Université Paris Cité
UFR de Mathématiques

Initiation à la Recherche

Le Cryptosystème RSA : Fondements Mathématiques et Analyse de Sécurité

Rapport rédigé par

MANSOURI Walaa

Encadrante : Mireille Fouquet

Année universitaire 2025–2026

Résumé

Ce rapport présente une étude du cryptosystème RSA (Rivest, Shamir et Adleman, 1978), l'un des algorithmes de cryptographie à clé publique les plus influents de l'histoire de la sécurité informatique.

Après avoir introduit les principes fondamentaux de la cryptographie classique et moderne : chiffrements par substitution, par permutation,par flot, chiffrement symétrique AES, échange de clés Diffie-Hellman et fonctions de hachage, nous développons les outils mathématiques nécessaires à la compréhension de RSA : arithmétique modulaire, algorithme d'Euclide étendu, fonction indicatrice d'Euler, théorème d'Euler, théorème des restes chinois et exponentiation modulaire rapide. Nous décrivons ensuite en détail le fonctionnement de l'algorithme RSA, depuis la génération des clés jusqu'aux mécanismes de chiffrement, de déchiffrement, de signature numérique et d'optimisation par RSA-CRT.

Une attention particulière est accordée à l'analyse de la sécurité de RSA. Nous étudions le lien entre la sécurité du système et la difficulté de factoriser de grands entiers, en présentant trois algorithmes de factorisation : la méthode ρ de Pollard, le Crible quadratique et le Crible algébrique. Nous analysons ensuite plusieurs attaques connues : l'attaque par module commun, l'attaque par masquage exploitant l'homomorphisme multiplicatif de RSA, et l'attaque de Bleichenbacher sur le bourrage PKCS#1, qui illustre comment une simple réponse binaire d'un serveur peut permettre de retrouver un message chiffré après environ 10^6 requêtes.

Enfin, nous discutons de la sécurité pratique de RSA face aux méthodes modernes de factorisation et à l'importance d'une implémentation correcte, illustrée par l'attaque de Bleichenbacher qui a directement compromis les protocoles SSL et TLS en 1998.

Table des matières

1 Introduction à la cryptographie

1.1 Qu'est-ce que la cryptographie ?

La cryptographie est la science qui consiste à transformer un message *lisible* (appelé **texte en clair** ou *plaintext*) en un message *incompréhensible* (appelé **texte chiffré** ou *ciphertext*), de sorte que seul un destinataire autorisé puisse le déchiffrer [3].

L'objectif principal de la cryptographie est de permettre à deux entités de communiquer sur un canal public en présence d'un adversaire. Même si celui-ci peut observer les échanges, il ne doit pas être en mesure de comprendre les messages transmis ni de les modifier sans être détecté.

Cette discipline constitue aujourd'hui un élément essentiel de la sécurité des systèmes informatiques et des communications numériques. Elle est notamment utilisée pour protéger les échanges sur Internet, les transactions bancaires et les données personnelles.

1.2 Quelques notions fondamentales

Définition 1.1 (Système cryptographique)**.** *Soient $\mathcal{P}$ l'espace des textes en clair, $\mathcal{C}$ l'espace des textes chiffrés, et $\mathcal{K}$ l'espace des clés. Un* système cryptographique *est un couple de fonctions :*

$$E_k : \mathcal{P} \to \mathcal{C} \quad \textit{(chiffrement)} \qquad D_k : \mathcal{C} \to \mathcal{P} \quad \textit{(déchiffrement)}$$

tels que $D_k(E_k(m)) = m$ pour tout message $m \in \mathcal{P}$.

De plus, pour chaque clé $k \in \mathcal{K}$, la fonction E_k doit être injective *: deux messages distincts ne peuvent pas produire le même texte chiffré, ce qui garantit que le déchiffrement D_k est bien défini.*

1.3 Exemples de chiffrement

Exemple 1.1 (Chiffrement de César)**.** *Les chiffrements par substitution constituent l'une des plus anciennes méthodes de cryptographie. Leur principe consiste à remplacer chaque lettre du message clair par une autre lettre selon une règle déterminée.*

Pour décrire ces transformations, on associe généralement les lettres de l'alphabet aux éléments de l'ensemble $\mathbb{Z}_{26}$:

$$A \leftrightarrow 0, \quad B \leftrightarrow 1, \quad \ldots, \quad Z \leftrightarrow 25.$$

Les calculs sont alors effectués modulo 26, ce qui permet de revenir au début de l'alphabet après la lettre Z. Le chiffrement de César est un chiffrement par substitution

dans lequel chaque lettre est remplacée par celle située trois positions plus loin dans l'alphabet.

En identifiant les lettres de l'alphabet aux éléments de $\mathbb{Z}_{26} = \mathbb{Z}/26\mathbb{Z}$*, la transformation d'une lettre* x *peut s'écrire*

$$y \equiv x + 3 \pmod{26}.$$

Le déchiffrement consiste alors à effectuer l'opération inverse :

$$x \equiv y - 3 \pmod{26}.$$

Par exemple,

$$R \equiv 17 \mapsto 20 \equiv U,$$

$$S \equiv 18 \mapsto 21 \equiv V,$$

$$A \equiv 0 \mapsto 3 \equiv D.$$

Ainsi, le mot `RSA` *est chiffré en* `UVD`*.*

Bien que simple à mettre en œuvre, ce système est aujourd'hui considéré comme non sécurisé. En effet, il ne possède que 26 clés possibles, ce qui permet de retrouver la clé par une recherche exhaustive en essayant successivement tous les décalages.

Plus généralement, les chiffrements par substitution monoalphabétique admettent jusqu'à $25!$ *permutations possibles de l'alphabet. Malgré cet espace de clés important, ils restent vulnérables à l'analyse fréquentielle. Cette attaque exploite le fait que certaines lettres apparaissent plus fréquemment que d'autres dans une langue donnée. Par exemple, en français, la lettre* `E` *est généralement la plus fréquente. Ainsi, la lettre la plus fréquente du texte chiffré est souvent associée à* `E`*, ce qui permet progressivement de reconstituer la substitution utilisée et de retrouver le message en clair.*

Exemple 1.2 (Chiffrement de Vigenère)**.** *Le chiffrement de Vigenère est une généralisation du chiffrement de César. Au lieu d'utiliser un unique décalage pour toutes les lettres, il utilise un* mot-clé *dont les lettres déterminent successivement différents décalages. Il s'agit donc d'un chiffrement* polyalphabétique.

En identifiant les lettres de l'alphabet aux éléments de $\mathbb{Z}_{26}$*, considérons le mot-clé* `CLE`*. Celui-ci correspond aux décalages :*

(2,11,4).

Pour chiffrer le message `BONJOUR`*, on répète le mot-clé autant de fois que nécessaire :*

Message	*B*	*O*	*N*	*J*	*O*	*U*	*R*
Clé	*C*	*L*	*E*	*C*	*L*	*E*	*C*

En effectuant l'addition modulo 26 *lettre par lettre, on obtient :*

Message	1	14	13	9	14	20	17
Clé	2	11	4	2	11	4	2
Chiffré	3	25	17	11	25	24	19

ce qui correspond au texte chiffré :

[`DZRLZYT.` *]*

Pour déchiffrer le message, il suffit de soustraire les mêmes décalages modulo 26.

Contrairement au chiffrement de César, une même lettre du texte en clair n'est pas toujours remplacée par la même lettre dans le texte chiffré. Cette propriété rend l'analyse fréquentielle plus difficile et améliore la sécurité du système.

Exemple 1.3 (Le chiffrement de Hill)**.** *Le chiffrement de Hill est un système cryptographique polyalphabétique introduit par Lester S. Hill en 1929. Soit* m *un entier positif et définissons*

$$P = C = (\mathbb{Z}_{26})^m.$$

L'idée est de chiffrer des blocs de m *lettres à l'aide de multiplications matricielles modulo 26.*

Soit K *une matrice clé de taille* $m \times m$*. Pour un vecteur de texte clair* $x \in (\mathbb{Z}_{26})^m$*, le chiffrement est défini par*

$$y = xK \pmod{26}.$$

Le déchiffrement s'effectue à l'aide de la matrice inverse (lorsqu'elle existe) :

$$x = yK^{-1} \pmod{26}.$$

Une matrice K *est inversible modulo 26 si et seulement si* $\det(K)$ *est inversible dans* $\mathbb{Z}_{26}$*.*

Soit la matrice clé

$$K = \begin{pmatrix} 5 & 2 \\ 7 & 3 \end{pmatrix}.$$

On calcule

$$\det(K) = 5 \cdot 3 - 2 \cdot 7 = 15 - 14 = 1 \equiv 1 \pmod{26}.$$

Comme $1^{-1} \equiv 1 \pmod{26}$*, l'inverse est*

$$K^{-1} = \begin{pmatrix} 3 & 24 \\ 19 & 5 \end{pmatrix}.$$

Chiffrons maintenant le texte `JULY`*. En utilisant l'encodage standard* $A = 0, B =$

$1, \ldots, Z = 25$, *on obtient :*

$$JU = (9, 20), \quad LY = (11, 24).$$

Chiffrement :

$$(9, 20)\begin{pmatrix} 5 & 2 \\ 7 & 3 \end{pmatrix} = (9 \cdot 5 + 20 \cdot 7,\ 9 \cdot 2 + 20 \cdot 3) \equiv (3, 0) \pmod{26},$$

$$(11, 24)\begin{pmatrix} 5 & 2 \\ 7 & 3 \end{pmatrix} = (11 \cdot 5 + 24 \cdot 7,\ 11 \cdot 2 + 24 \cdot 3) \equiv (15, 16) \pmod{26}.$$

Ainsi, le texte chiffré est

$$(3, 0)(15, 16) \Rightarrow \texttt{DAPQ}.$$

Déchiffrement :

$$(3, 0)\begin{pmatrix} 3 & 24 \\ 19 & 5 \end{pmatrix} = (9, 20), \quad (15, 16)\begin{pmatrix} 3 & 24 \\ 19 & 5 \end{pmatrix} = (11, 24).$$

On retrouve donc le texte original `JULY`.

Exemple 1.4 (Le chiffrement par permutation)**.** *Le chiffrement par permutation (ou chiffrement par transposition) consiste à conserver les lettres du message en clair tout en modifiant leur position selon une permutation donnée. Contrairement aux chiffrements par substitution, les lettres ne sont pas remplacées ; seul leur ordre est réorganisé.*

Soit une permutation p de l'ensemble $\{1, 2, \ldots, m\}$. Pour un bloc de texte

$$(x_1, x_2, \ldots, x_m),$$

le chiffrement est défini par

$$e_p(x_1, x_2, \ldots, x_m) = (x_{p(1)}, x_{p(2)}, \ldots, x_{p(m)}).$$

Le déchiffrement utilise la permutation inverse p^{-1} afin de retrouver l'ordre initial des caractères.

Considérons des blocs de longueur $m = 5$ et la permutation suivante :

$$p = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 3 & 5 & 1 & 2 \end{pmatrix}.$$

Le texte clair est :

SELLS

Les positions des lettres sont :

$$(S, E, L, L, S).$$

En appliquant la permutation p, on obtient :

$$(L, S, E, S, L).$$

Le bloc chiffré est donc :

LSESL.

Pour déchiffrer, on utilise la permutation inverse

$$p^{-1} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 5 & 2 & 1 & 3 \end{pmatrix},$$

ce qui permet de retrouver le texte original :

SELLS.

Ainsi, le chiffrement par permutation modifie uniquement l'ordre des caractères sans changer leur valeur.

Exemple 1.5 (Les chiffrements en chaine)**.** *Les chiffrements par flot constituent une alternative aux chiffrements par blocs. Au lieu de chiffrer des blocs de plusieurs caractères, ils chiffrent les données caractère par caractère à l'aide d'une suite de clés appelée flot de clés (keystream).*

Soient :

$$x = (x_1, x_2, \dots)$$

le texte clair,

$$z = (z_1, z_2, \dots)$$

le flot de clés, et

$$y = (y_1, y_2, \dots)$$

le texte chiffré.

Le chiffrement est défini par :

$$y_i = (x_i + z_i) \bmod 26,$$

et le déchiffrement par :

$$x_i = (y_i - z_i) \bmod 26.$$

On distingue deux grandes catégories de chiffrements par flot :

- ***Chiffrement par flot synchrone** : le flot de clés est généré uniquement à partir de la clé secrète. L'émetteur et le récepteur doivent rester synchronisés pour utiliser la même suite de clés.*
- ***Chiffrement par flot asynchrone** : le flot de clés dépend non seulement de la clé secrète, mais également de certains caractères précédemment chiffrés ou déchiffrés. Un exemple classique est le chiffrement à clé automatique (Autokey Cipher).*

Considérons un chiffrement par flot synchrone.
Le texte clair est :

SALUT

En utilisant la correspondance $A = 0, \ldots, Z = 25$, on obtient :

$$(18, 0, 11, 20, 19).$$

Supposons que le flot de clés généré soit :

$$(4, 7, 2, 5, 8).$$

Le chiffrement est obtenu en effectuant l'addition modulo 26 :

$$(18 + 4,\ 0 + 7,\ 11 + 2,\ 20 + 5,\ 19 + 8)$$

$$= (22, 7, 13, 25, 1) \bmod 26.$$

Ce qui correspond aux lettres :

WHNZB.

