Travaux Pratiques - Cybersécurité

RSA

Binôme :

Auteurs :
Arnaud Rosay
Gérald Lejeune

Année 2022-2023

Objectifs des séances de TP sur le RSA

Deux séances de TP sont consacrées à l'algorithme de cryptographie asymétrique RSA. L'objectif est de mieux comprendre son fonctionnement en reprenant les éléments vus en cours et en TD, de faire une implémentation en C du textbook RSA et d'expérimenter certaines attaques. Comme pour les TP AES, il ne s'agit pas de faire une implémentation pour un produit mais de vous aider à intégrer les concepts de cet algorithme important de la cybersécurité.

Votre code devra respecter les contraintes suivantes :

— la trame devra être conforme à la trame fournie,

— les prototypes de fonctions déjà présents dans le projet doivent être utilisés sans modification,

— les conseils de codage vus en cours doivent être respectés (secure coding).

Vous devrez fournir en guise de compte-rendu unique ce document rempli, en veillant à répondre aussi bien aux aspects théoriques que pratiques. Le document devra être accompagné d'un zip contenant un répertoire avec la totalité des fichiers (code source, makefile, executable), de façon à pouvoir tester et reproduire vos résultats.

La première partie du consiste à implémenter le RSA textbook de façon simple, non optimisé. Pour la deuxième partie, certaines des fonctions seront remplacées par des versions optimisée. Des mesures de performance permettront d'illustrer l'importance de ces optimisations, particulièrement utiles dans un système embarqué.

Partie 1 - Génération des clefs RSA, chiffrement et déchiffrement

L'Algorithme 1 explique la démarche à suivre pour générer les clefs RSA publique et privée, respectivement $K_{pub} = (N, e)$ et $K_{priv} = (p, q, d)$.

Algorithme 1 : Génération des clefs RSA.

$n \leftarrow$ taille des clef en nombre de bit
$borne\_inf \leftarrow 2^{\frac{n}{2}}$
$borne\_sup \leftarrow 2^{\frac{n+1}{2}}$
$p \leftarrow$ nombre premier tel que $borne\_inf \leq p < borne\_sup$
$q \leftarrow$ nombre premier tel que $borne\_inf \leq q < borne\_sup$ et q $\neq$ p
$N \leftarrow p \times q$
$\varphi(N) \leftarrow (p-1) \times (q-1)$
`// on veut` e `ni trop petit, ni trop grand, avec un poids de Hamming faible`
$e \leftarrow$ nombre aléatoire premier avec $\varphi(N)$
$d \leftarrow e^{-1} \mod \varphi(N)$
$K_{pub} \leftarrow (N, e)$
$K_{priv} \leftarrow (p, q, d)$
return K_{pub}, K_{priv}

Cet algorithme nécessite différentes fonctions de base comme la génération d'un nombre premier, le calcul de l'inverse de e et l'exponentiation modulaire. Nous allons nous intéresser à l'implémentation naïve de ces fonctions.

Génération d'un nombre premier

Dans un crypto-système RSA, il est nécessaire de générer de grands nombres premiers aléatoires. En pratique, on tire un nombre aléatoire et on vérifie s'il convient grâce à test de primalité.

La méthode la plus simple, qu'on appellera méthode naïve, consiste à vérifier si le nombre à tester N est divisible par l'un des entiers compris entre 2 et $N-1$. Pour rendre plus rapide ce test, on réduira la plage à tous les nombres entre 2 et $\sqrt{N}$ car s'il existe p tel que $p|N$, alors $N = pq$ avec $p \leq \sqrt{N}$ ou $q \leq \sqrt{N}$. On peut encore réduire le nombre d'éléments à tester en supprimant tous les nombres pairs après que la divisibilité par 2 a échoué.

Travail à faire

Test de primalité

Donner une description en langage naturel de la méthode naïve .

Implémenter et tester votre test de primalité naïf avec les valeurs 5, 15, 561, 67829. Votre programme devra afficher le résultat du test pour chacune de ces valeurs.

