Introduction à la sécurité informatique IFT3725 - IFT6271- Hiver 2024 - Devoir 1

Par: Georges Arthur Engono Essame (20202586), Nada Miled (20291032) et Shaula-Cristale Ndikumasabo (20175784)

Question 1

Le nom d'un auteur célebre a été chiffré avec RSA-*textbook*. Vous devez me dire de qui il s'agit. Il faut évidemment expliquer la technique utilisée en plus de donner la réponse correcte.

La clé publique est (N,e) et le cryptogramme est C.

N = 22047878981784695400889682760214636760309418643390230147558018532621314777497446356869293489053919885985565708884379181253393432943096559011430043731831749389041964637378860031438691586246948899528232781581165114394163377153331395526118696594345015555954774918057087351494111146348877247622316296246297040364452166392685826077219237469066741327891562245850129548813274286892784506007702311897237739253876295115147889228933259461245408315850679974550276869822634097828729982903655968487197853345154544815841542005520972492103960133396334541163621609748166256626042836442967477 48899431313503085367572 3

C = 16250093121834914738489778721283850815023220823910545716502847730384833157965639674158854530545421681485384544525983824875

Réponse :

Le nom de l'auteur qui a été chiffré est **Jorge Luis Borges**.

Sachant que pour chiffrer un message M avec RSA, nous avons que :

$$E_{(N,e)}(M) = M^e \bmod N = C \tag{1}$$

De ce fait, nous observons que si $M^e < N$, alors le cryptogramme C deviens $C = M^e$. C'est ce qui se produit dans cet example puisque $e = 3$ et $M^3 << N$.

Ainsi, nous avons que $M = C^{\frac{1}{e}}$ sans la partie $\bmod N$.

Pour trouver le message en clair M, nous n'avons qu'à calculer la racine cubique $C^{\frac{1}{3}}$.

Ci-dessous se trouve le code python qui a été utilisé pour déchiffrer le message.

```
def decode(message:int) -> str:
```

```
    decimal_to_bin = str(message)
    while len(decimal_to_bin) % 8 != 0:
        decimal_to_bin = '0' + decimal_to_bin

    m_bin = []

    for i in range(0, len(decimal_to_bin), 8):
        curr_char_bin = decimal_to_bin[i: i + 8]
        curr_char_ascii = int(str(curr_char_bin), 2)
        m_bin.append(chr(curr_char_ascii))

    return ''.join(m_bin)

def main():
    m_cubed = gmpy2.mpz(c)
    m, exact = gmpy2.iroot(m_cubed, 3)
    if exact:
        m_bin = int(bin(int(m))[2:])
        decoded_message = decode(m_bin)
        print("decoded message :", decoded_message)
    else:
        print("Error")
```

Question 2

Soit $AES_k^2(x) = AES_{k1}(AES_{k2}(x))$ avec k = k1 || k2 et k1, k2 des clés de 256 bit. Cette approche possède une faille majeure même si l'attaque nécessite une très grande quantité de mémoire vive. En supposant que vous avez une mémoire vive illimitée, expliquez comment briser AES^2. La quantité de travail de l'attaque se compte dans le nombre d'évaluations de AES. Avec la recherche exhaustive, vous avez besoin de 2^{511} évaluation pour une probabilité de succès de 50%, l'attaque que vous devez produire le réalise avec au plus 2^{257} évaluations.

Réponse :

La principale faille se trouve au niveau du fait qu'encrypter le texte clair revient à décrypter le message chiffré.

En se basant sur cette information, un cryptanalyste qui se trouve dans une situation dans laquelle non seulement le texte chiffré $AES_k^2(x) = AES_{k2}(AES_{k1}(x))$ est connu mais aussi le texte clair correspondant x est connu, alors il pourrait mettre en place une attaque meet-in-the-middle pour pouvoir se procurer les deux clés privées et ainsi lui permettre de modifier la communication entre deux parties.

Concrètement, le cryptanalyste utiliserait l'attaque meet-in-the-middle comme suit :

1. Crypter le texte clair (x) en utilisant les 2^{256} possibilités de clés tout en stockant chaque résultat (x').
2. Décrypter le cryptogramme ($AES_{k2}(AES_{k1}(x))$) en utilisant les 2^{256} possibilités de clés. On les stocke ensuite dans une structure de données appropriée.

3. Faire une comparaison de paires (x' et y') jusqu'à ce que l'on arrive à trouver un match tel que $x' = y'$. En trouvant cette paire, le cryptanalyste aurait potentiellement trouvé les clés privées.

En ce qui concerne le nombre d'évaluations, en considérant qu'on fait des cryptage et des décryptages pour comparer les différents résultats, on aurait au maximum : 2^{256} + 2^{256} = 2^{257} évaluations.