Le texte chiffré est donc :

$$\boxed{\textit{WHNZB}}.$$

Pour déchiffrer, on soustrait les mêmes valeurs du flot de clés :

$$(22 - 4,\ 7 - 7,\ 13 - 2,\ 25 - 5,\ 1 - 8) \equiv (18, 0, 11, 20, 19) \pmod{26}.$$

On retrouve ainsi le texte original :

$$\boxed{SALUT}.$$

1.4 Exemple de cryptanalyse : l'analyse fréquentielle

L'analyse fréquentielle est l'une des plus anciennes techniques de cryptanalyse. Elle repose sur l'observation que certaines lettres apparaissent plus fréquemment que d'autres dans une langue donnée. Par exemple, en français, la lettre `E` est généralement la plus fréquente, suivie notamment des lettres `A`, `S`, `I` et `N`.

Dans un chiffrement par substitution monoalphabétique, chaque lettre est toujours remplacée par le même symbole. Les fréquences des lettres sont donc conservées dans le texte chiffré. Un cryptanalyste peut alors comparer les fréquences observées dans le texte chiffré avec celles de la langue utilisée afin d'établir des correspondances probables entre les lettres.

Par exemple, si une lettre apparaît beaucoup plus souvent que les autres dans le texte chiffré, elle peut être associée à la lettre `E`. En poursuivant cette analyse sur les autres lettres ainsi que sur les groupes de lettres fréquents, il devient possible de reconstituer progressivement la substitution utilisée et de retrouver le message en clair.

Cette méthode est particulièrement efficace contre les chiffrements par substitution classiques, ce qui explique pourquoi ces systèmes sont aujourd'hui considérés comme insuffisamment sécurisés.

2 Cryptographie moderne et fondements de RSA

2.1 Chiffrement symétrique

Le chiffrement symétrique repose sur l'utilisation d'une même clé pour le chiffrement et le déchiffrement des données. L'émetteur et le destinataire doivent donc partager préalablement cette clé secrète.

Parmi les algorithmes de chiffrement symétrique les plus connus figurent le DES (*Data Encryption Standard*) et l'AES (*Advanced Encryption Standard*). Le DES, adopté en 1977, a longtemps constitué le standard de référence pour la protection des données. Il chiffre des blocs de 64 bits à l'aide d'une clé effective de 56 bits et applique une succession de 16 rondes de transformations.Toutefois, l'augmentation de la puissance de calcul et des techniques d'attaque a progressivement rendu possible la recherche exhaustive de sa clé, ce qui a conduit à son remplacement.

L'AES, sélectionné par le NIST en 2001 pour succéder au DES, est aujourd'hui l'algorithme de chiffrement symétrique le plus utilisé. Il traite les données par blocs de 128 bits et accepte des clés de 128, 192 ou 256 bits. Contrairement au DES, l'AES

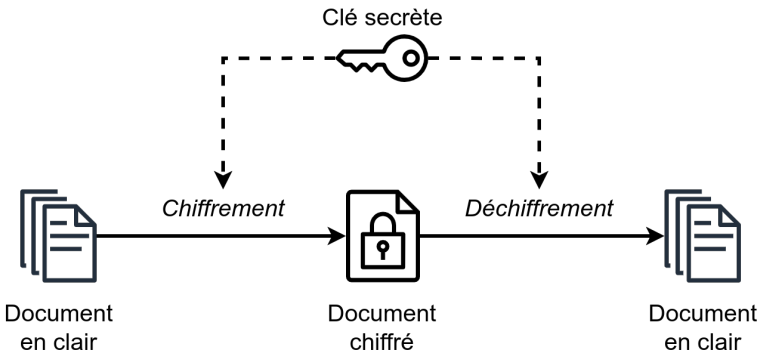

FIGURE 1 – Principe général du chiffrement symétrique

ne repose pas sur un schéma de Feistel mais sur plusieurs transformations successives appliquées aux données : substitution des octets (*SubBytes*), permutation des lignes (*ShiftRows*), mélange des colonnes (*MixColumns*) et ajout d'une sous-clé (*AddRoundKey*). Selon la taille de la clé utilisée, l'algorithme effectue 10, 12 ou 14 rondes. Grâce à ses clés plus longues et à sa conception moderne, l'AES offre aujourd'hui un niveau de sécurité largement supérieur à celui du DES.

Afin d'illustrer le fonctionnement de l'AES, considérons un exemple simplifié. Dans l'algorithme réel, les données sont organisées sous la forme d'une matrice de 16 octets appelée *State*. Une ronde de l'AES applique plusieurs transformations successives à cette matrice.

Supposons que l'on dispose de la ligne suivante :

$$\begin{bmatrix} 12 & 7 & 4 & 9 \end{bmatrix}$$

Lors de l'étape *ShiftRows*, les éléments sont décalés vers la gauche. Après un décalage d'une position, on obtient :

$$\begin{bmatrix} 7 & 4 & 9 & 12 \end{bmatrix}$$

Une sous-clé peut ensuite être combinée aux données lors de l'étape *AddRoundKey*. Si la sous-clé est :

$$\begin{bmatrix} 5 & 3 & 10 & 6 \end{bmatrix},$$

l'opération XOR donne :

$$7 \oplus 5 = 2,\ 4 \oplus 3 = 7,\ 9 \oplus 10 = 3,\ 12 \oplus 6 = 10.$$

On obtient alors :

$$\begin{bmatrix} 2 & 7 & 3 & 10 \end{bmatrix}.$$

Dans l'AES réel, ces opérations sont réalisées sur une matrice de 4×4 octets et répétées au cours de plusieurs rondes. L'enchaînement de substitutions, de permutations et de combinaisons avec des sous-clés permet d'assurer la confidentialité des données.[3, 8]

2.2 Le problème de l'échange de clés

Dans un système de chiffrement *symétrique*, Alice et Bob utilisent la même clé secrète pour chiffrer et déchiffrer les messages. La sécurité du système repose entièrement sur la confidentialité de cette clé.

Cependant, une difficulté majeure apparaît : avant toute communication sécurisée, Alice et Bob doivent trouver un moyen de partager cette clé sans qu'un tiers ne puisse la découvrir. Si la clé est transmise sur un canal non sécurisé, un attaquant peut l'intercepter et accéder à toutes les communications chiffrées.

Ce problème, appelé *problème de distribution des clés*, constitue l'une des principales limites de la cryptographie symétrique. Plus le nombre d'utilisateurs augmente, plus la gestion et l'échange sécurisés des clés deviennent complexes.

Une analogie courante consiste à imaginer qu'Alice souhaite envoyer un coffre verrouillé à Bob. Pour que Bob puisse l'ouvrir, il doit posséder la clé du cadenas. Mais transmettre cette clé de manière sûre pose exactement le même problème que celui que l'on cherche à résoudre. Il faut donc trouver un moyen d'échanger un secret sans disposer au préalable d'un canal sécurisé.

Cette difficulté a conduit au développement de la cryptographie asymétrique, dans laquelle les utilisateurs n'ont plus besoin de partager une clé secrète avant de communiquer.

2.3 Le paradigme asymétrique

La cryptographie à **clé publique** (ou *asymétrique*), introduite par Diffie et Hellman en 1976 [4], résout ce problème élégamment : chaque utilisateur possède une **paire de clés** :

— une **clé publique** pk, diffusée librement ;

— une **clé privée** sk, gardée secrète.

Un message chiffré avec pk ne peut être déchiffré qu'avec sk correspondant. Le principe général est le suivant : lorsqu'une personne souhaite envoyer un message confidentiel à un destinataire, elle chiffre ce message à l'aide de la clé publique de celui-ci. Une fois chiffré, le message ne peut être déchiffré qu'à l'aide de la clé privée correspondante.

Cette séparation des rôles présente un avantage majeur par rapport au chiffrement symétrique. En effet, il n'est plus nécessaire d'échanger secrètement une clé avant toute communication. La clé publique peut être distribuée librement sans compromettre la sécurité du système.

Le cryptosystème RSA, proposé en 1978 par Rivest, Shamir et Adleman [2], est l'un des exemples les plus connus de chiffrement asymétrique. Il repose sur des propriétés mathématiques liées à la théorie des nombres et à la difficulté de certains problèmes de calcul.

2.4 Fonctions à sens unique et à trappe

Définition 2.1 (Fonction à sens unique)**.** *Une fonction $f : X \to Y$ est dite* à sens unique *si elle est facile à calculer mais difficile à inverser sans information supplémentaire [3].*

Définition 2.2 (Trappe)**.** *Une* trappe *est une information secrète t qui permet d'inverser efficacement f. On dit alors que f est une* fonction à trappe*.*

La sécurité des systèmes de chiffrement asymétriques repose sur l'existence de fonctions dites **à sens unique**. Une fonction à sens unique est une fonction facile à calculer dans un sens, mais dont l'inversion est considérée comme extrêmement difficile sans information supplémentaire[3].

Un exemple simple est la multiplication de deux grands nombres premiers. Calculer leur produit est relativement rapide, tandis que retrouver les facteurs premiers à partir du produit obtenu constitue un problème beaucoup plus complexe.

Une **fonction à trappe** est une fonction à sens unique possédant une information secrète, appelée *trappe*, qui permet d'inverser efficacement l'opération. Sans cette information, le calcul inverse reste considéré comme difficile.

Le cryptosystème RSA repose précisément sur ce principe. La clé publique permet d'effectuer une opération mathématique simple pour chiffrer un message. En revanche, retrouver le message original à partir du texte chiffré est supposé difficile sans connaître la clé privée.

Dans RSA, la trappe est liée à la connaissance de la factorisation d'un grand entier construit à partir de deux nombres premiers. Cette information permet de calculer la clé privée et donc d'inverser efficacement l'opération de chiffrement [2].

2.5 Chiffrement asymétrique

Le chiffrement asymétrique repose sur l'utilisation d'une paire de clés : une clé publique, qui peut être diffusée librement, et une clé privée, qui doit rester secrète. In-

troduit par Diffie et Hellman en 1976 [4], ce paradigme permet de résoudre le problème de la distribution des clés rencontré dans les systèmes symétriques.

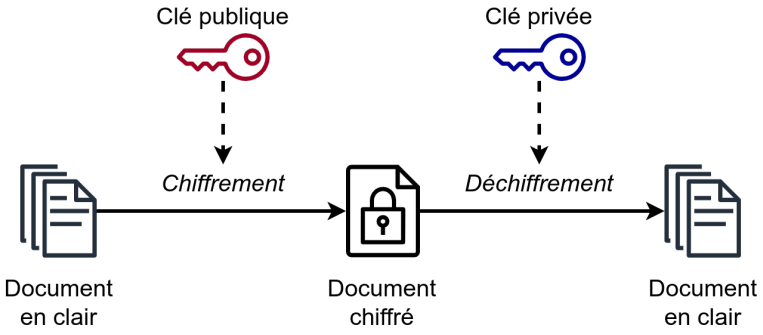

FIGURE 2 – Principe général du chiffrement symétrique

L'un des premiers protocoles à clé publique est l'échange de clés de Diffie-Hellman. Son objectif est de permettre à deux utilisateurs de construire une clé secrète commune à travers un canal public. La sécurité du protocole repose sur la difficulté du problème du logarithme discret.

Le cryptosystème RSA, proposé en 1978 par Rivest, Shamir et Adleman [2], est l'un des systèmes asymétriques les plus célèbres et constitue le sujet principal de ce rapport. Contrairement à Diffie-Hellman, RSA permet directement le chiffrement de messages ainsi que la création de signatures numériques. Sa sécurité repose sur la difficulté de factoriser un grand entier obtenu à partir du produit de deux nombres premiers.

Par exemple, si deux nombres premiers (p) et (q) sont choisis, leur produit

$$n = p \times q$$

est facile à calculer. En revanche, retrouver (p) et (q) à partir de (n) devient extrêmement difficile lorsque ces nombres comportent plusieurs centaines de chiffres. Cette propriété constitue le fondement de la sécurité de RSA.

La sécurité du protocole de Diffie–Hellman repose sur la difficulté du problème du logarithme discret. Soit $n \geq 2$ un entier et soit $\mathbb{Z}_n^*$ le groupe multiplicatif des entiers inversibles modulo n. Étant donnés deux éléments $g, h \in \mathbb{Z}_n^*$ tels que

$$h \equiv g^x \pmod{n},$$

il est facile de calculer h lorsque g, x et n sont connus. En revanche, le problème inverse consiste à déterminer l'exposant x à partir de g et h. Cet exposant est appelé le logarithme discret de h en base g.

À l'heure actuelle, aucun algorithme efficace n'est connu pour résoudre ce problème dans le cas général lorsque les paramètres sont suffisamment grands. Cette difficulté

constitue le fondement de la sécurité de plusieurs systèmes cryptographiques à clé publique, notamment le protocole de Diffie–Hellman.

