Université Paris-Saclay – M1 Informatique & M1 MINT

Calcul sécurisé – Rattrapage

18 juin 2018

Durée : 2h – Tous documents interdits – Aucun accès à un téléphone portable, une calculatrice, un PDA ou tout autre dispositif électronique, connectable ou non.

Questions de cours

1. Attaque par analyse de la consommation électrique sur RSA

   Soit $n = p \times q$ un entier, produit de deux nombres premiers p et q. On rappelle l'algorithme "square and multiply" pour calculer RSA :

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

   où $d = d_{k-1}2^{k-1} + d_{k-2}2^{k-2} + ... + d_1 2 + d_0$ (avec $d_{k-1} = 1$) est l'exposant secret RSA, de k bits.

   (a) Quelle est la complexité de cet algorithme (en fonction de k) ?
   (b) Quel est le principe de l'attaque SPA ? Comment permet-elle à l'attaquant de retrouver d ?
   (c) Décrire une contre-mesure contre l'attaque SPA.
   (d) Expliquer le principe de l'attaque DPA contre le RSA.
   (e) Décrire une contre-mesure contre l'attaque DPA.

2. Attaques par fautes sur RSA

   (a) Rappeler la méthode des "restes chinois" pour calculer $y = x^d \bmod n$. Quel avantage y a-t-il à l'utiliser ?
   (b) Expliquer pourquoi cette méthode est vulnérable contre une attaque par fautes. Montrer comment l'attaquant, à partir d'un résultat juste et d'un résultat faux, peut retrouver la clé secrète d.
   (c) Décrire une contre-mesure contre cette attaque par fautes.

3. Attaques de type "buffer overflow"

   (a) Quelle est la cause de ce type d'attaque ?
   (b) Expliquer le plus précisément possible comment l'attaquant peut l'exploiter.

4. Chiffrement homomorphe

   (a) Donner un exemple d'algorithme qui soit homomorphe pour une seule opération. Expliquer.
   (b) Quelle est la définition d'un algorithme de chiffrement complètement homomorphe. À quoi peut servir un tel algorithme ?

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 que l'algorithme de vérification de mot de passe est implémenté de la manière suivante. $\tilde{P}$ 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: P̃ = (P̃[0], ..., P̃[7]) (and P = (P[0], ..., P[7]))
Output: "True" or "False"
  for j = 0 to 7 do
    if (P̃[j] ≠ 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".

3. On suppose maintenant que dans l'algorithme de vérification, la comparaison est effectuée dans un ordre aléatoire. Plus précisément, soit $\mathcal{S}$ l'ensemble des permutations de l'ensemble $\{0, 1, \ldots, 7\}$. À chaque exécution, une permutation s est choisie aléatoirement dans $\mathcal{S}$ et la comparaison

```
if (P̃[j] ≠ P[j]) then Return "False"
```

   est remplacée par :

```
if (P̃[s(j)] ≠ P[s(j)]) then Return "False"
```

   (a) Pensez-vous que cette implémentation est sûre contre les "timing attacks" ? Pouvez-vous la casser ?

   (b) Pouvez-vous étendre votre attaque si un délai aléatoire est ajouté avant de retourner le statut "True" ou "False" ?

Exercice 2 (*Exemple de calcul bipartite*)
On considère, pour des entiers x et y dont l'écriture binaire fait au plus n bits (n étant fixé), la fonction $f(x, y)$ définie par

$$f(x,y) = \begin{cases} 0 \text{ si} & \text{l'écriture binaire de } x+y \text{ fait au plus } n \text{ bits} \\ 1 \text{ si} & \text{l'écriture binaire de } x+y \text{ fait strictement plus de } n \text{ bits} \end{cases}$$

On suppose que x est la valeur d'entrée que possède Alice et y la valeur d'entrée que possède Bob. Le but est de calculer $f(x, y)$ au moyen d'un protocole *bipartite*. Cela siginifie qu'Alice et Bob, à la fin du calcul, doivent avoir appris la valeur $f(x, y)$, mais rien d'autre.

1. Construire explicitement un circuit C pour la fonction f. Pour simplifier, on supposera que les deux valeurs d'entrée, x et y, ont chacune une longueur d'au plus 2 bits (c'est-à-dire que $n = 2$). Dessiner le circuit obtenu.

2. On considère le protocole "*garbled circuits*" de Yao pour évaluer de façon sécurisée la fonction f. On supposera qu'Alice construit les "garbled circuits", et que Bob les évalue. Décrire complètement le protocole de Yao dans le cas de la fonction f.

3. Expliquer comment on construirait le circuit C dans le cas général (n quelconque).