DM3 : Exponentiation modulaire

A rendre le vendredi 12 octobre 2021 à 8h.

Définition : Soit $M, d \in \mathbb{N}$ et $N \in \mathbb{N} \setminus \{0, 1\}$. Le calcul de M^d mod N est une exponentiation modulaire.

1. Pour $N = 55$, $d = 3$ et $M = 7$, calculez à la main et en détaillant la quantité M^d mod N.
2. Faites de même pour $N = 55$, $d = 27$ et $M = 7$. Organisez vos calculs de sorte à éviter de faire 26 produits modulo N.

Cette opération, pourtant simple, est l'opération principale du chiffrement RSA. Ce dernier intervient dans de nombreux protocoles, notamment celui permettant de payer par carte bleue de façon sécurisée et le protocole HTTPS. Evidemment, on aimerait éviter en tant qu'utilisateur de devoir attendre 2 minutes à chaque paiement par carte ou à chaque chargement d'une page web : c'est pourquoi on aimerait que le chiffrement RSA utilise un algorithme d'exponentiation modulaire performant. Mais en même temps, on aimerait aussi que ce chiffrement soit sûr : il faut donc veiller à ce que l'algorithme d'exponentiation modulaire utilisé ne permette pas de retrouver une information privée. Dans ce devoir, on s'intéresse à plusieurs algorithmes d'exponentiation modulaire qu'on évalue dans l'optique de les utiliser dans RSA.

Partie 1 *Exponentiation modulaire naïve*

Un algorithme naïf d'exponentiation modulaire est le suivant :

```
Entrée : M, d ∈ ℕ et N ∈ ℕ \ {0,1}
Sortie : M^d mod N
expo_naif (M, d, N) =
  Si d = 0
    Renvoyer 1
  Sinon
    Renvoyer (expo_naif (M, d-1, N) × M) mod N
```

1. Justifier la terminaison de cet algorithme.
2. Prouver la correction de cet algorithme vis-à-vis de sa spécification.
3. Pour $d \in \mathbb{N}$, notons c_d le nombre de multiplications (modulaires) effectuées par `expo_naif` pour un exposant d (on considèrera dans cette question que seules ces opérations sont coûteuses).
   a) A l'aide de l'algorithme, donner une relation de récurrence sur les termes de la suite $(c_d)_{d \geq 0}$.
   b) En déduire une expression explicite pour c_d pour tout $d \in \mathbb{N}$.
   c) Notons n le nombre de bits de d. Quelle est la complexité de `expo_naif` en fonction de n ?
   d) Dans le contexte de RSA et pour des raisons de sécurité, les recommandations actuelles préconisent de choisir $n = 2048$ bits au minimum. A supposer qu'on dispose d'un ordinateur pouvant effectuer 2 milliards d'opérations en une seconde, quel est l'ordre de grandeur du temps nécessaire pour faire une exponentiation modulaire avec `expo_naif` lorsque l'exposant d fait 2048 bits ?
4. Est-il envisageable d'utiliser `expo_naif` pour réaliser les exponentiations modulaires nécessaires au bon fonctionnement d'un chiffrement avec RSA ?

Partie 2 *Exponentiation modulaire binaire*

Pour tout $d \in \mathbb{N}$ de taille n bits, on note $(d_{n-1}...d_1d_0)_2$ l'écriture en base deux de d ; autrement dit, $d = \sum_{i=0}^{n-1} d_i 2^i$ avec tous les $d_i \in \{0, 1\}$. L'écriture binaire de 0 sera ainsi $(\,)_2$. Etudions l'algorithme suivant :

```
Entrée : M, d ∈ ℕ et N ∈ ℕ \ {0,1}
Sortie : M^d mod N
expo_binaire (M, d, N) =
  n ← nombre de bits de d
  R ← 1 mod N
  i ← n−1
  Tant que i ≥ 0
    R ← R² mod N
    Si d_i = 1
      R ← (R × M) mod N
    i ← i−1
  Renvoyer R
```

1. Utilisez cet algorithme afin de calculer 9^{17} mod 55 en détaillant les étapes. De combien d'élévations au carré (modulaires) et de combien de multiplications (modulaires) avez vous eu besoin ?