Il est aussi important de noter qu'il est possible d'obtenir plusieurs paires candidates. Dans ce cas là, il faudrait faire des tests plus poussés permettant d'obtenir la paire de clés la plus appropriée.

Question 3

Expliquer ce qui se produit lorsqu'un message chiffré avec RSA n'est pas copremier avec N si la clé publique est PK=(N,3). (M et N ne sont pas copremier). Quelle est la conséquence de chiffrer un tel message, quel est le danger réel que cela se produise, et quel est l'impact de l'utilisation du remplissage OAEP sur le problème.

Réponse :

Lorsque le message chiffré avec RSA n'est pas copremier avec N, cela peut rendre le déchiffrement ambigu : si la clé publique PK=(N,3) est utilisée, un message M non copremier avec N peut entraîner plusieurs valeurs possibles pour M après le déchiffrement. Par exemple, si nous prenons N = 35 et que nous chiffrons M = 10, le message chiffré serait C = 20. De même, si nous chiffrons un autre message M' = 15 nous obtenons C = 20. c'est-à-dire notre message chiffré est C = 20 et lorsque on veut le déchiffré on aura une ambiguïté vu qu'il peut provenir du chiffrement de l'un de deux messages différents.

Cela peut permettre à un attaquant de déduire des informations sur le message original sans avoir à le déchiffrer : le pirate peut envoyer des messages au serveur qui traite les messages chiffrés. En fonction de la façon dont le serveur réagit à ces messages, le pirate peut déduire des informations sur le message original, même s'il ne peut pas le lire directement. Donc les attaquants peuvent obtenir des informations confidentielles sans déchiffrer complètement le message.

L'utilisation du remplissage OAEP peut atténuer ce problème en introduisant un mécanisme de remplissage aléatoire qui rend les attaques de type « padding oracle » qu'on vien d'expliquer plus difficiles à executer : Le remplissage aléatoire rend chaque message chiffré unique, même si le contenu du message d'origine est le même. Cela rend plus difficile pour l'attaquant de déduire des informations sur le message original à partir des réponses du serveur, car chaque message chiffré à une structure différente.

Question 4

Vous devez déchiffrer le message suivant. Même si le masque jetable est inconditionnellement sécuritaire, il devient relativement facile à briser lorsque la clé est utilisée plus d'une fois. Vous devez récupérer le texte original qui a produit le cryptogramme suivant sachant que le même masque a été utilisé 2 fois. Le texte est un extrait de poésie française (publiquement disponible). Vous devez me donner les informations spécifiques aux textes comme le nom de son auteur, mais aussi m'expliquer comment vous avez réalisé l'attaque pour déchiffrer le document.

Réponse :

Il s'agit d'un extrait de "Tentative de description d'un dîner de têtes à Paris-France", par Jacques Prévert.

Pour déchiffrer ce message, j'ai utilisé le fait que le même masque jetable avait été utilisé deux fois et que :

- La sous-clé k_0 est de longueur 9334 = 18668 / 2
- Indice:$(M1 \oplus k) \oplus (M2 \oplus k) = M1 \oplus M2$

Des faits ci-dessous, nous comprenons que nous pouvons nous "débarrasser" de la sous-clé k en séparant le texte en deux parties m_1 et m_2 de longueur équivalente et en faisant le xor de m_1 et m_2.

Nous avons donc commencé par prendre le message original et le transformer en valeur décimal (pour facilter la lecture dans python) avec le code ci-dessous et sauvegarder le résultat dans un fichier m1.txt et m2.txt où message orginal = m1.txt + m2.txt:

```
def convert_bin_to_decimal(bin_str:str) -> list:
    return [str(int(bin_str[i:i+8], 2)) for i in range(0, len(bin_str), 8)]

''' Convert the message from binary to decimal '''
def convert_message_to_decimal():
    message = ""
    with open("q4_message.txt", "r") as file:
        for line in file:
            message += line.strip()

    message_decimal = convert_bin_to_decimal(message)

    with open("m1.txt", "w") as file:
        file.write('.'.join(message_decimal[:len(message_decimal)//2]))

    with open("m2.txt", "w") as file:
        file.write('.'.join(message_decimal[len(message_decimal)//2:]))
```

Ensuite, nous effectuons le xor de m1.txt et m2.txt et nous sauvegardons le résultat dans m1_xor_m2.txt :

```
def m1_xor_m2():
    m1, m2 = "", ""

    with open("m1.txt", "r") as file:
        m1 = file.read().strip()

    with open("m2.txt", "r") as file:
        m2 = file.read().strip()

    m1 = m1.split('.')
    m2 = m2.split('.')
```

```
    m1_xor_m2 = [str(int(m1[i]) ^ int(m2[i])) for i in range(len(m1))]

    with open("m1_xor_m2.txt", "w") as file:
        file.write('.'.join(m1_xor_m2))
```