2.6 Fonctions de hachage

Une fonction de hachage est un algorithme qui transforme un message de taille quelconque en une empreinte numérique de taille fixe appelée *haché* (*hash*) ou *digest*. Cette empreinte constitue une représentation condensée du message et permet notamment de vérifier l'intégrité des données.

Contrairement aux algorithmes de chiffrement, les fonctions de hachage ne sont pas conçues pour être inversées. À partir d'une empreinte, il doit être pratiquement impossible de retrouver le message d'origine. Une fonction de hachage doit également posséder plusieurs propriétés importantes :

— elle doit être rapide à calculer ;
— il doit être très difficile de reconstruire un message à partir de son empreinte ;
— il doit être extrêmement difficile de trouver deux messages produisant la même empreinte (*collision*) ;
— une faible modification du message doit entraîner une forte modification de l'empreinte.

Parmi les fonctions de hachage les plus utilisées figurent celles de la famille SHA (*Secure Hash Algorithm*), notamment SHA-256. Ces fonctions sont largement employées dans les protocoles de sécurité modernes, les certificats numériques, les signatures électroniques et la vérification de l'intégrité des fichiers.

Afin d'illustrer le principe d'une fonction de hachage, considérons un exemple simplifié. Supposons qu'une fonction de hachage additionne les codes ASCII des caractères d'un message puis conserve uniquement le reste de la division par 100.

Prenons le message :

[`RSA`]

Les codes ASCII correspondants sont :

[R = 82, S = 83, A = 65.]

La somme vaut :

[82 + 83 + 65 = 230.]

Le haché obtenu est alors :

$$h = 230 \bmod 100 = 30.$$

L'empreinte du message est donc :

$$h = 30.$$

Si l'on modifie légèrement le message en remplaçant `RSA` par `RSB`, on obtient :

$$82 + 83 + 66 = 231,$$

et donc :

$$h = 231 \bmod 100 = 31.$$

L'empreinte change alors que le message n'a été modifié que d'un seul caractère. Cet exemple est volontairement simplifié et n'offre aucune sécurité réelle, mais il permet de comprendre le principe général du hachage.

Par le passé, d'autres fonctions de hachage ont été largement utilisées, notamment MD5 (*Message Digest 5*), proposé par Ronald Rivest en 1992. Cependant, des faiblesses ont été découvertes dans cet algorithme, permettant de construire des collisions en un temps raisonnable. Pour cette raison, MD5 n'est aujourd'hui plus considéré comme sûr pour les applications cryptographiques et a été progressivement remplacé par des fonctions plus robustes telles que SHA-256.

Les fonctions de hachage jouent un rôle essentiel dans de nombreux mécanismes de sécurité. Dans le cas de RSA, elles sont notamment utilisées pour les signatures numériques. Plutôt que de signer directement un document complet, ce qui serait coûteux en calcul, on signe son empreinte. Cette approche améliore l'efficacité tout en garantissant l'intégrité des données.

2.7 Signatures numériques

Les signatures numériques permettent de garantir l'authenticité d'un document ainsi que son intégrité. Elles constituent l'un des usages les plus importants du cryptosystème RSA.

Le principe est différent du chiffrement classique. Lorsqu'un utilisateur souhaite envoyer un document signé, il ne chiffre pas le document avec sa clé privée. Il calcule d'abord son empreinte à l'aide d'une fonction de hachage telle que SHA-256. Cette empreinte est ensuite signée grâce à la clé privée RSA.

Le destinataire effectue alors deux vérifications :

1. il calcule lui-même l'empreinte du document reçu ;
2. il utilise la clé publique du signataire pour déchiffrer la signature.

Si les deux empreintes correspondent, le document est considéré comme authentique et n'a pas été modifié durant la transmission.

Dans les applications réelles, les valeurs utilisées sont beaucoup plus grandes et les empreintes proviennent généralement d'une fonction telle que SHA-256. L'association

des fonctions de hachage et de RSA permet ainsi de garantir l'identité de l'expéditeur et l'intégrité des données transmises [2].

2.8 Comparaison des algorithmes

Le tableau suivant résume les caractéristiques principales des algorithmes présentés dans ce chapitre :

Algorithme	**Taille de clé**	**Bloc**	**Rondes**	**Type**
AES	128 / 192 / 256 bits	128 bits	10 / 12 / 14	Chiffrement symétrique
DES	56 bits (effective)	64 bits	16	Symétrique (obsolète)
RSA	1024 / 2048 / 3072 / 4096 bits	n	—	Cryptosystème à clé publi

TABLE 1 – Comparaison des principaux algorithmes cryptographiques

Contrairement aux algorithmes de chiffrement symétrique par blocs, RSA ne comporte pas de rondes de chiffrement. Il repose sur des opérations d'exponentiation modulaire, qui constituent le cœur de son mécanisme de chiffrement et de déchiffrement.

3 Fondements mathématiques de RSA

3.1 Arithmétique modulaire

L'arithmétique modulaire est l'un des fondements mathématiques du cryptosystème RSA. Elle consiste à effectuer des calculs sur les restes d'une division euclidienne.

Soit n un entier strictement positif. Deux entiers a et b sont dits congrus modulo n si leur différence est divisible par n. On note :

$$a \equiv b \mod n.$$

définition ,

$$n \mid a - b.$$

Par exemple,

$$17 \equiv 5 \mod 12,$$

car

$$17 - 5 = 12,$$

et 12 est divisible par 12.

L'ensemble des nombres congrus entre eux forme une classe de congruence. Dans le cas du modulo 12, les nombres

[. . .,-7,5,17,29,41,. . .]

appartiennent à la même classe de congruence.

Les opérations usuelles conservent la congruence. Si

$$a \equiv b \mod n$$

et

$$c \equiv d \mod n,$$

alors :

$$a + c \equiv b + d \mod n,$$

$$a - c \equiv b - d \mod n,$$

$$ac \equiv bd \mod n.$$

Par exemple :

$$8 + 11 = 19.$$

En travaillant modulo 12 :

$$19 \equiv 7 \mod 12.$$

On écrit alors :

$$8 + 11 \equiv 7 \mod 12.$$

L'arithmétique modulaire permet également d'effectuer des calculs de puissances très importantes tout en manipulant des nombres relativement petits. Cette propriété est essentielle dans RSA, où les opérations de chiffrement et de déchiffrement reposent sur des exponentiations modulaires.

3.2 PGCD et algorithme d'Euclide

Le plus grand commun diviseur de deux entiers a et b, noté pgcd(a,b), est le plus grand entier qui divise simultanément a et b.

Par exemple :

$$pgcd(30, 18) = 6.$$

L'algorithme d'Euclide permet de calculer efficacement le plus grand commun diviseur (PGCD) de deux entiers a et b.

Il repose sur la propriété fondamentale suivante :

$$pgcd(a, b) = pgcd(b, a \bmod b),$$

où $a \bmod b$ désigne le reste de la division euclidienne de a par b.

À chaque étape, on remplace donc le couple (a, b) par (b, r), où r est le reste de la division de a par b. Cette transformation conserve le PGCD tout en réduisant strictement les valeurs considérées.

Ainsi, on obtient une suite décroissante de restes :

$$b > r_1 > r_2 > \cdots \geq 0,$$

ce qui garantit que l'algorithme se termine en un nombre fini d'étapes.

Lorsque le reste devient nul, le dernier reste non nul est le PGCD des deux nombres. Considérons l'exemple :

$$pgcd(252, 105).$$

On effectue les divisions euclidiennes successives :

$$252 = 2 \times 105 + 42,$$

$$105 = 2 \times 42 + 21,$$

$$42 = 2 \times 21 + 0.$$

Le dernier reste non nul est 21. On obtient donc :

$$pgcd(252, 105) = 21.$$

3.3 Algorithme d'Euclide étendu

L'algorithme d'Euclide étendu est une extension de l'algorithme d'Euclide classique. En plus de calculer le PGCD de deux entiers, il permet de déterminer les coefficients de Bézout.

Le théorème de Bézout affirme que pour tous entiers a et b, il existe des entiers u et v tels que :

$$au + bv = pgcd(a, b).$$

Considérons les entiers :

$$a = 26, \qquad b = 7.$$

L'algorithme d'Euclide donne :

$$26 = 3 \times 7 + 5,$$

$$7 = 1 \times 5 + 2,$$

$$5 = 2 \times 2 + 1,$$

$$2 = 2 \times 1 + 0.$$

On obtient donc :

$$pgcd(26, 7) = 1.$$

Pour déterminer les coefficients de Bézout, on remonte les calculs :

$$1 = 5 - 2 \times 2.$$

Or,

$$2 = 7 - 5.$$

Ainsi,

$$1 = 5 - 2(7 - 5) = 3 \times 5 - 2 \times 7.$$

Comme

$$5 = 26 - 3 \times 7,$$

on obtient :

$$1 = 3(26 - 3 \times 7) - 2 \times 7 = 3 \times 26 - 11 \times 7.$$

Les coefficients de Bézout sont donc :

$$u = 3, \qquad v = -11.$$

L'une des principales applications de l'algorithme d'Euclide étendu est le calcul des inverses modulaires.

On cherche un entier d tel que :

$$ed \equiv 1 \pmod{n}.$$

Dans l'exemple précédent :

$$7 \times (-11) + 26 \times 3 = 1.$$

Ainsi,

$$7 \times (-11) \equiv 1 \pmod{26}.$$

Comme

$$-11 \equiv 15 \pmod{26},$$

on obtient :

$$7^{-1} \equiv 15 \pmod{26}.$$

Une vérification donne :

$$7 \times 15 = 105 \equiv 1 \pmod{26}.$$

Dans RSA, cette méthode est utilisée pour calculer l'exposant privé d à partir de l'exposant public e et de la valeur $\varphi(n)$. En effet, la clé privée doit vérifier :

$$ed \equiv 1 \pmod{\varphi(n)}.$$

L'algorithme d'Euclide étendu constitue donc une étape essentielle de la génération des clés RSA.

Complexité

L'algorithme d'Euclide étendu effectue les mêmes divisions successives que l'algorithme d'Euclide classique tout en calculant simultanément les coefficients de Bézout. Son coût est donc du même ordre de grandeur.

Soit k le nombre de bits nécessaires pour représenter les entiers manipulés. À chaque étape, la taille des restes diminue rapidement et le nombre total d'itérations est de l'ordre de O(k).

Dans une implémentation classique, une division euclidienne entre deux entiers de k bits nécessite $O(k^2)$ opérations élémentaires. Comme l'algorithme réalise $O(k)$ divisions, sa complexité temporelle est :

$$O(k^3).$$

Cette complexité reste très raisonnable en pratique, même pour les grands entiers utilisés en cryptographie. L'algorithme d'Euclide étendu est ainsi largement utilisé pour calculer les inverses modulaires nécessaires à la génération des clés RSA.

Par ailleurs, son coût est généralement négligeable devant celui des exponentiations modulaires, qui constituent les opérations les plus coûteuses lors du chiffrement, du déchiffrement et de la signature RSA.

3.4 Fonction indicatrice d'Euler

La fonction indicatrice d'Euler, notée $\varphi(n)$, désigne le nombre d'entiers positifs non nuls inférieurs ou égaux à n qui sont premiers avec n.

Autrement dit :

$$\varphi(n) = |\{k \in \mathbb{N} \mid 1 \leq k \leq n \text{ et } pgcd(k, n) = 1\}|.$$

Cette fonction joue un rôle fondamental dans le cryptosystème RSA, car elle intervient directement dans la génération des clés.

Cas d'un nombre premier

Si p est un nombre premier, tous les entiers compris entre 1 et $p - 1$ sont premiers avec p. On obtient donc :

$$\varphi(p) = p - 1.$$

Par exemple :

$$\varphi(11) = 10.$$

Cas de deux nombres premiers

Considérons deux nombres premiers distincts p et q.

On pose :

$$n = pq.$$

Parmi les entiers de 1 à pq, les nombres qui ne sont pas premiers avec pq sont exactement les multiples de p ou de q.

Il y a q multiples de p :

$$p, 2p, \ldots, qp.$$

Il y a p multiples de q :

$$q, 2q, \ldots, pq.$$

Le nombre pq apparaît dans les deux listes ; il est donc compté deux fois. Le nombre total d'entiers non premiers avec pq est :

$$q + p - 1.$$

Par conséquent :

$$\varphi(pq) = pq - (p + q - 1) = pq - p - q + 1 = (p - 1)(q - 1).$$

$$\boxed{\varphi(pq) = (p - 1)(q - 1)}.$$

Par exemple, si

$$p = 5, \qquad q = 11,$$

alors :

$$\varphi(55) = (5 - 1)(11 - 1) = 4 \times 10 = 40.$$

Cette formule est essentielle dans RSA. En effet, lorsque le module RSA est défini par

$$n = pq,$$

la valeur

$$\varphi(n) = (p - 1)(q - 1)$$

est utilisée pour calculer les exposants public et privé.

