Calcul sécurisé – Feuille d'exercices numéro 1

Université Paris-Saclay – M1 Informatique/MINT

24 janvier 2018

Exercice 1 (*Mots de passe*)
Afin de restreindre l'accès à des donnés aux utilisateurs légitimes, certaines applications utilisent des mots de passe. Prenons l'exemple d'un fichier qui nécessite un mot de passe valide pour être ouvert. On suppose que le mot de passe a une taille de 8 octets.

1. Combien d'essais faut-il effectuer pour réaliser une recherche exhaustive sur le mot de passe ?

2. On suppose maintenant que l'algorithme de vérification de mot de passe est implémenté de la manière suivante[1]. $\tilde{P}$ désigne le mot de passe de 8 octets proposé par l'utilisateur et P désigne le mot de passe correct. La routine retourne "True" si le mot de passe proposé est valide et "False" sinon.

   Input: $\tilde{P} = (\tilde{P}[0], \ldots, \tilde{P}[7])$ (and $P = (P[0], \ldots, P[7])$)
   Output: "True" or "False"
   for $j = 0$ to 7 **do**
   if $(\tilde{P}[j] \neq P[j])$ **then Return** "False"
   end for
   Return "True"

   Montrer que cette implémentation n'est pas sûre contre un attaquant qui mesure le temps pris par la routine pour retourner le statut "True" ou "False".

Exercice 2 (*Timing attack sur un MAC*)
Soit un algorithme MAC déterministe, utilisant une clé K, et dont la sortie T fait n bits. L'algorithme de vérification V fonctionne de la façon suivante : on calcule d'abord $T' = \mathrm{MAC}_K(M)$, puis on applique les étapes suivantes :

for $i \longleftarrow 0$ to $n - 1$

[1] C'est essentiellement de cette façon qu'est implémentée – dans le langage C – la fonction *memcmp* qui compare deux zones mémoire.

if $T[i] \neq T'[i]$ output "reject" and exit
output "accept"

1. Montrer que cette implémentation est vulnérable à une "timing attack". Un attaquant qui peut soumettre des requêtes arbitraires à l'algorithme V et mesure précisément le temps de réponse de V peut forger un MAC valide de n'importe quel message M de son choix avec au plus $256 \cdot n$ requêtes à V.

2. Comment implémenteriez-vous V pour empêcher la timing attack de la question 1 ?

Exercice 3 (*Attaque de Bleichenbacher simplifiée*)
Supposons que l'on dispose d'un algorithme $\mathcal{A}$ qui :

- prend en entrée un chiffré y (obtenu par RSA) et la clé publique (n, e) utilisée par RSA ;
- donne en sortie le bit de poids faible du message clair x (tel que $y = x^e \bmod n$).

On va montrer que l'attaquant peut utiliser cet algorithme pour retrouver complètement le message clair M correspondant à un chiffré donné C.

1. Soit Ψ la fonction définie par

$$\Psi(u) = \begin{cases} 0 & \text{si } 0 \leq u \leq (n-1)/2 \ ; \\ 1 & \text{si } (n+1)/2 \leq u \leq n-1. \end{cases}$$

   Montrer qu'en utilisant l'algorithme $\mathcal{A}$ on peut obtenir $\Psi(M)$.

2. Montrer (de façon détaillée) comment – à l'aide de l'algorithme $\mathcal{A}$ – on peut retrouver complètement le message M.

Exercice 4 (*Méthode de Montgomery*)
Montgomery a mis au point en 1985 une méthode pour effectuer l'exponentiation modulaire $y := x^d \bmod n$. Les paramètres sont les suivants :

- $R := 2^\theta$, où θ est le nombre de *bits* du modulo n (typiquement, pour RSA 1024-bits, on a $\theta = 1024$);
- $d = \sum_{i=0}^{\theta-1} \delta_i . 2^i$ (représentation binaire de l'exposant d).

L'exponentiation modulaire $y := x^d \bmod n$ est alors effectuée de la manière suivante :

1. Calculer $x' := x.R \bmod n$ et $a := R \bmod n$;
2. Pour i de $\theta - 1$ à 0, faire :
   (a) Calculer $a :=$ `MontgomeryMult`(a, a);
   (b) Si $\delta_i = 1$ alors calculer $a :=$ `MontgomeryMult`(a, x').
3. Calculer $y :=$ `MontgomeryMult`$(a, 1)$.

