# CRY 2023

# Laboratoire #3

15-05-2023

## Préambule

- Ce laboratoire demande de résoudre différents challenges basés sur la cryptographie asymétrique. Chaque étudiant doit rendre un rapport **individuel** avec ses propres solutions ainsi que son code. Les façons d'arriver à la solution sont libres mais doivent être expliquées dans le rapport.
- Vous pouvez par contre discuter ensemble pendant que vous résolvez les problèmes. Essayez de ne pas spoiler !
- Voici une liste de fonctions qui pourraient vous être utiles en python :
  - dans le module pycryptodome [1] : `Crypto.Util.number` : `long_to_bytes` et `bytes_to_long`. Ces fonctions existent aussi nativement depuis Python 3.2 (`from_bytes` et `to_bytes`).
  - Dans le module pycryptodome : `Crypto.PublicKey` : `RSA`
  - Dans le module pycryptodome : `Crypto.Signature` : `pss`
- Les exercices contiennent des mots tirés aléatoirement. Ne vous en offensez pas et googlez-les à vos risques et périls.
- Vous trouverez vos entrées personnalisées dans le zip `nom_prenom.zip` sur cyberlearn (dans le devoir où vous devez rendre le rapport).
- Le **rapport** et votre **code** doivent être rendus dans un zip sur cyberlearn avant le **11 juin** à 23h55.
- Il se peut que nous annoncions des erreurs sur cyberlearn/teams.

## 1 RSA avec exposant 3 (1 pt)

Nous allons étudier dans cet exercice deux attaques pouvant survenir lorsque l'exposant public $e$ de RSA vaut $e = 3$.

1. Dans un premier temps, le message `m1` est chiffré avec textbook RSA et exposant public 3. Ce message fait **moins de 600 bits**. Vous trouverez dans votre fichier de paramètres la clef publique `pk11` ainsi que le texte chiffré `c1`. Decryptez ce message. Expliquez dans votre rapport pourquoi cela fonctionne.

   **Hint** : La clef vous est fournie en format PEM. Vous pouvez l'importer en utilisant la fonction `Crypto.PublicKey.RSA.import_key()`
   **Hint2** : Le message a été encodé en entier à l'aide de la fonction `bytes_to_long`
   **Hint3** : Le message est petit, l'exposant est petit et le modulo très grand.

2. Nous allons maintenant complexifier l'attaque précédente. Cette fois, le texte clair est beaucoup plus grand. Il fait **plus de 1300 bits**. Par contre, le même message a été envoyé à trois destinataires différents. Ces destinataires ont les clefs publiques `pk12`, `pk13` et `pk14` et les textes chiffrés obtenus sont `c2`, `c3`, `c4`. Decryptez ce message et expliquez dans votre rapport pourquoi cela fonctionne.

   **Hint** : Utilisez le théorème des restes chinois pour retomber sur la problématique de l'exercice précédent.

1. https://pycryptodome.readthedocs.io

## 2 RSA-PSS (1.5 pts)

Une tour de contrôle signe les autorisation d'atterrissage à l'aide d'RSA-PSS avec SHA256 comme fonction de hashage. Vous trouverez la clef publique de la tour de contrôle `pk2` dans votre fichier de paramètres. Un bug logiciel (très bizarre) de cette tour de contrôle fait qu'elle émet de temps en temps les quatre racines carrées d'un nombre aléatoire. Vous avez réussi à capturer ces quatre racines `r2`. En plus de cela, vous connaissez la signature `s2` (en base64) d'un message `m2`.

1. Implémentez (en utilisant les fonctions déjà existantes dans pycryptodome) la vérification de signature RSA-PSS et testez cette vérification sur la signature fournie.
2. Forgez la signature d'un message vous autorisant à atterrir partout (le contenu exact du message importe peu). Expliquez votre attaque dans votre rapport.

   **Hint** : Il va falloir à nouveau utiliser le théorème des restes chinois. Utilisez les racines carrées fournies pour trouver un élément qui est un multiple de $p$ mais qui n'est pas un multiple de $q$. Une fois cet élément trouvé, utilisez-le pour récupérer la clef privée.

## 3 ECDSA (1.5 pts)

Dans le fichier `ECDSA.sage`, vous trouverez du code permettant de vérifier des signatures ECDSA sur la courbe P256.

1. Il manque la fonction permettant de signer. Codez la !
2. L'inverse modulaire est calculé d'une manière différente que celle vue en cours. Pourquoi cela fonctionne ? Quel est l'avantage de cette méthode ?
3. Cette fonction permettant de vérifier les signatures souffre d'une vulnérabilité très grave (vue récemment dans un cas réel). Exploitez la pour obtenir une signature valide du message "Je dois 10000 CHF à Alexandre Duc". Cette signature doit être valide pour la clef publique `A` donnée dans vos paramètres. Vous y trouverez aussi une signature `sig` du message "Hello World !"
4. Comment faudrait-il corriger cette fonction de vérification de signatures afin d'empêcher l'attaque précédente ?
5. Imaginez que la même vulnérabilité existe dans une fonction de vérification de signature DSA. A quoi ressemblerait votre attaque dans ce cas ?

## 4 DSA sur $\mathbb{Z}_p$ (1 pt)

Votre apprenti a bien compris comment ECDSA convertit DSA sur un groupe additif qui est une courbe elliptique. Néanmoins, il n'aime pas les courbes elliptiques et préfère le groupe additif $\mathbb{Z}_p$, pour $p$ premier.

1. Expliquez mathématiquement comment fonctionne DSA sur $\mathbb{Z}_p$. Plus précisément, décrivez la signature et la vérification.
2. Implémentez cette construction (la signature et la vérification). Vous trouverez dans votre fichier de paramètres les valeurs de $g$ et $p$. A noter que $g$ a ordre $p$. Vous aurez besoin d'une fonction de hashage mappant un message vers un entier modulo $p$ qui vous est fournie dans le fichier `dsa.sage`.
3. Cette construction est très vulnérable. Vous trouverez dans votre fichier de paramètres une clef publique $A$, un message $m$ et sa signature $(r, s)$. Cassez la construction pour signer un autre message (qui a du sens).