Générateur de nombres premiers

Implémenter, tester la génération aléatoire de nombres premiers en mode naïf. Vous pourrez vérifier si le nombre généré est bien premier sur le site http://compoasso.free.fr/primelistweb/page/prime/liste_online_en.php.

Calcul d'un inverse dans $(\mathbb{Z}/n\mathbb{Z})^*$

La recherche d'un inverse modulaire se base sur l'utilisation du théorème 1, dit de Bachet-Bézout ou encore identité de Bézout.

Théorème 1 (Bachet-Bézout) *Soient a et b deux entiers relatifs. Si d est le PGCD de a et b, alors il existe deux entiers relatifs u et v tels que $au + bv = d$.*

Ce théorème se décline dans le cas particulier où a et b sont premiers entre eux. Dans ce cas, le Théorème 2 s'appelle le théorème de Bézout.

Théorème 2 (Bézout) *Deux entiers relatifs a et b sont premiers entre eux si et seulement s'il existe deux entiers relatifs u et v tels que $au + bv = 1$*

Par un jeu d'écriture, on obtient $au = 1 + bv$, ce qui est équivalent à $au \equiv 1 \mod b$. Par conséquent, u est l'inverse de $a \mod b$. Calculer l'inverse modulaire revient à trouver un couple de coefficients de Bézout (u, v).

Exemple pratique : recherche de l'inverse de $3 \mod 26$.
On calcule $PGCD(3, 26)$. Comme $26 = 2 \times 13$, il n'y a pas de diviseur commun à 3 et 26. On en déduit que $PGCD(3, 26) = 1$. D'après le Théorème 2, il existe u et v tels que $3 \cdot u + 26 \cdot v = 1$.
Appliquons l'algorithme d'Euclide étendu :
$2 = 26 - 8 \times 3$
$1 = 3 - 1 \times 2 = 3 - 1 \times (26 - 8 \times 3) = 9 \times 3 + (-1) \times 26$
On en conclut que $u = 9$ et $v = -1$. L'inverse de $3 \mod 26$ est donc 9.

L'algorithme 2, appelé, algorithme d'Euclide étendu est une description permettant le calcul du $PGCD(a, b)$ et d'un couple de coefficients de Bézout (u, v).

Travail à faire

Calculer en justifiant le nombre d'éléments dans $(\mathbb{Z}/26\mathbb{Z})^*$. Calculer à la main les inverses suivants :

- $5 \mod 26$
- $15 \mod 26$
- $25 \mod 26$
- $7 \mod 160$

Implémenter l'algorithme d'Euclide étendu. Utiliser le code que vous avez écrit pour retrouver

Algorithme 2 : Algorithme d'Euclide étendu.

Input : a et b, deux entiers tels que $a > b$
Output : $PGCD(a, b)$ and Bézout's coefficients u, v

```
u ← 1                                          // premier coefficient de Bézout
v ← 0                                          // deuxième coefficient de Bézout
s ← 0
t ← 1
while b > 0 do
    q ← a/b                                                        // quotient
    r ← a%b                                                          // reste
    a ← b                                  // PGCD lorsque l'exécution est finie
    b ← r
    tmp ← s
    s ← u − q × r
    u ← tmp
    tmp ← t
    t ← v − q × t
    v ← tmp
return a, u, v
```

les inverses calculés à la main. Votre programme doit afficher les résultats lorsqu'on l'exécute.

Calcul d'inverses modulaires

Chiffrement et déchiffrement

Le chiffrement et déchiffrement RSA est basé sur l'exponentiation modulaire. Le cryptosystème RSA est défini comme suit.
Soit $N = pq$ avec p et q deux nombres premiers.
Soit $m \in \mathbb{Z}_n$, le message en clair et $c \in \mathbb{Z}_n$ le message chiffré.
On a les clefs $K_{pub} = (N, e)$ et $K_{priv} = (p, q, d)$.
Les algorithmes de chiffrement et déchiffrement sont :
$c = E(K_{pub}, m) = m^e \mod N$
$m = D(K_{priv}, c) = c^d \mod N$

Exponentiation

La façon habituelle de calculer une exponentiation m^e consiste à multiplier $e - 1$ fois par m. Quand e est grand, ce qui est en générale le cas pour le RSA, cette méthode de calcul, qu'on appellera méthode naïve, conduit à des calculs très longs.