3.5 Théorème d'Euler

Le théorème d'Euler généralise le petit théorème de Fermat et constitue le fondement mathématique du cryptosystème RSA.

Énoncé

Soit n un entier strictement positif et soit a un entier premier avec n.
Alors :

$$a^{\varphi(n)} \equiv 1 \pmod{n}.$$

Autrement dit, lorsque a et n sont premiers entre eux, élever a à la puissance $\varphi(n)$ donne toujours un reste égal à 1 modulo n.

Exemple

Prenons :

$$n = 10 = 2 \times 5.$$

Les entiers premiers avec 10 sont :

$$1, 3, 7, 9.$$

On a donc :

$$\varphi(10) = (5-1)(2-1) = 4.$$

Choisissons :

$$a = 3.$$

Le théorème d'Euler affirme que :

$$3^4 \equiv 1 \pmod{10}.$$

En effet :

$$3^4 = 81,$$

et

$$81 \equiv 1 \pmod{10}.$$

Interprétation

Le théorème d'Euler montre qu'une puissance particulière d'un entier premier avec n ramène toujours au reste 1 modulo n.

Cette propriété permet de construire des opérations réversibles : une exponentiation peut être annulée par une autre exponentiation convenablement choisie.

3.6 Théorème des restes chinois

Le théorème des restes chinois (TRC) est un résultat fondamental de l'arithmétique modulaire. Il permet de résoudre un système de congruences modulo plusieurs entiers premiers entre eux.

Énoncé

Soient n_1 et n_2 deux entiers premiers entre eux.

Pour tout système :

$$\begin{cases} x \equiv a \mod n_1, \\ x \equiv b \mod n_2. \end{cases}$$

il existe une solution unique modulo $n_1 n_2$.

Autrement dit, connaître les restes d'un entier modulo n_1 et modulo n_2 permet de reconstruire cet entier modulo $n_1 n_2$.

Exemple

Considérons le système :

$$\begin{cases} x \equiv 2 \mod 3, \\ x \equiv 3 \mod 5. \end{cases}$$

Les nombres congrus à 2 modulo 3 sont :

$$2, 5, 8, 11, 14, \ldots$$

Parmi eux, le premier qui est congru à 3 modulo 5 est :

$$8.$$

On obtient donc :

$$x \equiv 8 \pmod{15}.$$

3.7 Exponentiation modulaire rapide

Les opérations de chiffrement, de déchiffrement et de signature RSA reposent sur le calcul de puissances modulaires de la forme :

$$a^e \bmod n.$$

Lorsque l'exposant e contient plusieurs centaines ou milliers de bits, un calcul naïf serait extrêmement coûteux.

Pour effectuer efficacement ces calculs, on utilise l'algorithme d'exponentiation modulaire rapide, également appelé *square-and-multiply*.

Principe

L'idée consiste à exploiter l'écriture binaire de l'exposant.

Par exemple :

$$13 = 1101_2.$$

On calcule successivement les puissances :

$$a^1, \quad a^2, \quad a^4, \quad a^8,$$

par des opérations de mise au carré, puis on multiplie uniquement les puissances correspondant aux bits égaux à 1.

Exemple

Calculons :

$$3^{13} \bmod 17.$$

Comme :

$$13 = 1101_2,$$

on obtient :

$$3^1 \equiv 3 \pmod{17},$$

$$3^2 \equiv 9 \pmod{17},$$

$$3^4 \equiv 9^2 = 81 \equiv 13 \pmod{17},$$

$$3^8 \equiv 13^2 = 169 \equiv 16 \pmod{17}.$$

Les bits à 1 correspondent à :

$$13 = 8 + 4 + 1$$

Ainsi :

$$3^{13} = 3^8 \times 3^4 \times 3^1.$$

Modulo 17 :

$$3^{13} \equiv 16 \times 13 \times 3 \pmod{17}.$$

$$16 \times 13 = 208 \equiv 4 \pmod{17},$$

$$4 \times 3 = 12.$$

Finalement :

$$3^{13} \equiv 12 \pmod{17}.$$

Complexité

Soit e un exposant codé sur k bits.

Une méthode naïve nécessiterait environ :

$$O(e)$$

multiplications.

L'algorithme square-and-multiply n'effectue qu'un nombre de multiplications proportionnel au nombre de bits de l'exposant :

$$O(k).$$

Comme

$$k \approx \log_2(e),$$

la complexité devient :

$$O(\log e).$$

Du point de vue informatique, si les entiers manipulés possèdent n bits et que chaque multiplication coûte $O(n^2)$, la complexité globale est :

$$O(n^2 \log e).$$

4 L'algorithme RSA

4.1 Historique de RSA

Le développement de la cryptographie à clé publique commence avec les travaux de Diffie et Hellman en 1976. Dans leur article fondateur [4], ils introduisent pour la première fois l'idée qu'il est possible de communiquer de manière sécurisée sans partager préalablement une clé secrète. Leur contribution majeure est conceptuelle : ils proposent un nouveau paradigme de cryptographie, mais ne fournissent pas encore de système concret de chiffrement à clé publique.

Deux ans plus tard, en 1978, Rivest, Shamir et Adleman proposent le premier système de chiffrement à clé publique réalisable : le cryptosystème RSA [2]. Leur idée repose sur des outils de théorie des nombres et sur la difficulté supposée de factoriser un grand entier produit de deux nombres premiers.

Contrairement aux propositions antérieures, RSA fournit un algorithme complet permettant à la fois le chiffrement, le déchiffrement et la génération de signatures numériques. Ce système marque ainsi la première implémentation concrète du paradigme introduit par Diffie et Hellman, et constitue encore aujourd'hui l'un des fondements de la cryptographie moderne.

4.2 Génération des clés

La génération des clés RSA repose sur des opérations simples en théorie, mais difficiles à inverser en pratique.

On commence par choisir deux grands nombres premiers distincts p et q. Ces nombres doivent être choisis de manière aléatoire et suffisamment grands afin de garantir la sécurité du système.

On définit ensuite :

$$n = pq.$$

Le nombre n est appelé module RSA et constitue une partie commune aux deux clés.

On calcule ensuite la fonction indicatrice d'Euler :

$$\varphi(n) = (p-1)(q-1),$$

résultat fondamental issu de la théorie des nombres.

On choisit alors un entier e tel que :

$$1 < e < \varphi(n) \quad \text{et} \quad pgcd(e, \varphi(n)) = 1.$$

Cet entier e est appelé exposant public.

Enfin, on détermine l'entier d tel que :

$$ed \equiv 1 \pmod{\varphi(n)}.$$

Autrement dit, d est l'inverse modulaire de e modulo $\varphi(n)$, calculé à l'aide de l'algorithme d'Euclide étendu.

On obtient alors :

— la **clé publique** : (n, e),

— la **clé privée** : (n, d).

La clé publique peut être diffusée librement, tandis que la clé privée doit rester strictement secrète.

Théorème d'Euler

Dans RSA, on choisit deux exposants e et d tels que :

$$ed \equiv 1 \pmod{\varphi(n)}.$$

Il existe alors un entier k vérifiant :

$$ed = 1 + k\varphi(n).$$

Pour un message M premier avec n, on obtient :

$$M^{ed} = M^{1+k\varphi(n)} = M(M^{\varphi(n)})^k.$$

D'après le théorème d'Euler :

$$M^{\varphi(n)} \equiv 1 \pmod{n}.$$

Ainsi :

$$M^{ed} \equiv M \pmod{n}.$$

Cette propriété garantit que le déchiffrement retrouve le message initial après le chiffrement. Le théorème d'Euler constitue donc la base théorique de la correction du cryptosystème RSA.

4.3 Chiffrement

Le chiffrement RSA consiste à transformer un message M, représenté comme un entier de $\mathbb{Z}_n$, en un message chiffré C à l'aide de la clé publique.

Le chiffrement est défini par :

$$C = M^e \bmod n.$$

La clé publique (n, e) est utilisée pour chiffrer les messages, ce qui signifie que toute personne peut envoyer un message confidentiel au détenteur de la clé privée.

Cependant, seule la personne possédant la clé privée d est capable de retrouver le message original. Cette séparation garantit la confidentialité des communications.

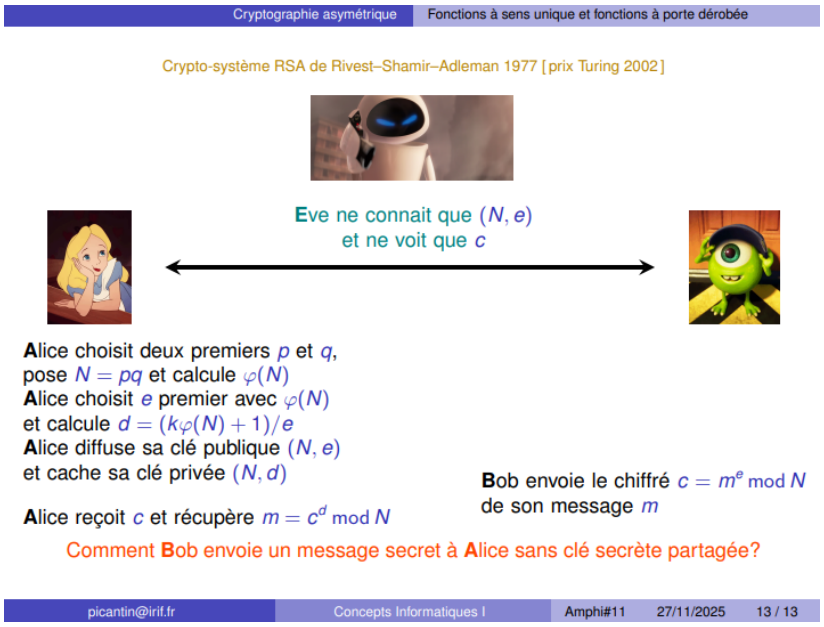

FIGURE 3 – Principe général de RSA

4.4 Déchiffrement

Le déchiffrement RSA est l'opération inverse du chiffrement. Il permet de retrouver le message original M à partir du message chiffré C en utilisant la clé privée.

Le déchiffrement est défini par :

$$M = C^d \bmod n.$$

Dans cette formule, l'exposant d correspond à la clé privée du système. Il est calculé de manière à être l'inverse de l'exposant public e modulo $\varphi(n)$, ce qui garantit la cohérence mathématique du procédé.

La clé privée (n, d) est détenue uniquement par le destinataire du message. Elle permet de réaliser l'opération inverse du chiffrement et de retrouver efficacement le message original. Sans cette clé, le calcul inverse de l'exponentiation modulaire est considéré comme infaisable en pratique lorsque n est suffisamment grand.

Ainsi, la sécurité de RSA repose sur le fait que, bien que le chiffrement soit public, le déchiffrement reste inaccessible sans la connaissance de la clé privée.

Importance pratique : exponentiation modulaire rapide

Sans l'exponentiation modulaire rapide, RSA serait impraticable pour les tailles de clés utilisées aujourd'hui (2048 bits ou davantage).

Cet algorithme constitue l'une des optimisations fondamentales de toutes les implémentations modernes de RSA, aussi bien pour le chiffrement que pour le déchiffrement et les signatures numériques.

4.5 Preuve de correction

Nous montrons dans cette section que le déchiffrement RSA permet de retrouver le message initial après le chiffrement.

Soit $M \in \mathbb{Z}_n$, où $n = pq$ avec p et q deux nombres premiers distincts. Les exposants public et privé e et d sont choisis de telle sorte que

$$ed \equiv 1 \pmod{\varphi(n)},$$

où

$$\varphi(n) = (p-1)(q-1).$$

Il existe donc un entier k tel que

$$ed = 1 + k\varphi(n).$$

Lors du chiffrement, on calcule

$$C = M^e \bmod n.$$

Le déchiffrement consiste à calculer

$$C^d \bmod n.$$

En remplaçant C par sa définition, on obtient

$$C^d = (M^e)^d = M^{ed}.$$

Il suffit donc de montrer que

$$M^{ed} \equiv M \pmod{n}.$$

Si : $pgcd(M, n) = 1$

Si M est premier avec n, le théorème d'Euler donne

$$M^{\varphi(n)} \equiv 1 \pmod{n}.$$

Comme

$$ed = 1 + k\varphi(n),$$

on obtient

$$M^{ed} = M^{1+k\varphi(n)} = M\left(M^{\varphi(n)}\right)^k.$$

Par conséquent,

$$M^{ed} \equiv M \cdot 1^k \equiv M \pmod{n}.$$

Le déchiffrement restitue donc correctement le message.

Cas général

Supposons maintenant que M ne soit pas premier avec n. Comme $n = pq$, deux situations peuvent se présenter.

Si : $M \equiv 0 \pmod{n}$**.** Dans ce cas,

$$C = M^e \equiv 0 \pmod{n}.$$

Par conséquent,

$$C^d \equiv 0 \equiv M \pmod{n}.$$

Le déchiffrement restitue donc correctement le message.

