Séance 10 (et 11) - Travaux Pratiques
Cryptanalyse des Chiffrements Asymétriques

Yann Rotella

2026

Toute cette séance se fait en petits groupes. L'objectif est de construire son fichier python contenant des fonctions cryptographiques vues en cours, puis de programmer des cryptanalyses sur ces différentes fonctions.

Programmation des systèmes cryptographiques connus

Exercice 1. *Echange de clefs de Diffie-Hellman.*
Si ce n'est déjà fait, programmez l'échange de clefs de Diffie et Hellman vu en cours sur $\mathbb{Z}_p$.

Exercice 2. *ElGamal.*
Idem, programmez les fonctions de chiffrement et de déchiffrement du cryptosystème d'El Gamal.

Exercice 3. *RSA.*
Programmez les fonctions de chiffrement et de déchiffrement du cryptosystème de RSA.

Interlude

Exercice 4. *Génération de nombres premiers.*
Tous les algorithmes vus jusqu'à présent nécessitent la création de nombres premiers de grande taille. Cependant, nous n'avons pas d'algorithme « facile » permettant de produire des nombres premiers aléatoires. On sait cependant tester si un nombre est premier avec le test de Fermat. La stratégie consiste donc à tirer un nombre aléatoire, tester si celui-ci est premier et recommencer.

(1) Rappeler le petit théorème de Fermat.

(2) On utilise ce théorème comme test probabiliste. Programmez une fonction `test` qui prend en entrée un nombre et test si celui-ci est premier.

(3) On peut montrer que la probabilité que le test de Fermat échoue après ℓ itérations est au plus $2^{-\ell}$. Combien d'itérations doit-on effectuer afin d'avoir une probabilité d'erreur inférieure à 2^{-80} ?

(4) Programmez une génération de nombres premiers en python avec un taux de réussite suffisant à votre goût.

Cryptanalyse du log discret et de RSA

Exercice 5. *La factorisation simple.*
Soit $n = pq$ impair avec $p > q$.

(1) Programmez la méthode de factorisation par division successive en partant de $i = 2$.

(2) Vérifier que $n = t^2 - s^2$ avec $t = \frac{p+q}{2}$ et $s = \frac{p-q}{2}$.

(3) On suppose maintenant que p est très proche de q, montrer alors que t est proche de $\sqrt{n}$.

(4) En déduire une manière de factoriser deux nombres lorsque ceux-ci sont proches. Programmez cette méthode appelée méthode de Fermat.

Exercice 6. *Baby-step giant step sur log discret.*
Comme vu en cours, la sécurité de plusieurs chiffrements repose sur DDH (ou CDH).

(1) Rappelez ces problèmes formellement.

(2) Rappeler le problème du logarithme discret.

(3) Montrer pourquoi le logarithme discret est « plus dur »que CDH, qui lui-même est plus dur que DDH.

(4) On cherche maintenant à calculer le logarithme discret dans $\mathbb{Z}_p$. Pour cela, on utilise l'idée suivante :

$$x = x_1 + x_2\lceil\sqrt{p}\rfloor$$

avec x_1 et x_2 inférieurs à $m = \lceil\sqrt{p}\rfloor$.

(5) Montrer que pour tout $h \in (\mathbb{Z}_p)^\times$, il existera toujours x_1 et x_2 tels que $hg^{-x_1} = g^{x_2\lceil\sqrt{p}\rfloor} \mod p$

(6) En python, programmez l'algorithme dit de Baby-Step-Giant-Step : Calculer la liste

$$L = \{1, hg, hg^2, \dots hg^m\}$$

où g est un générateur du groupe multiplicatif.

(7) Ensuite pour chaque $i \in \{1, \dots, m\}$, tester si $(g^m) \in L$ et renvoyer la valeur d'un logarithme discret.

(8) Tester votre algorithme sur plusieurs valeurs de p générées avec votre fonction de génération de nombres premiers, différents g.

(9) D'ailleurs, pouvez-vous réfléchir à comment trouver un générateur de $\mathbb{Z}_p$?

(10) Comparez des challenges que vous réussissez et donnez les aux autres groupes. Comparez vos temps d'éxécution.

(11) Une idée pour aller plus vite ?

Exercice 7. *Pollard-rho sur RSA.*
Ici n dénote un module RSA. Nous avons vu au TD sur les fonctions de hachage comment trouver une collision sans mémoire. On peut réaliser la même technique pour RSA ! D'après le théorème des restes chinois, on sait que $\mathbb{Z}_n \approx \mathbb{Z}_p \times \mathbb{Z}_q$.

(1) Si x_1 et x_2 collisionnent dans $\mathbb{Z}_p$, que peut-on dire de $\text{pgcd}(x_1 - x_2, n)$?

(2) On utilise alors la fonction $f : \mathbb{Z}_n \to \mathbb{Z}_n$ définie par $f(x) = \lambda x$ où λ est une constante fixée. Soit x_0 un élément donné, que signifie l'existence d'une collision dans $\mathbb{Z}_p$ pour la suite définie par $u_0 = x_0$ et $u_i = \lambda u_{i-1}$? Que dire de la suite u_i ?

(3) Même question avec une suite arithmétique ?

(4) En pratique on utilise une fonction de la forme $x \mapsto x^2 + c$. En utilisant la recherche de cycle donnée aux TDs sur les fonctions de hachage ainsi que la détection avec le pgcd comme vu précédemment, trouver des collisions sur RSA.

(5) Tracer les courbes de temps d'éxécution de votre algorithme pollard-rho en fonction de la taille des entiers p (et q).

(6) Pouvez-vous dire quelle est la complexité de cet algorithme ?

(7) Faites-le sur `126765060022840279008235697491 7` (ça peut prendre un peu de temps...)

Exercice 8. *Pollard-rho sur log discret.*
Pour aller plus loin et si vous avez le temps, passez du temps ici :
`https://fr.wikipedia.org/wiki/Algorithme_rho_de_Pollard_(logarithme_discret)`

(1) Programmez l'algorithme rho de pollard sur le logarithme discret.

(2) Comparez les temps d'éxécution avec baby-step giant-step.

(3) Expliquez.