Travail à faire

Implémenter l'algorithme d'exponentiation modulaire. Vous le testerez et mesurerez son temps d'exécution avec les valeurs de la Table 1 (que votre programme devra également afficher), avec $m = 357$.

Calcul	$m^{17} \mod 533280$	$m^{257} \mod 533280$	$m^{65537} \mod 533280$	$m^{372833} \mod 533280$
Résultat				
Temps d'exéc.				

TABLE 1 – Calcul d'inverses modulaires

Implémenter les fonctions de chiffrement et déchiffrement RSA avec les fonctions que vous avez développées. Tester votre implémentation avec $m = 357$ with $(N, e) = (534749, 65537)$. La valeur chiffrée devrait être 371724. En déchiffrant 371724 avec $(p, q, d) = (809, 661, 372833)$, on obtient $m = 357$.

Partie 2 - Optimisation des algorithmes

Exponentiation

L'exponentiation rapide, aussi appelée exponentiation binaire permet un calcul optimisé en temps de l'exponentation. Deux variantes existent : l'exponentiation rapide de gauche à droite (L2R) et l'exponentiation rapide de droite à gauche (R2L). La méthode la plus ancienne - L2R - est mentionnée par Pingala, un poète et mathématicien indien aux environs de l'an -200. La méthode R2L a été inventé par Al-Kashi en 1427. Cette version est plus facile à implémenter mais peut être moins efficace que la méthode L2R.

L'idée sur laquelle repose l'algorithme de l'exponentiation rapide est très naturelle. En calculant le carré de x, puis le carré de x^2, puis le carré de x^4, on obtient x^8 en trois multiplications

(en fait des élévations au carré) au lieu des sept nécessaires avec l'algorithme naïf. Ce procédé n'est pas seulement intéressant pour les exposants k qui sont des puissance de 2. En effet, on peut décomposer le cas d'une puissance impaire $x^{2k+1} = x^{2k} \times x$. Ainsi, on peut décomposer n'importe quelle exponentiation avec seulement deux opérations : l'élévation au carré et la multiplication. C'est pour cette raison que cet algorithme est aussi appelé *Square and Multiply*.

L'algorithme se base sur l'écriture binaire de l'exposant e. On ignore tous les bits de poids fort à 0 ainsi que le premier bit à 1. Pour tous les bits restants, si le bit vaut 1, on élève au carré le résultat précédent, sinon on multiplie le résultat précédent par la valeur de la base. L'algorithme 3 détaille la procédure.

Algorithme 3 : Exponentiation rapide de gauche à droite.

Input : b (la base), e (l'exposant) et n (le module), trois entiers
Output : $b^e \mod n$
$i \leftarrow size\_in\_bits(e)$
$bit \leftarrow (e \; and \; (1 << i-1)) >> i-1$
`// recherche du premier bit à 1 en partant des poids forts`
while *bit !=1* **do**
    $i \leftarrow i-1$
    $bit \leftarrow (e \; and \; (1 << i-1)) >> i-1$
`// on passe au bit suivant (on ne prend pas en compte le premier bit à 1)`
$i \leftarrow i-1$
$bit \leftarrow (e \; and \; (1 << i-1)) >> i-1$
`// calcul de l'exponentiation`
$r \leftarrow b$
while $i > 0$ **do**
    $r \leftarrow r \times r \mod n$
    if $bit == 1$ **then**
        $r \leftarrow r \times b \mod n$
    $i \leftarrow i-1$
    $bit \leftarrow (e \; and \; (1 << i-1)) >> i-1$
return r

Travail à faire

Implémenter la méthode d'exponentiation rapide de gauche à droite.