Si : $M \not\equiv 0 \pmod{n}$**.** Puisque M n'est pas premier avec n, il est divisible par exactement l'un des deux facteurs premiers p ou q.

Supposons par exemple que

$$M \equiv 0 \pmod{p}.$$

Alors

$$C = M^e \equiv 0^e \equiv 0 \pmod{p},$$

et donc

$$C^d \equiv 0^d \equiv 0 \equiv M \pmod{p}.$$

En revanche, modulo q, on a

$$pgcd(M, q) = 1,$$

puisque q ne divise pas M.On a alors :

$$M^{q-1} \equiv 1 \pmod{q}.$$

Comme

$$ed = 1 + k(p-1)(q-1),$$

on obtient

$$M^{ed} = M\left(M^{q-1}\right)^{k(p-1)} \equiv M \pmod{q}.$$

Ainsi,

$$C^d \equiv M \pmod{p}$$

et

$$C^d \equiv M \pmod{q}.$$

Autrement dit, les entiers C^d et M ont le même reste dans la division par p ainsi que par q. Comme p et q sont deux nombres premiers distincts, ils sont premiers entre eux.

Le théorème des restes chinois affirme alors qu'il existe un unique entier modulo pq satisfaisant simultanément ces deux congruences. Par conséquent,

$$C^d \equiv M \pmod{pq}.$$

Or, par définition de RSA,

$$n = pq.$$

On obtient donc

$$C^d \equiv M \pmod{n}.$$

Ainsi, le message obtenu après le déchiffrement est identique au message initial, ce qui établit la correction du cryptosystème RSA pour tout message $M \in \mathbb{Z}_n$.

$$C^d \equiv M \pmod{n}.$$

Le cas où $M \equiv 0 \pmod{q}$ se démontre de manière entièrement analogue.
Dans tous les cas,

$$M^{ed} \equiv M \pmod{p}.$$

Modulo q. Le raisonnement est identique.

— Si $M \equiv 0 \pmod{q}$, alors

$$M^{ed} \equiv 0 \equiv M \pmod{q}.$$

— Si $M \not\equiv 0 \pmod{q}$, alors $pgcd(M, q) = 1$ et d'après le résultat précédent

$$M^{q-1} \equiv 1 \pmod{q}.$$

On obtient alors

$$M^{ed} = M\left(M^{q-1}\right)^{k(p-1)} \equiv M \pmod{q}.$$

Ainsi,

$$M^{ed} \equiv M \pmod{q}.$$

Les congruences

$$M^{ed} \equiv M \pmod{p}$$

et

$$M^{ed} \equiv M \pmod{q}$$

étant simultanément vérifiées, le théorème des restes chinois permet de conclure que

$$M^{ed} \equiv M \pmod{pq}.$$

Comme $n = pq$, on obtient finalement

$$C^d = M^{ed} \equiv M \pmod{n}.$$

Le message obtenu après le déchiffrement est donc toujours identique au message initial. Cette propriété garantit la correction du cryptosystème RSA pour tout message appartenant à $\mathbb{Z}_n$.

4.6 Exemple numérique complet

Nous illustrons maintenant le fonctionnement du cryptosystème RSA à l'aide d'un exemple simple utilisant de petits nombres premiers. Bien que ces valeurs soient beaucoup trop faibles pour garantir une sécurité réelle, elles permettent de comprendre les différentes étapes de l'algorithme.

Génération des clés

Choisissons les deux nombres premiers :

$$p = 11, \qquad q = 13.$$

On calcule alors le module RSA :

$$n = pq = 11 \times 13 = 143.$$

La fonction indicatrice d'Euler vaut :

$$\varphi(n) = (p-1)(q-1) = 10 \times 12 = 120.$$

Choisissons comme exposant public :

$$e = 7.$$

On vérifie que :

$$pgcd(7, 120) = 1.$$

Il faut ensuite déterminer l'exposant privé d tel que

$$ed \equiv 1 \pmod{120}.$$

On cherche donc l'inverse modulaire de 7 modulo 120.
Comme

$$7 \times 103 = 721 = 6 \times 120 + 1,$$

on obtient :

$$d = 103.$$

Les clés sont donc :
— Clé publique : $(n, e) = (143, 7)$;
— Clé privée : $(n, d) = (143, 103)$.

Chiffrement

Supposons que le message à transmettre soit

$$M = 9.$$

Le chiffrement RSA est défini par :

$$C = M^e \bmod n.$$

On calcule donc :

$$C = 9^7 \bmod 143.$$

Utilisons l'exponentiation modulaire rapide.

$$9^2 = 81,$$

$$9^4 = 81^2 = 6561 \equiv 126 \pmod{143},$$

$$9^7 = 9^4 \times 9^2 \times 9.$$

Ainsi :

$$9^7 \equiv 126 \times 81 \times 9 \pmod{143}.$$

Or,

$$126 \times 81 = 10206 \equiv 53 \pmod{143},$$

et

$$53 \times 9 = 477 \equiv 48 \pmod{143}.$$

On obtient donc :

$$C = 48.$$

Le message chiffré transmis est donc :

$$\boxed{C = 48}.$$

Déchiffrement

Le destinataire utilise sa clé privée pour calculer :

$$M = C^d \bmod n.$$

Dans notre exemple :

$$M = 48^{103} \bmod 143.$$

Le calcul complet est long, mais l'exponentiation modulaire rapide permet de l'effectuer efficacement. On obtient :

$$48^{103} \equiv 9 \pmod{143}.$$

Ainsi :

$$M = 9.$$

Vérification

Le message obtenu après déchiffrement est identique au message initial :

$$9 \longrightarrow 48 \longrightarrow 9.$$

Le chiffrement puis le déchiffrement ont donc permis de retrouver exactement le message d'origine.

Cet exemple illustre concrètement le fonctionnement du cryptosystème RSA et confirme la propriété démontrée dans la section précédente :

$$M^{ed} \equiv M \pmod{n}.$$

4.7 Signature RSA

Outre le chiffrement, le cryptosystème RSA permet également de réaliser des signatures numériques. Celles-ci jouent un rôle essentiel dans la sécurité des communications modernes en garantissant l'authenticité et l'intégrité des documents échangés.

Le principe est différent de celui du chiffrement. Au lieu d'utiliser la clé publique pour chiffrer un message, l'expéditeur utilise sa clé privée pour signer une empreinte du document.

Soit (M) un message et (H(M)) son empreinte obtenue à l'aide d'une fonction de hachage cryptographique telle que SHA-256.

La signature RSA est définie par :

$$S = H(M)^d \bmod n,$$

où (d) est l'exposant privé du signataire.

Le couple ((M,S)) est ensuite transmis au destinataire.

Pour vérifier la signature, le destinataire calcule l'empreinte du message reçu :

$$H(M),$$

puis utilise la clé publique du signataire afin de calculer :

$$S^e \bmod n,$$

où (e) est l'exposant public.

La signature est considérée comme valide si :

$$S^e \equiv H(M) \pmod{n}.$$

Cette propriété repose sur le fait que les exposants (e) et (d) vérifient :

$$ed \equiv 1 \pmod{\varphi(n)}.$$

On obtient alors :

$$(H(M)^d)^e = H(M)^{de} \equiv H(M) \pmod{n}.$$

Ainsi, seule la personne possédant la clé privée peut produire une signature qui sera correctement vérifiée à l'aide de la clé publique correspondante.

Exemple

Considérons les paramètres RSA suivants :

$$n = 55, \qquad e = 3, \qquad d = 27.$$

Supposons que l'empreinte du document soit :

$$H(M) = 12.$$

L'expéditeur calcule alors la signature :

$$S = 12^{27} \bmod 55 = 23.$$

Le destinataire reçoit le document ainsi que la signature (S=23).
Pour vérifier la signature, il calcule :

$$S^e \bmod n = 23^3 \bmod 55.$$

Or :

$$23^2 = 529 \equiv 34 \pmod{55},$$

et

$$23^3 = 23 \times 23^2 = 23 \times 34 = 782 \equiv 12 \pmod{55}.$$

On obtient donc :

$$S^e \equiv 12 \equiv H(M) \pmod{55}.$$

La valeur obtenue correspond à l'empreinte du document. La signature est donc valide.

Propriétés des signatures numériques

Les signatures RSA permettent de garantir plusieurs propriétés fondamentales :

- **Authenticité** : la signature prouve que le document a été signé par le détenteur de la clé privée correspondante ;
- **Intégrité** : toute modification du document entraîne une modification de son empreinte et rend la signature invalide ;
- **Non-répudiation** : le signataire ne peut pas nier avoir signé le document, puisqu'il est le seul à posséder la clé privée utilisée pour produire la signature.

Les signatures numériques constituent aujourd'hui l'une des applications les plus importantes de RSA. Elles sont notamment utilisées dans les certificats numériques, les logiciels signés, les transactions électroniques et les documents administratifs dématérialisés [2].

4.8 Optimisation du déchiffrement : RSA-CRT

Dans RSA, le module est défini par :

$$n = pq,$$

où p et q sont deux nombres premiers.
Sans utiliser le théorème des restes chinois, le déchiffrement consiste à calculer

$$C^d \bmod n.$$

En utilisant l'algorithme *square-and-multiply*, le nombre d'itérations est proportionnel au nombre de bits de l'exposant d. En posant

$$k = \log_2(d),$$

la complexité de l'exponentiation est

$$O(k).$$

Si une multiplication sur des entiers de taille n coûte

$$O(n^2),$$

la complexité du déchiffrement est alors

$$O(kn^2).$$

Avec RSA-CRT, le calcul est remplacé par deux exponentiations :

$$C^d \bmod p \quad \text{et} \quad C^d \bmod q.$$

Comme p et q sont de tailles comparables, on a

$$p \approx q \approx \sqrt{n}.$$

Chaque exponentiation a donc un coût

$$O\left(\frac{k}{2}p^2\right) = O\left(\frac{k}{2}(\sqrt{n})^2\right) = O\left(\frac{k}{2}n\right).$$

De même pour q :

$$O\left(\frac{k}{2}q^2\right) = O\left(\frac{k}{2}(\sqrt{n})^2\right) = O\left(\frac{k}{2}n\right).$$

Ainsi, les deux exponentiations donnent

$$O\left(\frac{k}{2}n\right) + O\left(\frac{k}{2}n\right) = O\left(\left(\frac{k}{2} + \frac{k}{2}\right)n\right) = O(kn).$$

La reconstruction finale par le théorème des restes chinois est négligeable devant les exponentiations.

Ainsi, la complexité passe de

$$O(kn^2) \quad \text{à} \quad O(kn),$$

ce qui explique l'accélération significative apportée par RSA-CRT.

4.9 Distinguabilité du RSA

Une propriété importante d'un système de chiffrement moderne est l'indistinguabilité. Intuitivement, un attaquant ne devrait pas être capable de déterminer lequel de deux messages a été chiffré à partir du texte chiffré observé.

Le RSA de base ne possède pas cette propriété. En effet, le chiffrement RSA est déterministe : pour une clé publique donnée, un même message produit toujours le même texte chiffré.

Considérons la clé publique :

$$(n, e) = (143, 7).$$

Supposons qu'un attaquant sache que le message envoyé est soit

$$M_0 = 9,$$

soit

$$M_1 = 10.$$

L'attaquant peut alors calculer lui-même les deux chiffrements :

$$9^7 \bmod 143 = 48,$$

et

$$10^7 \bmod 143 = 10.$$

Supposons maintenant qu'il intercepte le texte chiffré

$$C = 48.$$

Comme ce résultat correspond au chiffrement de M_0, il peut immédiatement conclure que le message envoyé est

$$M_0 = 9.$$

L'attaquant a donc réussi à distinguer les deux messages possibles sans connaître la clé privée.

Cette propriété montre que RSA utilisé sans mécanisme supplémentaire n'est pas suffisamment sûr pour les applications modernes. Un adversaire capable de tester plusieurs messages candidats peut parfois déterminer lequel a été transmis.

Pour remédier à cette faiblesse, les implémentations actuelles utilisent un schéma de bourrage (*padding*) tel que OAEP. Grâce à l'ajout d'aléa lors du bourrage avant le chiffrement, deux chiffrements successifs d'un même message produisent généralement des textes chiffrés différents. Le système devient ainsi beaucoup plus difficile à analyser et résiste aux attaques fondées sur la reconnaissance de messages.

5 Analyse de sécurité et attaques contre RSA

5.1 Le problème de la factorisation

La première attaque naturelle contre RSA consiste à tenter de factoriser le module public N. Comme l'explique Boneh, la factorisation de N constitue une attaque par force brute contre le cryptosystème RSA [1].