Pour briser le message, il faut essentiellement faire de l'essaie-erreur et essayer de deviner des mots ou bouts de phrase qui pourraient se trouver dans le message original et faire le xor du "guess" avec tous les sous-ensembles consécutifs de lettres (de même longueur que le "guess") et essayer d'identifier le résultat du xor. Si le résultat ne fait aucun sens, alors le mot deviné ne fait ni partie de m1.txt ni de m2.txt. Si le mot fait du sens, alors il y a une possibilité que le mot deviné se retrouve soit dans m1.txt ou m2.txt.
Ensuite, il suffit de d'essayer d'ajouter des lettres autours du guess et de voir si le xor résultant fait encore du sens.

Pour les guess initiales, j'ai simplement consulté une liste des mots les plus populaire de la langues française. Cette liste contenait des mots comme ' de ', ' le ', ' la ', ' je ', ' que ', ' ne ', ' et ', ' plus ', ' mon ', ' qui ', ' un ', ' du ', ' au ', ' des ', ' se ', ' à ', ' les ', ' ! ', " d'un ", ' sans ', ' cœur ', ' pas ', ' fait ', ' nous ', ' est ', ' pour ', ' me ', ' comme ', ' doux ', ' sur ', ' dans ', ' avec '.

Sachant qu'un mot est précédé et suivie d'un espace, on pouvait alors effectuer la rechercher du "guess" sur l'entièreté de m1_xor_m2.txt

```
def search(word:str, m1_xor_m2: list, word2 = "") -> list:
    word_ascii = str_to_ascii(word)

    cleaned_text = []

    for i in range(0, len(m1_xor_m2) - len(word)):
        xor_val = xor(m1_xor_m2[i:i+len(word)], word_ascii)

        if not is_valid_word(xor_val):
            continue

        xor_str = ascii_to_str(xor_val.split('.'))

        is_valid = True
        if word2 != "" and word2 in xor_str:
            is_valid = True
            for l in "0123456789":
                if l in xor_str:
                    is_valid = False
                    break

        if is_valid and is_a_word(xor_str):
            print(word, "--->" , xor_str)
```

```
            print("----------------------------------------")

    return cleaned_text
```

Voici un example d'appel:

```
def main():
    m1_xor_m2  = ''
    with open("data/m1_xor_m2.txt", "r") as file:
        m1_xor_m2 = file.read().strip()

    m1_xor_m2 = m1_xor_m2.split('.')

    guessed_word = "pas de cuillère"
    search(guessed_word, m1_xor_m2, "poisson")  # "poisson" sert de filtre pour les
résultats
```

Et le résultat:

```
garthur007@Georgess-MBP src % python3 q4.py
pas de cuillère ---> ent le poisson
```

Ceci signie que les extraits "pas de cuillère" et "ent le poisson" se trouve effectivement dans le message original. À partir de cette information, nous pouvons essayer d'ajouter chaque caractères valides de la langue francaise comme l'alphabet et les signes de pontuationet le signe de l'espace devant et ensuite après le "guessed word" et vérifier si le mot résultant fait encore du sens.

Dans l'exemple ci-dessus par exemple, il est légitime de penser qu'on peux rajouter des espaces aux extrémités de l'extrait "pas de cuillère".

Le nouveau résultat devient:

```
garthur007@Georgess-MBP src % python3 q4.py
pas de cuillère  ---> lent le poisson
c
```

Maitenant, nous pouvont faire l'inverse et rechercher l'extrait "lent le poisson" et se servant du mot "cuillère" comme filtre.

Voici le résultat d'appel pour un mot qui ne se trouve pas dans le texte:

```
...
random_word ---> !2ffomKyhjd
----------------------------------------
random_word ---> panpajGw!hc
----------------------------------------
random_word ---> 6q,"hbBpc'n
----------------------------------------
random_word ---> 0'ikrjS"e77
----------------------------------------
random_word ---> b drofHe!tn
----------------------------------------
random_word ---> 6`xdmqXxecd
...
```

Avec cette méthode, j'ai pu trouver les extraits de phrase suivant:

- "l'acné des col"
- "on pour le faire "
- "" que l'homme eu"
- ", c'était pour rire."
- "parce qu'ils voient v"
- "ez que l'homme est à l'i"
- "ait pas souvent, n'ose p"

Finalement, une recherche de "l'acné des collégiens" (par déduction) sur internet m'a permis d'identifier le texte original. Comme confirmation, j'ai pris un paragraphe de ce texte que j'ai utilisé pour faire la recherche et le texte résultant était un autre paragraphe du même texte.