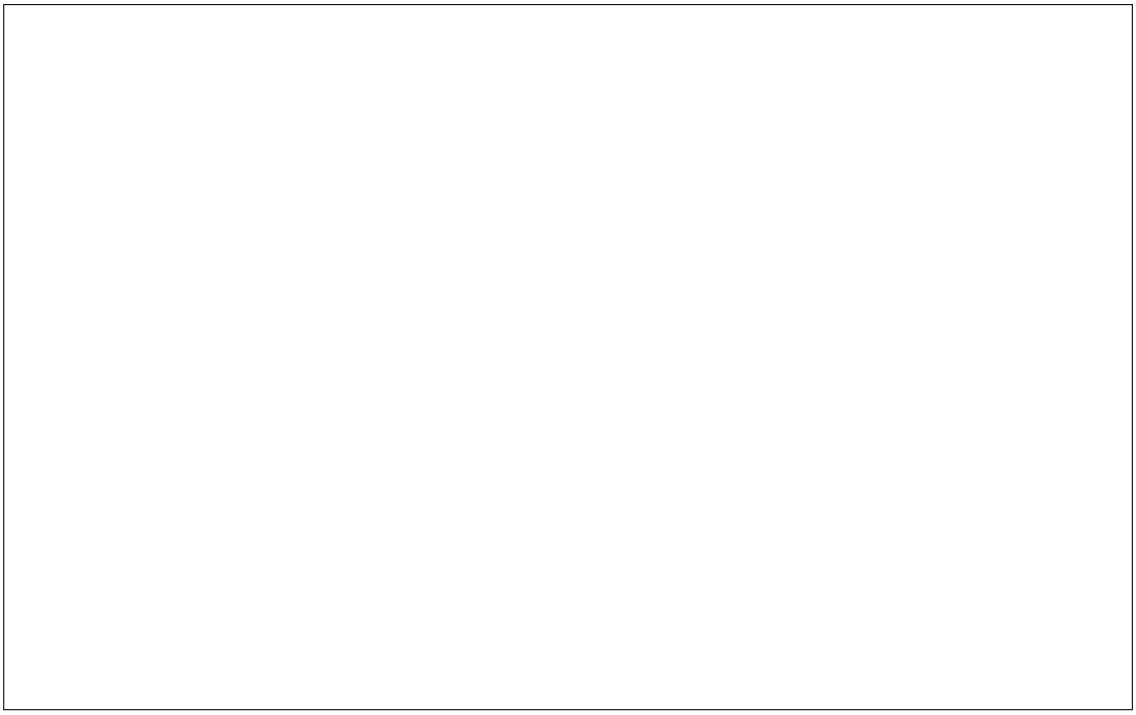
Mesurer le temps nécessaire au calcul de $m^e \mod N$ avec $m = 157$ avec les valeurs de la Table 2 (que votre programme devra également afficher).

Calcul	$m^{17} \mod 533280$	$m^{257} \mod 533280$	$m^{65537} \mod 533280$	$m^{372833} \mod 533280$
Temps d'exéc.				

TABLE 2 – Calcul d'inverses modulaires

Commenter vos observations sur les performances relatives de la méthode L2R par rap-

port à la méthode naïve (Table 1), puis indiquer ce que pouvez vous dire sur les performances de déchiffrement quand les clefs sont $K_{pub} = (N, e) = (534749, 65537)$ et $K_{priv} = (p, q, d) = (809, 661, 372833)$.



Génération d'un nombre premier

Pour déterminer si un nombre est premier, il existe des méthodes alternatives à celle implémentée dans la partie comme le test de primalité de Fermat (le plus simple), le test de Miller-Rabin ou encore de Solovay-Strassen. Pour cette partie, on se limitera au test de primalité de Miller-Rabin qu'on comparera à la méthode naïve.

Soit p un nombre premier impair que l'on peut décomposer sous la forme $n = 2^s \times d + 1$ avec d impair. Alors, $\forall a \in \mathbb{Z}_n$, on a :
soit $a^d \equiv 1 \mod p$,
soit $\exists i \in [1, s-1]$ tel que $a^{2i \times d} \equiv -1 \mod p$.

Le test de Miller-Rabin consiste pour un nombre n impair à trouver s et d tels que $n = 2^s \times d + 1$. On choisit aléatoirement a et on teste si $a^d \equiv 1 \mod p$. Si c'est le test échoue, on teste si l'une des valeurs $i \in [1, s-1]$ vérifie le test $a^{2i \times d} \equiv -1 \mod p$. Si le test échoue, alors n est un nombre composé (on appelle nombre composé un entier > 1 qui n'est pas premier) et a est appelé témoin de Miller. Si le test réussit, on dit que n est **probablement premier**.