En effet, le module RSA est défini comme le produit de deux grands nombres premiers :

$$N = pq.$$

Si un attaquant parvient à retrouver les facteurs premiers p et q, il peut alors calculer la fonction indicatrice d'Euler :

$$\varphi(N) = (p-1)(q-1).$$

Connaissant $\varphi(N)$ et l'exposant public e, il devient alors possible de déterminer l'exposant privé d en calculant l'inverse modulaire de e modulo $\varphi(N)$:

$$d \equiv e^{-1} \pmod{\varphi(N)}.$$

Ainsi, la factorisation du module RSA permet directement de reconstruire la clé privée. Le lien entre la factorisation et la sécurité de RSA peut être résumé par la chaîne suivante :

$$N \longrightarrow (p, q) \longrightarrow \varphi(N) \longrightarrow d.$$

Boneh souligne que les meilleurs algorithmes de factorisation connus, notamment le *General Number Field Sieve* (GNFS), restent très coûteux pour les modules RSA de taille suffisante. Pour cette raison, la sécurité de RSA repose essentiellement sur la difficulté pratique de factoriser de grands entiers composés de deux nombres premiers secrets [1].

Cependant, une question théorique importante demeure ouverte : casser RSA est-il exactement aussi difficile que factoriser N ? Boneh rappelle qu'il est évident que la factorisation permet de casser RSA, mais que la réciproque n'a jamais été démontrée. Il n'est donc pas établi que toute méthode efficace permettant de casser RSA doive nécessairement factoriser le module N [1].

5.2 Méthodes modernes de factorisation

La sécurité de RSA repose directement sur la difficulté de factoriser le module $N = pq$. Depuis l'invention de RSA, de nombreux algorithmes ont été développés afin de résoudre ce problème plus efficacement.

Les méthodes les plus simples, comme la division d'essai ou la méthode ρ de Pollard, permettent de trouver rapidement de petits facteurs premiers mais deviennent inefficaces lorsque les entiers considérés atteignent plusieurs centaines ou milliers de bits [9].

Dans ce qui suit, nous illustrons **Méthode de ρ Pollard** sur un exemple.

considérons l'entier :

$$N = 8051.$$

On cherche à retrouver ses facteurs premiers à l'aide de la méthode ρ de Pollard. On choisit la fonction d'itération :

$$f(x) = x^2 + 1 \pmod{8051},$$

et la valeur initiale :

$$x_0 = 2.$$

Pour détecter un cycle, on utilise l'algorithme de Floyd, appelé méthode de la *tortue et du lièvre*. À chaque étape, on calcule :

$$x_{n+1} = f(x_n),$$

$$y_{n+1} = f(f(y_n)),$$

puis

$$d = pgcd(|x_n - y_n|, N).$$

Si $d > 1$, alors d est un facteur non trivial de N.

Étape 1

$$x = f(2) = 5,$$

$$y = f(f(2)) = f(5) = 26.$$

On calcule :

$$d = pgcd(|26 - 5|, 8051) = pgcd(21, 8051) = 1.$$

Aucun facteur n'a encore été trouvé.

Étape 2

$$x = f(5) = 26,$$

$$f(26) = 26^2 + 1 = 677,$$

$$f(677) = 677^2 + 1 \equiv 7474 \pmod{8051}.$$

Ainsi,

$$y = 7474.$$

On obtient :

$$d = pgcd(|7474 - 26|, 8051) = pgcd(7448, 8051) = 1.$$

On poursuit les calculs.

Étape 3

$$x = f(26) = 677.$$

Pour le lièvre :

$$f(7474) \equiv 2839 \pmod{8051},$$

$$f(2839) \equiv 871 \pmod{8051}.$$

Donc :

$$y = 871.$$

On calcule alors :

$$d = pgcd(|871 - 677|, 8051) = pgcd(194, 8051) = 97.$$

Comme

$$1 < 97 < 8051,$$

on a trouvé un facteur non trivial de N.
L'autre facteur est obtenu par division :

$$\frac{8051}{97} = 83.$$

On obtient finalement :

$$8051 = 83 \times 97.$$

Cet exemple montre que la méthode ρ de Pollard permet de factoriser efficacement des entiers de taille modérée. Cependant, pour les modules RSA modernes contenant plusieurs centaines de chiffres décimaux, cette méthode devient insuffisante et doit être remplacée par des algorithmes plus performants comme le *Quadratic Sieve* ou le *General Number Field Sieve* [9].

L'amélioration progressive de ces techniques a conduit à une augmentation régulière des tailles de clés recommandées pour RSA. Les sous-sections suivantes présentent les deux algorithmes les plus importants dans l'histoire de la factorisation moderne.

5.2.1 Crible quadratique

Le *Crible quadratique* (QS), proposé par Carl Pomerance au début des années 1980, a longtemps représenté l'algorithme le plus efficace pour la factorisation des grands entiers.

Son principe repose sur la recherche de deux entiers (x) et (y) vérifiant :

$$x^2 \equiv y^2 \pmod{N},$$

$$x \not\equiv \pm y \pmod{N}$$

Lorsqu'une telle congruence est obtenue, un facteur non trivial de (N) peut généralement être calculé à l'aide de :

$$pgcd(x - y, N).$$

Phase de criblage Pour construire ces congruences, le QS recherche un grand nombre de relations entre des entiers dont les décompositions en facteurs premiers sont connues. Il utilise pour cela un crible permettant d'identifier rapidement les valeurs friables relativement à une base de facteurs.

Algèbre linéaire modulo 2 Les relations obtenues sont ensuite représentées sous forme de vecteurs d'exposants modulo 2. Une étape d'algèbre linéaire permet de trouver une dépendance entre ces vecteurs afin de construire une congruence de carrés exploitable.

Exemple Considérons à nouveau l'entier :

$$N = 8051.$$

On remarque que :

$$90^2 = 8100,$$

d'où

$$90^2 - 8051 = 49 = 7^2.$$

On obtient donc :

$$90^2 \equiv 7^2 \pmod{8051}.$$

Ainsi,

$$x = 90 \qquad \text{et} \qquad y = 7.$$

Comme $90 \not\equiv 7 \pmod{8051}$ et $90 \not\equiv -7 \pmod{8051}$, on peut calculer :

$$pgcd(90 - 7, 8051) = pgcd(83, 8051) = 83.$$

On obtient immédiatement un facteur non trivial de (8051).

L'autre facteur est alors :

$$\frac{8051}{83} = 97.$$

Ainsi,

$$8051 = 83 \times 97.$$

Cet exemple illustre l'idée fondamentale du *Quadratic Sieve* : construire une congruence de carrés permettant d'obtenir un facteur à l'aide d'un calcul de PGCD. Sur de grands entiers, la difficulté réside dans la recherche automatique de telles congruences.

Le *Quadratic Sieve* est nettement plus efficace que les méthodes élémentaires telles que la méthode ρ de Pollard. Là où Pollard cherche un facteur de N en testant des valeurs au hasard, le QS construit systématiquement des congruences de carrés en exploitant la structure arithmétique de N. Cette approche organisée permet de factoriser des entiers bien plus grands, ce qui explique pourquoi le QS a permis de nombreux records de factorisation dans les années 1980 et 1990 et constitue une étape majeure dans l'histoire de la cryptanalyse de RSA.

Cependant, lorsque N dépasse quelques centaines de chiffres décimaux, ses performances sont dépassées par celles du *General Number Field Sieve*, qui est aujourd'hui l'algorithme de référence [1].

5.2.2 Crible algébrique

Le *Crible algébrique* (GNFS) est actuellement le meilleur algorithme classique connu pour factoriser de grands entiers sans structure particulière. Boneh le présente comme l'état de l'art de la factorisation des grands nombres composites [1].

Contrairement au *Crible quadratique*, qui travaille uniquement dans $\mathbb{Z}$, le GNFS effectue ses calculs dans des corps de nombres algébriques. L'idée centrale reste la même : construire deux entiers x et y tels que

$$x^2 \equiv y^2 \pmod{N}, \qquad x \not\equiv \pm y \pmod{N},$$

car on peut alors obtenir un facteur non trivial de N en calculant $pgcd(x - y, N)$. Ce que le GNFS améliore, c'est la méthode pour construire ces congruences : en travaillant dans une structure algébrique plus riche que $\mathbb{Z}$, il trouve ces relations beaucoup plus rapidement que le QS lorsque N est très grand.

L'algorithme se déroule en trois grandes étapes :

1. **Sélection polynomiale.** On définit la fonction quadratique :

$$Q(x) = \left(\lceil\sqrt{N}\rceil + x\right)^2 - N.$$

Cette construction garantit que le membre gauche est un carré parfait, tandis que le membre droit $Q(x)$ reste relativement petit. Pour $N = 87463$, on a $\sqrt{87463} \approx 295{,}74$, donc $\lceil\sqrt{87463}\rceil = 296$, et :

$$Q(x) = (296 + x)^2 - 87463.$$

On obtient ainsi des relations de la forme :

$$(296 + x)^2 \equiv Q(x) \pmod{87463}.$$

L'objectif est de trouver un sous-ensemble de ces relations dont le produit donne une congruence de carrés modulo N.

2. **Recherche des nombres B-friables.** Un nombre est dit *B-friable* s'il n'admet aucun facteur premier supérieur à B. On fixe une borne B et on ne conserve que les valeurs $Q(x)$ qui sont B-friables. On appelle *base de facteurs* l'ensemble des nombres premiers inférieurs ou égaux à B.

Pour $N = 87463$ et $B = 19$, la base de facteurs est $\{2, 3, 5, 7, 11, 13, 17, 19\}$. On calcule les premières valeurs :

x	$(296 + x)^2 - 87463$	Factorisation	Friable ?
0	153	$3^2 \times 17$	oui
1	746	2×373	non $(373 > 19)$
2	1341	$3^2 \times 149$	non $(149 > 19)$
3	1938	$2 \times 3 \times 17 \times 19$	oui

En poursuivant le crible sur un intervalle plus large, on trouve d'autres valeurs friables, notamment pour $x = 20$:

$$316^2 - 87463 = 12393 = 3^6 \times 17.$$

Plutôt que de tester les valeurs une par une, le crible améliore cette recherche en travaillant sur un intervalle entier : pour chaque premier p de la base de facteurs, on repère périodiquement les indices x pour lesquels $p \mid Q(x)$, ce qui permet de traiter toutes les valeurs simultanément et beaucoup plus rapidement.

3. **Algèbre linéaire.** Une fois suffisamment de valeurs B-friables collectées, on cherche un sous-ensemble dont le produit est un carré parfait. Le produit de plu-

sieurs relations est un carré si et seulement si chaque exposant premier apparaît un nombre pair de fois dans le produit total.

Nous disposons des deux relations friables suivantes :

Relation	Factorisation	Vecteur d'exposants
$R_1 : \ 296^2 \equiv 153 \pmod{87463}$	$3^2 \times 17^1$	$(0,\ 2,\ 0,\ 0,\ 0,\ 0,\ 1,\ 0)$
$R_2 : \ 316^2 \equiv 12393 \pmod{87463}$	$3^6 \times 17^1$	$(0,\ 6,\ 0,\ 0,\ 0,\ 0,\ 1,\ 0)$

Chaque vecteur représente les exposants de $2, 3, 5, 7, 11, 13, 17, 19$. On cherche une combinaison dont la somme est nulle modulo 2. La matrice des exposants modulo 2 est :

$$M = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \pmod 2$$

Élimination de Gauss modulo 2. On ajoute R_1 à R_2 :

$$L_2 \leftarrow L_2 + L_1 = (0,0,0,0,0,0,1,0) + (0,0,0,0,0,0,1,0) = (0,0,0,0,0,0,0,0) \pmod 2.$$

La ligne R_2 devient le vecteur nul. Cela signifie que la combinaison $R_1 + R_2$ produit tous les exposants pairs. On vérifie :

$$296^2 \cdot 316^2 \equiv (3^2 \cdot 17^1)(3^6 \cdot 17^1) \equiv 3^8 \cdot 17^2 \pmod{87463}.$$

Tous les exposants sont bien pairs. On extrait la racine carrée :

$$x \equiv 296 \times 316 \bmod (87463) \equiv 93536 \bmod (87463) = 6073,$$

$$y = \sqrt{3^8 \cdot 17^2} = 3^4 \times 17 = 81 \times 17 = 1377.$$

On exploite l'identité $x^2 - y^2 = (x - y)(x + y)$ et on calcule :

$$d = pgcd(x - y,\, N) = pgcd(6073 - 1377,\, 87463) = pgcd(4696,\, 87463).$$

On applique l'algorithme d'Euclide :

$$\begin{aligned} 87463 &= 18 \times 4696 + 2935, \\ 4696 &= 1 \times 2935 + 1761, \\ 2935 &= 1 \times 1761 + 1174, \\ 1761 &= 1 \times 1174 + 587, \\ 1174 &= 2 \times 587 + 0. \end{aligned}$$

Le dernier reste non nul est 587. On obtient :

$$pgcd(4696,\, 87463) = 587.$$

L'autre facteur est :

$$\frac{87463}{587} = 149.$$

On obtient la factorisation :

$$\boxed{87463 = 587 \times 149.}$$

Boneh précise que toute attaque sur RSA prenant plus de temps que le GNFS n'a pas d'intérêt pratique [1] : il est alors plus efficace de factoriser directement N. Le GNFS constitue ainsi la référence à laquelle toutes les autres attaques contre RSA doivent être comparées.