`MontgomeryMult` est l'opération "multiplication de Montgomery". Elle utilise les paramètres suivants :

- Soit β le nombre de bits des opérations élémentaires du microprocesseur (*i.e.* $\beta = 8$ sur une plateforme 8 bits, ou $\beta = 32$ sur une plateforme 32 bits).
- Soit λ le nombre de mots de β bits du modulo n, *i.e.* $\lambda := \theta/\beta$.
- Soit $b := 2^\beta$.
- Soit $n' := -n^{-1} \bmod b$

Typiquement, on a $\beta = 8$ et donc $\lambda = 64$ (pour RSA 1024-bits sur une plateforme 8 bits).

Étant donnés $u = (u_{\lambda-1}, .., u_0)$ et $v = (v_{\lambda-1}, ..., v_0)$ (avec $0 \leq u < n$ et $0 \leq v < n$), `MontgomeryMult` calcule $a := u.v.R^{-1} \bmod n$ de la façon suivante :

1. Initialiser $a = (a_\lambda, a_{\lambda-1}, ..., a_0) := (0, 0, ..., 0)$;
2. Pour j de 0 à $\lambda - 1$, faire :
   (i) Calculer $w_j := (a_0 + u_j.v_0).n' \bmod b$;
   (ii) Calculer $a := (a + u_j.v + w_j.n)/b$.
3. Si $a \geq n$ alors $a := a - n$.

Les questions qui se posent sont les suivantes :

1. Montrer que l'algorithme `MontgomeryMult` calcule effectivement $a := u.v.R^{-1} \bmod n$.
2. Montrer que l'exponentiation modulaire avec calcule effectivement $y := x^d \bmod n$.
3. Comparer la taille mémoire (RAM) nécessaire :

- pour la méthode de Montgomery
- pour la méthode d'exponentiation modulaire "classique".

4. Expliquer pourquoi la méthode de Montgomery est potentiellement vulnérable à une "timing attack".

Exercice 5 (*Square-and-multiply-always*)
Soit $n = p \times q$ un entier, produit de deux nombres premiers p et q. On considère d un exposant secret RSA de k bits, dont l'écriture binaire est $d = d_{k-1}2^{k-1} + d_{k-2}2^{k-2} + ... + d_1 2 + d_0$ (avec $d_{k-1} = 1$).
La fonction $f : Z/n\mathbb{Z} \to \mathbb{Z}/n\mathbb{Z}$, définie par $f(x) = x^d \bmod n$, est implémentée dans une carte à puce de la manière suivante (dite "square-and-multiply-always") :

Input: x, d, n
Output: $y_0 = x^d \bmod n$
  $y_0 := x$
  for $i = k - 2$ down to 0 **do**
    $y_0 := y_0^2 \bmod n$
    $y_1 := y_0 \times x \bmod n$
    $y_0 := y_{d_i}$
  end for
  Return y_0

1. Expliquer l'avantage de la méthode "square-and-multiply-always" sur la méthode "naïve" suivante :

   Input: x, d, n
   Output: $y = x^d \bmod n$
     $y := x$
     for $i = k - 2$ down to 0 **do**
       $y := y^2 \bmod n$
       if $d_i = 1$ **then** $y := y \times x \bmod n$
     end for
     Return y

2. On se place maintenant dans l'hypothèse suivante : l'attaquant est capable de détecter (par exemple par observation des courbes de consommation électrique) que la carte effectue deux fois le même calcul. Plus précisément, si la carte calcule $u^2 \bmod n$ (à un instant t_1), puis $v^2 \bmod n$ (à un instant $t_2 > t_1$), l'attaquant n'est pas capable d'en déduire la valeur de u ni celle de v, mais est capable de dire si $u = v$.