Note importante : Si le test de Miller-Rabin indique que n est composé, alors n n'est pas premier. **La réciproque est fausse**. Dans le cas où le test n'indique pas que n est composé alors qu'il l'est, on dit que a est un menteur fort.

Exemple avec $n = 561$ (qui est divisible par 3) : on peut décomposer n avec $d = 35$ et $s = 4$ pour obtenir la forme $561 = 2^4 \times 35 + 1$.

On choisit $a = 2$. On remarque que $2^{35} \equiv 263 \mod 561$, donc le premier test échoue. on teste alors :
$i = 1$ et on obtient $263^2 \equiv 166 \mod 561$
$i = 2$ et on obtient $166^2 \equiv 67 \mod 561$
$i = 3$ et on obtient $67^2 \equiv 1 \mod 561$
Le deuxième test échoue, n est composé et $a = 2$ est un témoin de Miller.
A noter : si, pour le même exemple, on prend $a = 50$, on obtient $50^{35} \equiv 560 \mod 261$, le deuxième test réussit, illustrant que 50 est un menteur fort.

La probabilité qu'un nombre n, impair et composé, de t bits pris entre 2^{t-1} et $m = 2^t$ est de l'ordre de $\frac{ln(m)}{2\times 4^k}$. Pour éviter un faux positif, c'est à dire de déclarer à tort un nombre n comme probablement premier, on fera le test de Miller-Rabin avec k valeurs différentes de a pour garantir la fiabilité requise.

Le test de primality de Miller-Rabin est décrit par les Algorithmes 4 et 5.

Algorithme 4 : Test de primalité de Miller-Rabin.

Input : n (la valeur à tester), *prob* (la probabilité de déclarer probablement premier un nombre composé)
Output : COMPOSITE ou PROBABLY_PRIME
if $n < 2$ **then**
  return COMPOSITE
if $n == 2$ **then**
  return PROBABLY_PRIME
if $n! = 2$ *and* $n \equiv 0 \mod 2$ **then**
  return COMPOSITE
// chercher s et d tels que $n = d.2^s$
$s \leftarrow 0$
$d \leftarrow n - 1$
while $LSB(d) == 0$ **do**
  $d \leftarrow d >> 1)$
  $s \leftarrow s + 1$
// trouver le nombre de valeurs à tester pour atteindre le bon degré de confiance
$m \leftarrow 1 << (size\_in\_bits(n) + 1)$
$n\_val \leftarrow \frac{ln\left(\frac{ln(m)}{2\times prob}\right)}{ln(4)}$
// calcul de l'exponentiation
// !!! cas spécial si n très petit à considérer !!!
while *number of tested values is not enough* **do**
  $a \leftarrow$ random value $\in [2, n-2]$ // value of a shouldn't be several times
  if a *is not a Miller witness* **then**
    return COMPOSITE
return PROBABLY_PRIME

Algorithme 5 : Témoin de Miller.

Input : a (la base), s, d et n tels que $n = d.2^s + 1$, quatre entiers
Output : True (a est un témoin de Miller) or False

```
b = a^d mod n
if b ≠ 1 and (b ≠ n − 1) then
    if s == 1 then
        return True
    for i=0 ;i<s-1 ;i++ do
        b = b^2 mod n
        if b == 1 then
            return True
        if b == n − 1 then
            return False
    if b ≠ n − 1 then
        return True
return False
```

Travail à faire

Calculer k, le nombre de valeurs différentes de a à tester avec l'algorithme de Miller-Rabin pour tester un nombre de 32 bits et 3072 bits avec une fiabilité de l'ordre de 10^{-3}.

Implémenter la fonction permettant de déterminer le nombre de valeurs à tester en fonction

de la taille de la clef.

