1 Protocoles de signature

Signature El-Gamal

Definition 1. *Soit G un groupe fini et g un générateur de G. Le problème du logarithme discret est, étant donné un élément $h \in G$ de trouver x tel que $g^x = h$.*

Soit p un nombre premier et h une fonction de hachage. Soit g un élément générateur de $G = (\mathbb{Z}/p\mathbb{Z})^*$. On considère le protocole de signature suivant :

- KeyGen $\rightarrow (pk = (p, b, g), sk = a \xleftarrow{\$} [0, p-2])$ avec $b = g^a$
- $\text{Sign}(m, sk) = \sigma = (m, r, s)$ avec $r = g^k \mod p$ où k est un élément de $[2, p-2]$ différent à chaque utilisation et $s = k^{-1}(h(m) - ra) \mod p-1$. Si $s = 0$ recommencer avec un autre k.
- $\text{Verif}(m, \sigma, pk)$ retourne 1 si et seulement si $0 \leq r \leq p-1$ et $g^{h(m)} = b^r r^s \mod p$

1. Quels sont les propriétés que doit vérifier une signature ?
2. Montrer que la fonction de vérification est correcte, i.e. renvoie 1 si $\sigma = \text{Sign}(m, sk)$.
3. A priori, quel élément un attaquant doit calculer pour signer. Imaginer une attaque pour trouver cet élément.
4. Quel est la taille de p à choisir pour que le système de signature ne soit pas vulnérable à cette attaque.
5. Peut-on, en terme de temps de calcul, utiliser ce système en pratique ?

Importance de k

6. Montrer que si un même élément k est utilisé deux fois il est possible de retrouver la clef secrète a.

Importance de la condition $0 \leq r \leq p-1$

Supposons que le protocole ne vérifie pas cette condition. Soit (m, r, s) une signature valide. On veut se servir de cette signature pour signer un message m'. Pour cela on calcule

$$u = h(m')h(m)^{-1} \mod p-1 \text{ et } s' = su \mod p-1$$

On trouve ensuite r' tel que $r' = ru \mod p-1$ et $r' = r \mod p$.

7. A quelle condition ces calculs sont-ils possible ?

8. Vérifier que (m', r', s') est une signature valide.

9. En supposant que la fonction h est sans collision, comparer la sécurité du protocole à celle du logarithme discret.

Importance de la fonction de hachage

Supposons que l'on utilise pas de fonction de hachage ($h(m) = m$ dans le protocole), on va réaliser une autre falsification : Soient i, j des entiers dans $[0, p-2]$, on cherche r sous la forme $g^i b^j \mod p$

10. Montrer que la relation $g^{h(m)} = b^r r^s$ est équivalente à $g^{y-is} = b^{r+js} \mod p$.
C'est en particulier le cas si $y - is$ et $r + js$ sont congrus à $0 \mod p-1$

11. A quelle condition ce système admet-il une solution ? Déterminer (r, s, y) en fonction de i et j.

12. Pourquoi cette attaque est-elle moins puissante que la précédente ?

2 Algorithmes de factorisation

2.1 Consignes générales

Le but de cette partie est de programmer différents algorithmes de factorisation. Le code est demandé en Python. L'utilisation de librairie Python ou Sage est autorisée (voir recommandée dans le cas de l'algorithme de Dixon) dans les limites du raisonnable. Pour tester vos codes : https://sagecell.sagemath.org/. Permet de tester des codes Python et Sage si le temps d'exécution est assez court.
Les algorithmes ont été plus ou moins détaillés en cours, des recherches d'optimisation peuvent être nécessaires pour factoriser tous les nombres mais il n'est pas nécessaire de résoudre tous les cas proposés pour avoir une bonne note, elle ne sera pas proportionnelle au nombre de cas résolus. En revanche la clarté du code est une donnée importante.
Le travail demandé est personnel, comme d'ailleurs pour le reste du DM. Il n'est pas interdit de communiquer entre vous ou d'utiliser internet mais le travail rendu doit être le votre. (En particulier si votre algorithme ne parvient pas à factoriser un nombre test, inutile de demander à quelqu'un d'autre la factorisation)
Il y a quelques questions en plus du code ne les oubliez pas.

2.2 Pour d'échauffer

1. Programmer un algorithme par division successive. Quels nombres arrivez-vous à factoriser en temps raisonnable (quelques secondes).
   Rappel : on cherche à factoriser des modules RSA, de la forme d'un produit de deux nombres premiers de même taille.

2. Avant de factoriser un nombre il vaut mieux vérifier qu'il n'est pas premier. Comment fonctionne la fonction isprime de Python ? Quel est le meilleur algorithme connu en terme de complexité pour tester la factorisation ? Commenter.

2.3 Pollard-Rho

- Implémenter l'algorithme de factorisation de Pollard-rho.
- Factorisez N= = 603311938244551010580282695217 53
- Factorisez N= 276474933387964773460419532857385928669681
- Analyser les variations temporelles entre deux exécutions sur la même entrée.

2.4 Méthode p-1

- Implémenter la méthode $p - 1$.
- Factorisez N= 47001798322511142685133
- Factorisez N= 29595672059884008807862646486653321192483193895814 4441

2.5 Algorithme de Dixon

- Implémenter l'algorithme de factorisation de Dixon.
- Factorisez N= 8591966237
- Factorisez N= 2251802665812493
- Factorisez N= 73786976659910426999
- Comparer les temps d'exécutions à la complexité théorique, notamment selon le choix de B.
- Comparer les différents algorithmes (de façon relativement poussée).