitant les traces ;
  - avec la/les clé/s retrouvée/s, le déchiffrement des messages cryptés ;
  - Proposez éventuellement des contre-mesures ;

Les travaux individuels feront l'objet d'un rendu (sur la plateforme *Moodle*), comportant une archive nommée `Votre_Nom_SPA2022` contenant :

- tous les fichiers du code source ainsi que le `Makefile`, l'éventuel `Readme` ;
- les messages déchiffrés ;
- un rapport sur les mesures de performances, les difficultés rencontrées, l'influence des optimisations réalisées et une conclusion sur le travail effectué.

Enfin, il est demandé une présentation orale de 10 minutes suivi de quelques questions en classe entière, concernant ces travaux et conclusions.

Annexes

A Mesures de performances

Les mesures de performances d'un exécutable utilisent un compteur de cycles d'horloge du processeur. Les performances elles-mêmes dépendent de nombreux paramètres, dont la plateforme d'exécution. Dans le cas des systèmes basés sur les processeurs de la famille `x86-64` (`Intel`®), le jeu d'instruction comporte `rdtsc` (*ReaD Time Stamp Counter*). Cette instruction renvoie l'état du compteur d'horloge global du processeur. Cette instruction est accessible via le fichier d'en-tête `ccount.h` fourni, qui comporte plusieurs macros, dont la macro `STAMP()`, qui prend pour argument une variable de type `unsigned long long int`.

Tous les détails ont leur importance. Par exemple, les processeurs `Intel`® disposent de mécanismes d'amélioration de performances tels que le `turbo-boost`®. Ce dispositif permet à chaque cœur d'avoir sa propre horloge et la fréquence d'un cœur peut ainsi être accélérée et devenir plus rapide que celle du compteur d'horloge global accessible par `rdtsc`. Par conséquent, cette option doit normalement être désactivé pour les mesures de performances. Pour ce faire, il faut cependant disposer des droit `root` sur la plateforme.

On peut formuler la même remarque à propos de l'`hyper-threading`®, censé améliorer la gestion des `threads`, et qui facilitent le changement de contexte d'un `thread` à l'autre, avec le concept de cœurs virtuels. Ceci n'est important que dans le cas d'implantations algorithmes parallèles.

De même, les options de compilation ont une grande importance sur les performances de l'exécutable obtenu.

Voici la procédure à suivre :

1. il faut commencer par "chauffer" les caches, c'est à dire lancer l'exécution du code à tester quelques centaines de fois ;
2. il faut ensuite placer les macros `STAMP(START)` et `STAMP(STOP)` de part et d'autre du code à tester, le tout dans une boucle avec environ mille itérations ; il convient de calculer la différence `STOP-START` à chaque fois et évaluer le minimum (ou sauver chaque valeur dans un fichier).
3. il faut recommencer l'opération plusieurs fois avec des jeux de données différents (30 fois est un bon compromis) ;
4. le résultat attendu, c'est à dire le nombre de cycle estimé pour l'exécution du code à tester, c'est la moyenne des minimums obtenus pour chaque jeu de données.

Le calcul de la moyenne des minimums s'explique ainsi :

— le minimum pour chaque jeu de données correspond à l'exécution dans les conditions "idéales" de l'algorithme, c'est à dire sans perturbations dues au système d'exploitation (qui est préemptif dans la plupart des cas), avec l'état des mémoires caches optimal, avec le meilleur fonctionnement micro-architectural (prédiction de saut, pipeline, exécution *out-of-order*...)

— effectuer la moyenne de ces minimums permet d'éliminer l'influence des jeux de données : dans les calculs multi-précision que l'on effectue ici, la distribution des retenues, les soustractions de fin d'opération de Montgomery...

— l'examen des résultats en détail permet de "pister" tous ces phénomènes.

Ceci doit s'accompagner d'expérimentations en variant les options de compilation et en examinant le code exécutable au besoin (en utilisant la commande `objdump -D %code_exec | less` dans un terminal).