   On suppose en outre que l'attaquant peut effectuer le calcul $x^d \bmod n$ avec les valeurs x de son choix. Montrer qu'il peut alors retrouver

la valeur de l'exposant secret d, dans le cas de l'algorithme "square-and-multiply-always" (pour cette attaque de type SPA, on pourra considérer la suite des valeurs intermédiaires, successivement pour les valeurs d'entrée x et $x' = x^2 \bmod n$)

Exercice 6 (*Attaques sur ElGamal*)
On considère le *schéma de signature ElGamal*, décrit de la façon suivante :

- Soit p un nombre premier, et g et x deux entiers aléatoires, tels que $2 \leq g \leq p-2$ et $1 \leq x \leq p-1$. La *clé privée* est alors x, et la *clé publique* est le triplet $(y \equiv g^x \bmod p,\ g,\ p)$.

- Pour *signer* un message m, le signataire tire aléatoirement un entier k premier avec $p-1$. Puis il calcule

$$w \equiv g^k \bmod p \qquad \text{et} \qquad s \equiv (H(m) - xw)/k \bmod (p-1),$$

où H est une *fonction de hachage* (transformant un message m de taille quelconque en un entier $h = H(m)$ tel que $0 \leq h \leq p-1$). La signature est alors le couple (w, s).

- Le vérifieur accepte la signature (w, s) d'un message donné m si et seulement si :

$$y^w w^s \equiv g^{H(m)} \bmod p.$$

1. Montrer qu'une signature correctement calculée par l'algorithme ElGamal est toujours acceptée par le vérifieur.

2. On suppose maintenant qu'au cours du calcul de la signature d'un message donnné m, une attaque par *injection de faute* provoque l'erreur suivante : le i-ème bit x_i de x est changé en son complément.

   Montrer que la signature (erronée) obtenue, notée (w, s'), permet à l'attaquant de trouver la valeur de x_i.

3. Généraliser au cas où l'attaquant modifie ℓ bits de x (chacun de ces bits étant changé en son complément), sans connaître la position de ces ℓ bits. Quelle est la complexité de l'attaque en fonction de ℓ et de la taille t (mesurée en bits) de p ?

4. Expliquer précisément comment une implémentation (sans protection particulière) du schéma de signature ElGamal peut être attaquée par DPA (on supposera que le calcul est effectué dans une carte à puce avec un microprocesseur 8 bits).

Exercice 7 (*Chiffrement CBC*)
Soit E_K un algorithme (symétrique) de chiffrement par blocs, dont la clé secrète est K et la taille de bloc vaut n bits. Le *mode CBC* de chiffrement (avec valeur initiale aléatoire) est défini comme ceci : pour un message M divisé en ℓ blocs de n bits $(M[1], M[2], \ldots, M[\ell])$, on commence par engendrer une valeur IV aléatoire de n bits. Le chiffré $(C[0], C[1], \ldots, C[\ell])$ est alors obtenu de la façon suivante :

- $C[0] := IV$;
- $C[i] := E_K(C[i-1] \oplus M[i])$.

On peut montrer que ce schéma est *sémantiquement sûr* contre les attaques "à clair choisi". On rappelle ce que cela signifie : si l'attaquant choisit deux messages $M_0 = (M_0[1], \ldots, M_0[\ell])$ et $M_1 = (M_1[1], \ldots, M_1[\ell])$, puis obtient le chiffré $(C_b[0], C_b[1], ..., C_b[\ell])$ de l'un des deux (b étant un bit aléatoire, non connu de l'attaquant), alors il est en pratique impossible à l'attaquant (qui peut en outre obtenir le chiffré de n'importe quel autre message de son choix) de deviner la valeur de b avec une probabilité meilleure que $1/2$.

1. On considère maintenant que la carte à puce procède de la manière suivante : lors du chiffrement (en mode CBC) d'un message $M = (M[1], M[2], \ldots, M[\ell])$, elle procède "pas à pas" de la façon suivante :

   - Demander le bloc $M[1]$
   - Renvoyer $(C[0], C[1])$
   - Demander le bloc $M[2]$
   - Renvoyer $C[2]$
   - ...
   - Demander le bloc $M[\ell]$
   - Renvoyer $C[\ell]$.

   Donner plusieurs avantages fonctionnels apportés par cette méthode (par rapport à : demander l'intégralité du message, puis faire tout le calcul, et enfin sortir tout le chiffré).

2. Montrer que, si on l'implémente de cette façon, le schéma de chiffrement CBC n'est plus sémantiquement sûr contre les attaques à clair choisi (on trouvera une façon de déterminer b dans le cas $\ell = 2$).