M2 CRYPTO

CRYPTANALYSE ASYMÉTRIQUE - CONTRÔLE CONTINU

1 HEURE

Exercice 1. (1) Le nombre $p = 3 \cdot 2^{201} + 1$ est premier. Le groupe $(\mathbb{Z}/p\mathbb{Z})^\times$ est-il intéressant pour instancier des cryptosystèmes fondés sur le logarithme discret ?

(2) Soit (N, e) une clé publique RSA, et un chiffré connu $c = m^e \bmod N$ d'un message inconnu m. On suppose aussi connu $m' \in \mathbb{Z}/N\mathbb{Z}$ et un entier $\alpha \geq 2$ premier avec e tels que $c^\alpha \equiv (m')^e\ [N]$. Montrer qu'on peut retrouver m efficacement.

Exercice 2 (Logarithme discret et factorisation)**.** Soit N un module RSA de factorisation inconnue $N = pq$, et $\mathbb{Z}_N^\times$ le groupe des inversibles correspondant. Le but de l'exercice est de montrer que savoir calculer efficacement des logarithmes discrets dans un grand sous-groupe cyclique G de $\mathbb{Z}_N^\times$ permet de factoriser N. On note donc g un générateur de G, et n son ordre.

(1) Soit $h_1, h_2 \in G$. Montrer que connaitre les logarithmes discrets de h_1, h_2 et $h_1 h_2$ révèle un multiple de n.

On suppose maintenant que G est un gros sous-groupe de $\mathbb{Z}_N^\times$, en demandant que $n \geq \varphi(N)/k$ pour k un petit entier. On a de plus un oracle $\mathcal{O}$ qui, pour une entrée $h \in \mathbb{Z}_N^\times$, renvoie un entier $0 \leq x < n$ tel que $h = g^x$ ou "ECHEC" si $h \notin G$.

(2) Décrire un algorithme probabiliste pour calculer n qui fait $O(k)$ appels à $\mathcal{O}$.

(3) Terminer en montrant comment factoriser N.

NE PAS PERDRE LE SUJET :
LES CHALLENGES DE CRYPTANALYSE SONT DERRIERES.

Pour les deux challenges ci-dessous, vous disposez d'une semaine, c'est-à-dire, vous devez m'envoyer les réponses **avant le mercredi 8 février 2023, minuit**. Vous les enverrez par email à

`alexandre.wallet@inria.fr`

avec l'objet "[EXAM M2] NOM PRENOM". Vous ajouterez en pièce jointe un document expliquant votre démarche (attaques, implémentations, expériences,...) et les réponses aux challenges proposés, et vos feuilles de code pour obtenir ces challenges.

Exercice 3 (Challenge 1)**.** Antoine, qui n'a pas très bien suivi le cours, veut faire générer des clés RSA personnalisées à ses camarades. Pour cela, il leur fournit un générateur de nombres premiers fonctionnant sur le modèle suivant. Le générateur commence par trouver un premier p de 512 bits. Ensuite, on calcule avec SHA-256 le hashé H_{NOM} du nom d'utilisateur et le hashé H_{PW} d'un mot de passe, tous deux au choix de l'utilisateur. L'écriture binaire de ces hash est convertis en deux entiers A_1, A_2, puis le générateur trouve ensuite un premier $q = \mathrm{NextPrime}(p + 2^{12} \cdot A_1 + A_2)$.
Les clés sont alors calculées par $N = pq$, et e est pris aléatoirement entre 2^{16} et 2^{32} jusqu'à ce que l'exposant secret d existe et soit obtenu. En suivant cette approche, Antoine a obtenu la clé publique (N, e) ci-dessous, avec $\log_2 N = 1023$:

$$\begin{aligned} N =& 75004897269665254079447790673015831536511611137418450123918838637791226225983// \\ & 760943492428925142689146081276606709637621370005743755646887443601507023195014// \\ & 543340109526850078507366153467142825500366927487797583602981085536710979097026// \\ & 379182444348082621855106940715330343646285984361141382721540367840513350823, \\ e =& 3287158109. \end{aligned}$$

Pour tester que tout fonctionne, sa camarade Bérénice lui a envoyé le chiffré

$$\begin{aligned} c =& 1940391869329091923373153732691197774388909300303729931928243196389449666685266// \\ & 33476685225193039703224768101572764403505753566352015367380137746737280079422994// \\ & 98969110370496441117245654946146906055968301684651630507611097004544085944936308// \\ & 771336289198942489467232633262821218772856004530517004573062129035412. \end{aligned}$$

et ils ont constaté que, bien que correct, le déchiffrement était sensiblement plus long que le chiffrement.

Objectif : retrouver le message qu'ils se sont envoyés.
Pour éviter les problèmes de parsing, la clé publique et le chiffré sont aussi disponibles sur `https://awallet.github.io/images/REC/ch1.txt`.

Exercice 4 (Challenge 2)**.** Antoine et Bérénice sont ennuyés car quelqu'un a (déjà) cassé leur protocole RSA ci-dessus. Ils décident donc de s'orienter vers un partage de secret à la Diffie-Hellman. Ils se mettent d'accord pour utiliser les inversibles du corps $\mathbb{Z}_p := \mathbb{Z}/p\mathbb{Z}$, où

$$p = 114491700883691901716637118175059801666128024136872697179620325608012434552801,$$

est un premier de 256 bits, car ils ont trouvé que $g = 11$ est un générateur de ce groupe ! Encore une fois, ils veulent personnaliser leurs clés, et décident de modifier la génération des données secrètes selon le schéma suivant :

$$\begin{array}{c|c|c} \text{Antoine} & \mathbb{Z}_p^\times, g & \text{Bérénice} \\ & & \\ r_a \leftarrow \mathrm{HashToRand}(str_a) & & r_b \leftarrow \mathrm{HashToRand}(str_b) \\ x \leftarrow \mathcal{U}(\mathbb{Z}_p^\times) + r_a & & y \leftarrow \mathcal{U}(\mathbb{Z}_p^\times) + r_b \\ A = g^x & \rightarrow A & \\ & B \leftarrow & B = g^y \end{array}$$

La fonction $\mathrm{HashToRand}(str)$ prend une chaine de caractères au choix de l'utilisateur, et renvoie l'entier construit à partir des 64 bits de poids faibles du hashé avec SHA-256 de cette chaîne. La notation $\mathcal{U}(\mathbb{Z}_p^\times)$ désigne un élément tiré uniformément dans $\mathbb{Z}_p^\times$. Après avoir généré des exposants secrets x, y, ils s'échangent les éléments

$$\begin{aligned} A =& 73093436829409499845695647841254502227989653773789172216753515465614879788328, \\ B =& 12462209829598893503362628034001227710380934499744362439048861708490213313355, \end{aligned}$$

et constatent qu'ils ont bien en commun $S = A^y = B^x$.

Objectif : récupérer leur secret commun S.
Pour éviter les problèmes de parsing, le premier p et les élements publiques sont aussi disponibles sur `https://awallet.github.io/images/REC/ch2.txt`.