2. Justifier la terminaison de expo_binaire.

3. a) On note P la propriété suivante : le contenu de la variable R vaut $M^{d(i)}$ mod N où $d(i)$ est l'entier dont la représentation binaire est $(d_{n-1}...d_{i+1})_2$. Montrer que P est un invariant pour la boucle tant que de expo_binaire.

   b) En déduire la correction de cet algorithme.

4. a) Comptez rigoureusement le nombre d'élévations au carré (modulaires) en fonction du nombre de bits de d et le nombre de multiplications (modulaires) en fonction du nombre de bits égaux à 1 dans l'écriture binaire de d.

   b) En admettant que les seules opérations coûteuses dans expo_binaire sont les multiplications et mises au carré modulaires, en déduire un majorant pour la complexité de cet algorithme en fonction de n, le nombre de bits de l'exposant d. Qu'en pensez-vous ?

Remarquez que expo_binaire est la variante modulaire de l'algorithme expo_rapide que nous avons vu en cours. Tout comme nous avions écrit expo_rapide de manière récursive, il est possible de réécrire expo_binaire dans un style plus fonctionnel. Dans l'algorithme suivant toutes les divisions sont des divisions entières.

```
Entrée : M, d ∈ ℕ et N ∈ ℕ \ {0, 1}
Sortie : M^d mod N
expo_binaire_rec (M, d, N) =
  Si d = 0
    Renvoyer 1
  Si d mod 2 = 0
    Renvoyer expo_binaire_rec(M^2 mod N, d/2, N)
  Sinon
    Renvoyer (expo_binaire_rec(M^2 mod N, d/2, N) × M) mod N
```

5. Etudiez la terminaison de l'algorithme expo_binaire_rec.

6. Prouvez rigoureusement la correction de cet algorithme.

Constatez que cet algorithme fait exactement le même nombre de multiplications et mises au carré que sa version itérative : sa complexité sera donc la même.

Partie 3 *Attaques par canaux auxiliaires*

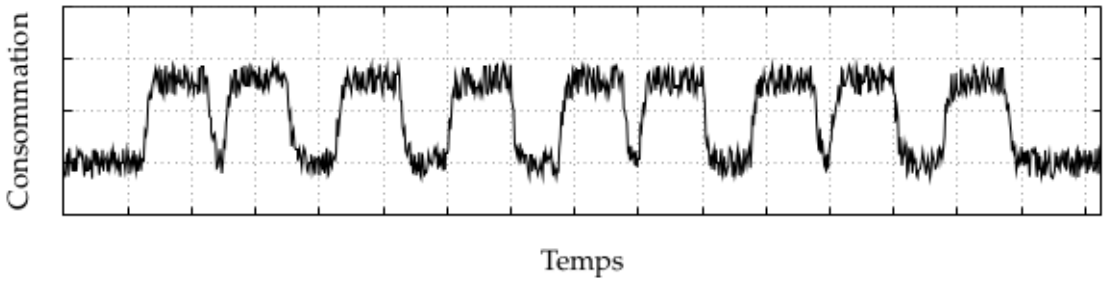
Lors d'un paiement par carte bancaire, la carte doit s'authentifier auprès du terminal de paiement, autrement dit, prouver qu'elle est une carte légitime, rattachée à une banque certifiée. Pour cela, chaque carte possède un identifiant (en langage cryptographique, on parle de clé) qu'on appelle ici d. Cet identifiant doit absolument être gardé secret : si quelqu'un découvre l'identifiant d'une carte bancaire C, il lui suffirait de le copier sur une fausse carte bancaire et faire tous les paiements qu'il souhaite au nom du propriétaire légitime de C. Sans rentrer dans les détails du protocole d'authentification (basé sur RSA), ce dernier demande à la carte de calculer une quantité $S = M^d$ mod N où M, N sont des entiers. Cette quantité est ensuite transmise au terminal qui fait lui même un calcul à partir de S afin de vérifier que la quantité qu'on lui a transmise est cohérente vis-à-vis du protocole : si c'est le cas, le terminal accepte de continuer d'échanger des informations avec la carte, sinon, il refuse. Pour calculer M^d mod N, la puce de la carte utilise l'alimentation électrique du terminal. Or, un commerçant malveillant peut placer dans son terminal de paiement un petit dispositif permettant de mesurer très précisément le courant électrique qui alimente la puce de la carte, à la manière d'un oscilloscope.

Imaginons que l'algorithme d'exponentiation modulaire utilisé par la carte pour calculer M^d mod N soit l'algorithme expo_binaire.