Les records modernes de factorisation d'entiers RSA ont presque tous été obtenus grâce à des variantes du GNFS. Ces résultats servent à déterminer les tailles minimales de clés recommandées : aujourd'hui, les modules RSA de 2048 bits ou plus sont généralement considérés comme sûrs face aux méthodes classiques de factorisation [1].

À plus long terme, l'apparition d'ordinateurs quantiques capables d'exécuter l'algorithme de Shor pourrait remettre en cause la sécurité de RSA en permettant une factorisation bien plus rapide que le GNFS. Cette menace constitue l'une des principales motivations du développement de la cryptographie post-quantique.

5.3 Attaques élémentaires

5.3.1 Attaque par module commun

Cette attaque apparaît lorsque plusieurs utilisateurs partagent le même module RSA N, mais utilisent des exposants publics différents.

Supposons qu'un même message M soit envoyé à deux utilisateurs partageant le même module RSA N. Le premier possède la paire de clés (e_B, d_B) et le second la paire (e_C, d_C), où

$$e_B d_B \equiv 1 \pmod{\varphi(N)}$$

et

$$e_C d_C \equiv 1 \pmod{\varphi(N)}.$$

Les clés publiques sont donc (N, e_B) et (N, e_C). Les textes chiffrés obtenus sont alors

$$C_B \equiv M^{e_B} \pmod{N}$$

et

$$C_C \equiv M^{e_C} \pmod{N}.$$

Si les exposants publics sont premiers entre eux, c'est-à-dire

$$pgcd(e_B, e_C) = 1,$$

alors le théorème de Bézout garantit l'existence d'entiers s_1 et s_2 tels que

$$e_B s_1 + e_C s_2 = 1.$$

Un attaquant connaissant C_B, C_C, e_B et e_C peut alors calculer

$$C_B^{s_1} C_C^{s_2}.$$

En remplaçant les expressions des textes chiffrés, on obtient

$$C_B^{s_1} C_C^{s_2} = (M^{e_B})^{s_1} (M^{e_C})^{s_2} = M^{e_B s_1} M^{e_C s_2} = M^{e_B s_1 + e_C s_2}.$$

Or,

$$e_B s_1 + e_C s_2 = 1,$$

donc

$$C_B^{s_1} C_C^{s_2} = M.$$

L'attaquant retrouve ainsi directement le message clair sans connaître aucune clé privée.

Cette attaque montre qu'un même module RSA ne doit jamais être partagé entre plusieurs utilisateurs. En pratique, chaque utilisateur génère son propre module $N = pq$.

5.3.2 Factorisation par racines carrées non triviales

Rappelons que dans RSA les exposants sont construits de manière à satisfaire la relation

$$ed \equiv 1 \pmod{\varphi(N)},$$

où e est l'exposant public, d l'exposant privé et $\varphi(N)$ la fonction indicatrice d'Euler. Il existe donc un entier k tel que

$$\alpha = ed - 1 = k\,\varphi(N).$$

$\varphi(N)$ est un multiple de 4. Puisque p et q sont des nombres premiers impairs, les entiers $p-1$ et $q-1$ sont pairs. Il existe donc deux entiers u et v tels que $p-1=2u$ et $q-1=2v$. On obtient alors :

$$\varphi(N) = (p-1)(q-1) = (2u)(2v) = 4uv,$$

ce qui montre que $\varphi(N)$ est un multiple de 4, et par conséquent $\alpha = k\varphi(N)$ est également un multiple de 4. En particulier, α est pair et on peut écrire $\alpha = 2m$ avec $m = \alpha/2 \in \mathbb{Z}$.

$b^\alpha \equiv 1 \pmod N$. Soit $b \in \mathbb{Z}$ tel que $\gcd(b, N) = 1$. D'après le théorème d'Euler :

$$b^{\varphi(N)} \equiv 1 \pmod N.$$

Donc :

$$b^\alpha = b^{k\varphi(N)} = \left(b^{\varphi(N)}\right)^k \equiv 1 \pmod N.$$

$x = b^{\alpha/2}$ est une racine carrée de 1 modulo N.

Puisque α est pair, on peut poser $x = b^{\alpha/2}$, et l'on a :

$$x^2 = b^\alpha \equiv 1 \pmod N.$$

Ainsi x est une racine carrée de 1 modulo N.

Les quatre racines carrées de 1 modulo N.

Puisque $N = pq$ avec p et q premiers, le théorème des restes chinois donne l'équivalence :

$$x^2 \equiv 1 \pmod N \iff \begin{cases} x^2 \equiv 1 \pmod p, \\ x^2 \equiv 1 \pmod q. \end{cases}$$

Or, modulo un nombre premier, l'équation $x^2 \equiv 1$ admet exactement deux solutions : $x \equiv 1$ et $x \equiv -1$. On obtient donc quatre systèmes possibles :

$$\text{(I)} \begin{cases} x \equiv 1 \pmod p, \\ x \equiv 1 \pmod q, \end{cases} \qquad \text{(II)} \begin{cases} x \equiv 1 \pmod p, \\ x \equiv -1 \pmod q, \end{cases}$$

$$\text{(III)}\ \begin{cases} x \equiv -1 \pmod{p}, \\ x \equiv 1 \pmod{q}, \end{cases} \qquad \text{(IV)}\ \begin{cases} x \equiv -1 \pmod{p}, \\ x \equiv -1 \pmod{q}. \end{cases}$$

D'après le théorème des restes chinois, chacun de ces systèmes admet une unique solution modulo N. Les solutions des systèmes (I) et (IV) sont respectivement $x \equiv 1$ et $x \equiv -1$ modulo N : ce sont les racines *triviales*. Les solutions des systèmes (II) et (III) sont des racines *non triviales*, notées β et $-\beta$.

Probabilité de succès. Parmi les quatre racines 1, -1, β, $-\beta$, deux sont non triviales. Pour un b choisi aléatoirement avec $pgcd(b, N) = 1$, la valeur $b^{\alpha/2}$ tombe sur chacune des quatre racines avec une probabilité approximativement égale. La probabilité que $b^{\alpha/2}$ soit une racine non triviale est donc d'au moins $\frac{1}{2}$.

Pourquoi le *pgcd* révèle un facteur. Supposons que $\beta \not\equiv \pm 1 \pmod{N}$. Puisque β est une racine carrée de 1 modulo N, on a :

$$\beta^2 \equiv 1 \pmod{N},$$

ce qui s'écrit :

$$\beta^2 - 1 \equiv 0 \pmod{N},$$

soit :

$$(\beta - 1)(\beta + 1) \equiv 0 \pmod{N}.$$

Donc $N \mid (\beta - 1)(\beta + 1)$.

Par hypothèse, $\beta \not\equiv 1 \pmod{N}$, donc $N \nmid (\beta - 1)$, et $\beta \not\equiv -1 \pmod{N}$, donc $N \nmid (\beta + 1)$.

Cela signifie que $N = pq$ partage ses facteurs entre $(\beta - 1)$ et $(\beta + 1)$: l'un contient p et l'autre contient q. Par conséquent :

$$pgcd(\beta - 1,\ N) \in \{p,\ q\},$$

ce qui donne directement la factorisation de N.

En pratique, lorsque $b^{\alpha/2} \not\equiv \pm 1 \pmod{N}$, on calcule :

$$pgcd\big(b^{\alpha/2} - 1,\ N\big) = p \quad \text{ou} \quad q,$$

et l'autre facteur s'obtient par division. Avec une probabilité d'au moins $\frac{1}{2}$, un b choisi aléatoirement suffit à factoriser N en un seul essai.

5.3.3 Attaque par masquage

L'attaque par masquage (*blinding attack*) exploite une propriété fondamentale du chiffrement RSA appelée **homomorphisme multiplicatif**. Cette propriété signifie que le chiffrement d'un produit est égal au produit des chiffrements :

$$(M_1 \cdot M_2)^e \equiv M_1^e \cdot M_2^e \pmod{N}.$$

Un attaquant peut exploiter cette propriété lorsqu'il a accès à un *serveur de déchiffrement* : un système distant capable de déchiffrer n'importe quel texte chiffré qui lui est soumis, sans que l'attaquant n'ait besoin de connaître la clé privée.

Scénario Supposons qu'un attaquant intercepte un texte chiffré

$$C \equiv M^e \pmod{N},$$

où M est le message clair qu'il cherche à retrouver. S'il soumet directement C à le serveur, celui-ci peut refuser de le déchiffrer (par exemple parce que C est identifié comme un message protégé). L'astuce consiste alors à *déguiser* C en un texte chiffré différent, dont le serveur accepte de renvoyer le déchiffrement.

Déroulement de l'attaque

1. L'attaquant choisit un entier aléatoire r tel que

   $$pgcd(r, N) = 1,$$

   de sorte que r soit inversible modulo N.

2. Il construit un nouveau texte chiffré en multipliant C par r^e :

   $$C' \equiv C \cdot r^e \pmod{N}.$$

   Grâce à l'homomorphisme multiplicatif de RSA, on obtient :

   $$C' \equiv M^e \cdot r^e \equiv (Mr)^e \pmod{N}.$$

   Ainsi, C' est le chiffrement légitime du message $Mr \bmod N$. Le serveur ne peut pas reconnaître que C' est lié à C, car r est aléatoire et C' semble être un texte chiffré quelconque.

3. L'attaquant soumet C' au serveur. Celui-ci calcule :

   $$(C')^d \equiv (Mr)^{ed} \equiv Mr \pmod{N},$$

puisque $ed \equiv 1 \pmod{\varphi(N)}$. Le serveur renvoie donc la valeur $Mr \bmod N$.

4. L'attaquant connaît r et son inverse modulaire $r^{-1} \bmod N$. Il retrouve alors le message initial en calculant :

$$M \equiv (Mr) \cdot r^{-1} \pmod{N}.$$

Exemple numérique Considérons les paramètres RSA suivants :

$$N = 143, \qquad e = 7, \qquad d = 103.$$

Supposons que le message chiffré intercepté soit

$$C = 48,$$

ce qui correspond au chiffrement de $M = 9$.
L'attaquant choisit $r = 2$. Il vérifie que $pgcd(2, 143) = 1$, puis calcule :

$$r^e = 2^7 = 128.$$

Il construit le texte masqué :

$$C' \equiv 48 \times 128 \equiv 6144 \equiv 6144 - 42 \times 143 = 6144 - 6006 = 138 \pmod{143}.$$

Il soumet $C' = 138$ à le serveur. Celui-ci calcule :

$$138^{103} \bmod 143 = 18.$$

L'attaquant calcule ensuite l'inverse de $r = 2$ modulo 143. Comme $2 \times 72 = 144 \equiv 1 \pmod{143}$, on obtient $r^{-1} = 72$. Il retrouve alors le message :

$$M \equiv 18 \times 72 \equiv 1296 \equiv 1296 - 9 \times 143 = 1296 - 1287 = 9 \pmod{143}.$$

On retrouve bien $M = 9$.

Conséquences pratiques Cette attaque montre que le RSA « brut » (*textbook RSA*) n'est pas sûr face aux attaques à texte chiffré choisi. Un attaquant disposant d'un serveur de déchiffrement peut retrouver n'importe quel message en soumettant une version masquée du texte chiffré.

Pour contrer cette vulnérabilité, les implémentations modernes utilisent des schémas de bourrage (*padding*) tels que OAEP (*Optimal Asymmetric Encryption Padding*),

qui introduisent de l'aléa avant le chiffrement et rendent inexploitable la propriété multiplicative de RSA.

5.4 Attaque de Bleichenbacher (PKCS#1)

L'attaque de Bleichenbacher, publiée en 1998 [10], est l'une des attaques pratiques les plus importantes contre RSA. Elle illustre comment une vulnérabilité dans le *schéma de bourrage* peut suffire à compromettre entièrement la sécurité d'un système, même lorsque les paramètres mathématiques de RSA sont correctement choisis.

Contexte : le bourrage PKCS#1

En pratique, un message M n'est jamais chiffré directement avec RSA. Avant le chiffrement, on lui applique un *schéma de bourrage* (*padding*) afin d'atteindre la taille du module N et d'ajouter de l'aléa. Le standard PKCS#1 impose le format suivant pour un module de k octets :

$$\underbrace{\texttt{0x00}}_{1\text{ octet}} \;\Big|\; \underbrace{\texttt{0x02}}_{1\text{ octet}} \;\Big|\; \underbrace{r_1 r_2 \cdots r_{k-|M|-3}}_{\geq 8\text{ octets aléatoires non nuls}} \;\Big|\; \underbrace{\texttt{0x00}}_{1\text{ octet}} \;\Big|\; \underbrace{M}_{|M|\text{ octets}}$$

Lorsque le destinataire reçoit un texte chiffré, il le déchiffre puis vérifie que les deux premiers octets du résultat sont bien `0x00 0x02`. Si ce n'est pas le cas, le serveur renvoie un message d'erreur distinct. C'est précisément cette réponse différenciée valide ou invalide qui permet à l'attaquant d'obtenir une information partielle sur le message déchiffré.