Implémenter le test de primalité de Miller-Rabin en utilisant la méthode d'exponentiation rapide. Vérifier son fonctionnement avec les valeurs de la Table 3

n	15	17	2147483437	4294967291
premier ou composé				

TABLE 3 – Durée du test de primalité

Mesurer le temps nécessaire pour tester la primalité d'un nombre N aléatoire de taille n bits pour les différents valeurs données dans la Table 4.

n	8	16	32
valeur à tester	251	65521	4294967291
durée méthode naïve			
durée méthode Miller-Rabin			

TABLE 4 – Durée du test de primalité

Aujourd'hui, une clef de 1024 bits est considérée comme non sûre. Il est conseillé d'utiliser au minimum 2048 bits, voire 3072 bits. Justifier le test de primalité à utiliser pour générer une telle clef et exprimer votre avis sur la possibilité d'implémenter la génération de clef RSA sur un système embarqué.

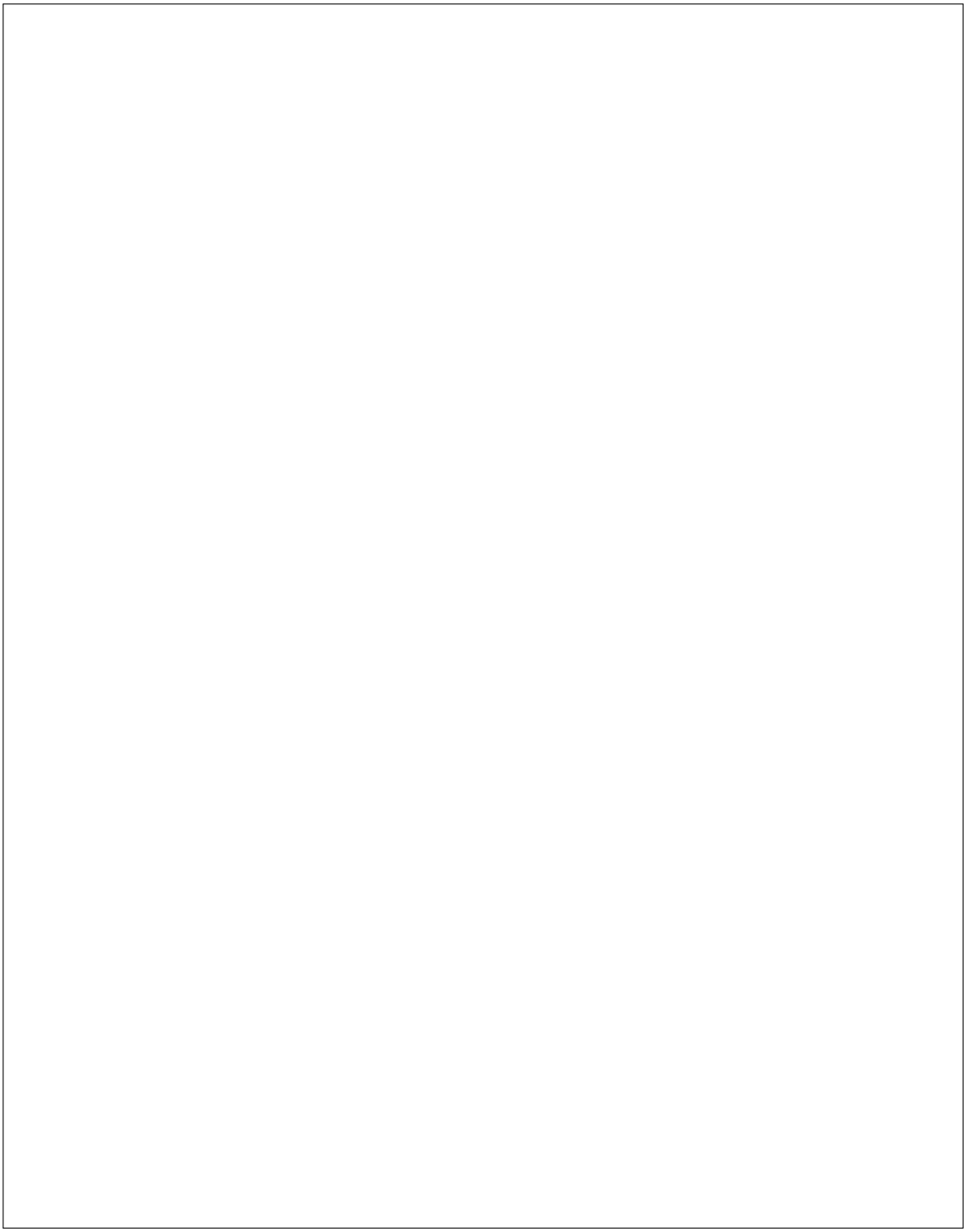
Travail à faire

Chiffrement et déchiffrement

En prenant $c = E((n, e), m) = m^e \mod n$ et $D((p, q, d), c) = c^d \mod pq$, montrer que $c^d \mod n = m \mod n$.

Implémenter les fonctions de chiffrement et déchiffrement en utilisant l'exponentiation rapide. Utiliser $K_{pub} = (N, e) = (534749, 65537)$ et $K_{priv} = (p, q, d) = (809, 661, 372833)$ pour chiffrer et déchiffrer $m = 357$. Vérifier que le déchiffrement permet de retrouver m. Votre programme doit afficher les résultats lorsqu'on l'exécute.

Quelles observations pouvez-vous tirer des mesures réalisées dans la Partie 2 ? Faites une comparaison avec les mesures de temps pour le chiffrement/déchiffrement AES.



Question Bonus - Génération de clefs RSA

A faire après avoir traité toutes les autres parties de ce TP

Implémenter la fonction de génération de clefs *RSA rsa_get_keys()* comme décrit dans l'Algorithme 1 et générer à l'aide de votre programme un couple de clefs RSA $K_{pub} = (N, e)$ et $K_{priv} = (p, q, d)$ pour une longueur de clef de $n = 64$ bits en mode naïf et en mode Miller-Rabin. Votre programme doit afficher les résultats lorsqu'on l'exécute.

Partie 3 - Attaque par module commun