1. Si δ est un des bits de l'exposant secret d, dire combien d'opérations modulaires (mises au carré ou multiplications) sont effectuées lors d'une itération de la boucle tant que de expo_binaire en fonction de la valeur de δ.

Il se trouve qu'une multiplication ou une élévation au carré (modulaires) est une opération coûteuse : aussi, lorsque la puce effectue une des ces opérations, elle consomme plus d'électricité que durant le reste de l'exécution du programme.

2. La trace de consommation électrique de la puce ci-dessous a été mesurée lors du calcul d'une exponentiation binaire pour un exposant d de 6 bits. Retrouvez la valeur de l'exposant secret d en expliquant votre démarche.



3. Que penser de l'algorithme expo_binaire dans ce contexte ?

Afin d'éviter l'attaque précédente (dite, attaque par analyse simple de consommation), on modifie ainsi l'algorithme expo_binaire :

Entrée : $M, d \in \mathbb{N}$ et $N \in \mathbb{N} \setminus \{0,1\}$
Sortie : M^d mod N
expo_cache (M, d, N) =
  n ← nombre de bits de d
  R ← 1
  Z ← 1
  i ← n−1
  Tant que $i \geq 0$
    R ← R^2 mod N
    Si $d_i = 1$
      R ← (R × M) mod N
    Sinon
      Z ← (R × M) mod N
    i ← i−1
  Renvoyer R

4. Estimez la complexité de expo_cache en fonction de n (le nombre de bits dans d) en supposant que les seules opérations coûteuses sont les multiplications et mises au carré modulaires. Comparez là à celle de expo_binaire et concluez.

5. Expliquez l'intérêt de calculer Z alors que cette quantité n'est pas utilisée dans la suite du calcul. Ce nouvel algorithme est-il vulnérable aux attaques par analyse simple de consommation ?

Malheureusement, un commerçant malveillant peut faire plus que mesurer la consommation de la puce de la carte : il peut introduire entre la carte et le terminal un dispositif qui contrôle la tension d'alimentation de la puce de la carte. Il lui est alors possible de baisser brutalement cette tension afin de faire faire des erreurs de calcul à des moments précis de l'exécution d'une exponentiation modulaire par la puce. Une telle attaque est appelée "attaque par injection de fautes".

6. Expliquez comment l'injection de fautes permet de détecter des opérations inutiles dans l'exécution d'un algorithme. Déduisez-en un moyen de déjouer la contre-mesure introduite dans l'algorithme expo_cache.

Partie 4 *Multiplication de Montgomery*

Dans les années 80, Montgomery proposa l'algorithme suivant (en anglais cet algorithme s'appelle fréquemment *Montgomery multiplication* ou *Montgomery powering ladder*. Cet algorithme est d'ailleurs adaptable à de nombreuses opérations arithmétiques). On rappelle que $(d_{n-1}...d_1d_0)_2$ est l'écriture binaire d'un entier d de n bits.

Entrée : $M, d \in \mathbb{N}$ et $N \in \mathbb{N} \setminus \{0,1\}$
Sortie : M^d mod N
Montgomery (M, d, N) =
  n ← nombre de bits de d
  T_0 ← 1
  T_1 ← M mod N
  i ← n−1
  Tant que i ≥ 0
    T_{1-d_i} ← T_0 × T_1 mod N
    T_{d_i} ← $T_{d_i}^2$ mod N
    i ← i−1
  Renvoyer T_0

1. Quelle est la complexité de cet algorithme en fonction de n (le nombre de bits de d) ?

2. a) Montrer qu'avant chaque itération de la boucle tant que, on a toujours $T_1 = T_0 \times M \bmod N$.

   b) En reprenant la notation de la question 3a) de la partie 2, on note P la propriété suivante : le contenu de la variable T_0 vaut $M^{d(i)} \bmod N$. Montrer que P est un invariant pour la boucle tant que de Montgomery.

   c) Etudiez la correction totale de cet algorithme.

3. L'algorithme Montgomery est-il vulnérable aux attaques par injection de fautes ? Est-il raisonnable d'utiliser cet algorithme pour effectuer les exponentiations modulaires demandées par RSA ?

Morale de l'histoire : Selon le contexte, il ne suffit pas d'être un algorithme rapide, correct et qui termine pour être un "bon" algorithme...