Introduction

Pour éviter aux utilisateurs de le faire eux-mêmes, le service informatique, dirigé par Roger, de la société *MySoftwareCompany* déploie un logiciel pour générer les clefs RSA. Les deux premiers associés Ingrid et Denis génèrent leurs clefs grâce au logiciel. Ils obtiennent leurs clefs privées et publiques respectives. Leurs clés privées $(221, d_1)$ et $(221, d_2)$ sont gardées secrêtes sur leurs ordinateurs respectifs. Leurs clés publiques $(n_1, e_1) = (221, 11)$ et $(n_2, e_2) = (221, 7)$ sont automatiquement publiées sur un serveur accessible à tous les associés.

Transmission et interception

Chacun de leur côté, Bob, le quatrième associé, et Roger voudraient devenir actionnaire majoritaire de la société. Quelques mois après le déploiement de la messagerie chiffrée, Bob offre le premier d'acheter leurs parts à Ingrid et Denis. Ces derniers ont le même nombre de parts. Bob leur envoie le même montant M du rachat de chaque part en K€. Evidemment, l'offre est confidentielle et le message chiffré. Ingrid reçoit $C_1 = 210$ et Denis $C_2 = 58$.

Roger sait que des tractations commencent et intercepte discrètement tous les messages. C'est l'occasion de surencherir mine de rien et au plus juste pour emporter les parts convoitées.

Travail à faire

Vous devez montrer que Roger est en mesure de déchiffrer les messages destinés à Ingrid et Denis. Pour cela, vous calculerez $C_1^u \cdot C_2^v$ avec $(u, v) \in \mathbb{Z}^2$ en faisant apparaître l'identité de

Bézout afin de montrer que ce produit est égal à M.

Grâce à l'identité de Bézout que vous avez fait apparaître et en la transformant en utilisant les congruences, vous utiliserez l'implémentation du calcul d'un inverse pour obtenir la première valeur du couple (u, v), puis la deuxième valeur.

Maintenant que l'ensemble des étapes a été abordée, implémenter une fonction qui prend les deux messages chiffrés et les clefs publiques en entrées et fournit en sortie le message en clair.

Utiliser votre programme pour identifier l'offre minimale que Roger doit proposer.

Offre minimale de Roger :

Partie 4 - Attaque de Håstad

Introduction

Arthur doit donner la réponse à La question de manière secrète avant la destruction de la Terre. Juste avant l'échange, quatre de ses cinq correspondants lui ont transmis leur clef publique RSA pour l'échange. Le temps presse et Arthur ne dispose pas de calculatrice, aussi les quatre destinataires ont généré des clefs très simples et ne tiendraient, au mieux, que quelques microsecondes à une attaque par force brute. Vous attendez toujours la cinquième clef.

Chiffrement et transmission

Arthur pose ses calculs dans l'ordre de réception des clefs publiques de ses correspondants $i \in [1, 4]$ et chiffre un message M identique quelque soit i pour obtenir le message chiffré C_i.
$(N_1, e_1) = (85, 3) \longrightarrow C_1 = 53$
$(N_2, e_2) = (69, 3) \longrightarrow C_2 = 51$
$(N_3, e_3) = (451, 3) \longrightarrow C_3 = 124$
$(N_4, e_4) = (329, 5) \longrightarrow C_4 = 259$.

Pendant qu'Arthur fait ses calculs, vous vous occupez de transmettre ses résultats aux quatre destinataires.

Vous recevez enfin le numéro et la clé du cinquième sur votre portable : $(N_5, e_5) = (33, 7)$. Alors que vous tendez le téléphone à Arthur, ce dernier disparaît dans une lumière blanche ...

Attaque vertueuse

Tout est perdu si la réponse à La question ne parvient pas au cinquième destinataire. Vous n'avez pas lu le message en clair par dessus l'épaule d'Arthur et ne disposez pas de ses calculs intermédiaires. En revanche, vous avez eu accès aux messages chiffrés et aux clefs publiques. Il vous faut donc retrouver le message en clair au plus vite.

Travail à faire

Première approche

Observer les clefs utilisées et indiquer une propriété commune, utile à une attaque RSA , partagées par quelques-unes d'entre elles.

Propriété commune :

Compte tenu de la réponse précédente, indiquer comment on peut attaquer les communications utilisant ces clefs (l'attaque par force brute étant exclue). Pour cela, vous pouvez vous renseigner sur https://fr.wikipedia.org/wiki/Chiffrement_RSA.

Votre réponse donnera le nom de l'attaque ainsi que le théorème pouvant être utilisé.

Attaque possible :

Selon le thérorème applicable trouvé au dessus, si $N_1, \cdots, N_k$ sont des entiers deux à deux premiers entre eux et qu'il existe $x_1, \cdots, x_k$ tels que

$$\begin{cases} M^k \equiv C_1 \mod N_1 \\ M^k \equiv C_2 \mod N_2 \\ \vdots \\ M^k \equiv C_k \mod N_k \end{cases}$$

Alors il existeune unique solution $x = M^k$ modulo N avec $N = \prod_{i=1}^{k} N_i$ tel que

$$M^k \equiv \sum_{i=1}^{k} C_i u_i \frac{N}{N_i} \mod N$$

Développer la relation donnant M^k pour $k = 3$

Vérifier avec votre programme que les conditions d'applications du théorème sont bien respectées.

En admettant (il est possible de le démontrer) que $\forall i$ N_i et $\frac{N}{N_i}$ sont premiers entre eux, on a $v_i N_i + u_i \frac{N}{N_i} = 1 \Leftrightarrow u_i \frac{N}{N_i} = 1 - v_i N_i \Rightarrow u_i \frac{N}{N_i} \equiv 1 \mod N_i$.
Utilisez cette relation pour écrire la liste des équations permettant de trouver les valeurs u_i.

A l'aide des fonctions disponibles dans votre code, résolvez les équations, calculez M^k et enfin, la réponse à La Grande Question sur la vie, l'univers et le reste, $M \in \mathbb{N}$. Reporter vos résultats ci-dessous.

Si vous pouvez chiffrer le message avec la $5^{ième}$ clef, donner le résultat. Sinon, justifier.

Qu'aurait-il fallu mettre en place pour empêcher une telle attaque ?