Intervalle de validité

Un message correctement bourré selon PKCS#1 commence par les octets `0x00 0x02`, ce qui signifie que sa valeur numérique m vérifie :

$$2B \leq m \leq 3B - 1,$$

où $B = 2^{8(k-2)}$ et k est la taille du module en octets.

Pour un module de $k = 3$ octets, on obtient $B = 2^8 = 256$, et l'intervalle de validité est $[512,\ 767]$.

Chaque fois que le serveur répond « valide », l'attaquant sait que $Ms \bmod N$ tombe dans cet intervalle, ce qui lui permet de restreindre progressivement l'ensemble des valeurs possibles de M.

Principe de l'attaque

L'attaque est une attaque à texte chiffré choisi adaptative (*adaptive chosen-ciphertext attack*). Elle exploite la même propriété multiplicative que l'attaque par masquage : pour tout entier s, le texte modifié

$$C' \equiv C \cdot s^e \pmod{N}$$

est le chiffrement de $Ms \bmod N$. Le serveur déchiffre C', vérifie le bourrage du résultat, et répond :

— **valide** si $2B \leq Ms \bmod N \leq 3B - 1$;

— **invalide** sinon.

L'attaquant envoie au serveur une succession de textes modifiés avec des valeurs $s_1, s_2, s_3, \ldots$ croissantes. À chaque réponse « valide », il affine l'intervalle contenant M.

Déroulement de l'attaque

L'attaquant maintient à chaque étape un ensemble $\mathcal{I}$ d'intervalles contenant $M \bmod N$.

1. **Initialisation.** On pose
$$\mathcal{I}_0 = [2B,\, 3B - 1],$$
puisque le message M est supposé correctement bourré.

2. **Recherche d'un multiplicateur valide.** L'attaquant choisit un entier s_i et soumet
$$C' = C \cdot s_i^e \bmod N$$
au serveur. Une réponse valide implique l'existence d'un entier $r \in \mathbb{Z}$ tel que
$$2B \leq Ms_i - rN \leq 3B - 1.$$
Cette condition provient du fonctionnement du déchiffrement RSA modulo N.
En effet, on a
$$C' = C \cdot s_i^e \bmod N.$$
Après déchiffrement, le serveur calcule :
$$(C')^d \bmod N.$$
Or, par les propriétés de RSA :
$$C^d \equiv M \pmod{N} \qquad \text{et} \qquad (s_i^e)^d \equiv s_i \pmod{N}.$$

Ainsi,

$$(C')^d \equiv Ms_i \pmod{N}.$$

Par définition de la congruence modulo N, il existe un entier $r \in \mathbb{Z}$ tel que

$$(C')^d = Ms_i - rN.$$

Comme le schéma de bourrage impose que la valeur déchiffrée appartienne à l'intervalle valide, on obtient :

$$2B \leq (C')^d \leq 3B - 1.$$

En remplaçant $(C')^d$ par $Ms_i - rN$, on en déduit :

$$2B \leq Ms_i - rN \leq 3B - 1.$$

3. **Réécriture sous forme d'inégalité sur M.**

   On part de l'inégalité :

$$2B \leq Ms_i - rN \leq 3B - 1.$$

   On ajoute rN aux trois membres :

$$2B + rN \leq Ms_i \leq 3B - 1 + rN.$$

   On divise ensuite par $s_i > 0$:

$$\frac{2B + rN}{s_i} \leq M \leq \frac{3B - 1 + rN}{s_i}.$$

   Ainsi, pour chaque entier r, on obtient que M appartient à l'intervalle :

$$\left[\frac{2B + rN}{s_i}, \frac{3B - 1 + rN}{s_i}\right].$$

4. **Réduction de l'intervalle.**

   Pour chaque intervalle $[a, b] \in \mathcal{I}$, on cherche les valeurs de r telles que les deux intervalles se recouvrent, c'est-à-dire :

$$[a, b] \cap \left[\frac{2B + rN}{s_i}, \frac{3B - 1 + rN}{s_i}\right] \neq \emptyset.$$

Cette condition est équivalente à l'existence d'un $M \in [a, b]$ tel que :

$$\frac{2B + rN}{s_i} \leq M \leq \frac{3B - 1 + rN}{s_i}.$$

On en déduit les deux inégalités :

$$a \leq \frac{3B - 1 + rN}{s_i} \qquad \text{et} \qquad b \geq \frac{2B + rN}{s_i}.$$

On isole r dans chaque cas :

$$as_i \leq 3B - 1 + rN \quad \Longrightarrow \quad r \geq \frac{as_i - 3B + 1}{N},$$

$$bs_i \geq 2B + rN \quad \Longrightarrow \quad r \leq \frac{bs_i - 2B}{N}.$$

Ainsi, les valeurs admissibles de r vérifient :

$$\frac{as_i - 3B + 1}{N} \leq r \leq \frac{bs_i - 2B}{N}.$$

Pour chaque tel r, on obtient un nouvel intervalle pour M :

$$\left[\left\lceil \frac{2B + rN}{s_i} \right\rceil, \left\lfloor \frac{3B - 1 + rN}{s_i} \right\rfloor \right].$$

5. **Convergence.** On répète les étapes précédentes pour différents s_i. À chaque itération, l'ensemble $\mathcal{I}$ se réduit jusqu'à devenir singleton :

$$\mathcal{I} = \{M\}.$$

L'attaquant retrouve alors le message M sans connaître la clé privée d.

Exemple numérique

Considérons les paramètres suivants :

$$N = 160901, \qquad e = 7, \qquad k = 3, \qquad B = 256.$$

L'intervalle de validité est $[2B, 3B - 1] = [512, 767]$. Le message bourré vaut $M = 530$, et le texte chiffré intercepté est $C = 530^7 \bmod 160901$.

Itération 1 : $s_1 = 305$.
L'attaquant envoie au serveur $C' \equiv C \cdot 305^7 \pmod{160901}$. Le serveur déchiffre C'

et obtient :

$$530 \times 305 \bmod 160901 = 161650 \bmod 160901 = 749.$$

Comme $512 \leq 749 \leq 767$, le serveur répond « valide ».

On cherche r tel que $2B \leq 530 \times 305 - r \times 160901 \leq 3B - 1$, soit :

$$\frac{530 \times 305 - 767}{160901} \leq r \leq \frac{530 \times 305 - 512}{160901},$$

$$\frac{160883}{160901} \leq r \leq \frac{161138}{160901},$$

$$0{,}9999 \leq r \leq 1{,}0015,$$

donc $r = 1$. Le nouvel intervalle est :

$$\left[\left\lceil \frac{512 + 160901}{305} \right\rceil,\ \left\lfloor \frac{767 + 160901}{305} \right\rfloor\right] = \left[\left\lceil \frac{161413}{305} \right\rceil,\ \left\lfloor \frac{161668}{305} \right\rfloor\right] = [529,\ 530].$$

Après une seule itération, l'intervalle contenant M est réduit à $[529,\ 530]$.

Itération 2 : $s_2 = 609$.

L'attaquant envoie au serveur $C' \equiv C \cdot 609^7 \pmod{160901}$. Le serveur déchiffre C' et obtient :

$$530 \times 609 \bmod 160901 = 322770 \bmod 160901.$$

$$322770 - 2 \times 160901 = 968.$$

Comme $968 > 767$, le serveur répond « invalide ». Cette réponse négative n'apporte pas de restriction supplémentaire à l'intervalle courant ; l'attaquant essaie la valeur suivante.

Itération 3 : $s_3 = 610$.

$$530 \times 610 \bmod 160901 = 323300 \bmod 160901.$$

$$323300 - 2 \times 160901 = 1498.$$

Toujours invalide. On continue.

Itération 4 : $s_4 = 911$.

$$530 \times 911 \bmod 160901 = 482830 \bmod 160901.$$

$$482830 - 3 \times 160901 = 127.$$

Invalide ($127 < 512$).

Itération 5 : $s_5 = 1213$.

$$530 \times 1213 \bmod 160901 = 642890 \bmod 160901.$$

$$642890 - 3 \times 160901 = 160187.$$

Invalide ($160187 > 767$). On cherche encore.

Itération 6 : $s_6 = 1819$**.**

$$530 \times 1819 \bmod 160901 = 964070 \bmod 160901.$$

$$964070 - 5 \times 160901 = 159565.$$

Invalide.

Itération 7 : $s_7 = 2126$**.**

$$530 \times 2126 \bmod 160901 = 1126780 \bmod 160901.$$

$$1126780 - 7 \times 160901 = 607.$$

Comme $512 \leq 607 \leq 767$, le serveur répond « valide ».

On cherche r :

$$\frac{530 \times 2126 - 767}{160901} \leq r \leq \frac{530 \times 2126 - 512}{160901},$$

$$\frac{1126013}{160901} \leq r \leq \frac{1126268}{160901},$$

$$6{,}9981 \leq r \leq 6{,}9997,$$

donc $r = 6$ n'est pas dans cet intervalle et $r = 7$ non plus exactement. Recalculons :

$$\frac{1126013}{160901} = 6{,}9981\ldots \qquad \frac{1126268}{160901} = 6{,}9997\ldots$$

Aucun entier dans [6,998, 6,999]. Cela signifie que la valeur $s_7 = 2126$ ne restreint pas l'intervalle malgré la réponse valide, car le résidu r n'est pas entier. On cherche une valeur s donnant r entier.

Itération 8 : $s_8 = 2429$**.**

$$530 \times 2429 \bmod 160901 = 1287370 \bmod 160901.$$

$$1287370 - 8 \times 160901 = 1162.$$

Invalide.

Itération 9 : $s_9 = 3040$**.**

$$530 \times 3040 \bmod 160901 = 1611200 \bmod 160901.$$

$$1611200 - 10 \times 160901 = 190.$$

Invalide.

Itération 10 : $s_{10} = 3343$.

$$530 \times 3343 \bmod 160901 = 1771790 \bmod 160901.$$

$$1771790 - 11 \times 160901 = 579.$$

Comme $512 \leq 579 \leq 767$, le serveur répond « valide ».

On cherche r :

$$\frac{530 \times 3343 - 767}{160901} \leq r \leq \frac{530 \times 3343 - 512}{160901},$$

$$\frac{1771023}{160901} \leq r \leq \frac{1771278}{160901},$$

$$11{,}0007\ldots \leq r \leq 11{,}0023\ldots$$

Donc r n'est pas entier dans cet intervalle non plus. En pratique, les itérations se poursuivent automatiquement jusqu'à trouver une valeur s pour laquelle r est entier et l'intervalle se réduit. On admet ici que la convergence finale est atteinte après quelques dizaines d'itérations supplémentaires, ramenant l'intervalle [529, 530] à la valeur unique $M = 530$.

Cet exemple illustre le mécanisme fondamental de l'attaque : chaque réponse « valide » du serveur restreint l'intervalle contenant M. Sur un module RSA réel de 1024 bits, Bleichenbacher a montré que ce processus nécessite environ 10^6 requêtes envoyées au serveur, un nombre tout à fait réalisable contre un serveur répondant automatiquement.

Impact historique

Lors de sa publication en 1998, cette attaque a directement affecté les protocoles SSL v3 et TLS 1.0, dans lesquels le serveur renvoyait un message d'erreur différent selon que le bourrage était valide ou non. Des variantes de cette attaque, connues sous le nom de *ROBOT* (*Return Of Bleichenbacher's Oracle Threat*), ont été redécouvertes en 2017 et ont affecté de nombreux serveurs HTTPS majeurs [11].

Contre-mesures

- **Utiliser OAEP.** Le schéma OAEP (*Optimal Asymmetric Encryption Padding*), défini dans PKCS#1 v2.1, est prouvablement sûr contre les attaques à texte chiffré choisi et doit remplacer PKCS#1 pour tout nouveau déploiement.
- **Ne pas distinguer les erreurs.** Si PKCS#1 doit être conservé pour des raisons de compatibilité, le serveur doit renvoyer exactement la même réponse qu'il s'agisse d'une erreur de bourrage ou non, afin de ne révéler aucune information à l'attaquant.
- **Implémentation en temps constant.** La vérification du bourrage doit s'exécuter en un temps indépendant du résultat, afin d'éviter toute fuite d'information par canal auxiliaire temporel.

L'attaque de Bleichenbacher illustre un principe fondamental : la robustesse mathématique de RSA ne suffit pas. La moindre information révélée par le comportement d'un serveur même aussi simple qu'un message d'erreur binaire peut suffire à compromettre entièrement le système.

6 Conclusion

Ce rapport présente une étude du cryptosystème RSA, depuis ses fondements mathématiques jusqu'à l'étude de certaines vulnérabilités.

Les outils mathématiques. Nous avons développé les outils arithmétiques nécessaires à la compréhension de RSA : arithmétique modulaire, algorithme d'Euclide étendu pour le calcul des inverses modulaires, fonction indicatrice d'Euler $\varphi(n)$, théorème d'Euler, théorème des restes chinois et exponentiation modulaire rapide. Ces outils ne sont pas de simples prérequis formels : chacun joue un rôle précis dans la génération des clés, le chiffrement, le déchiffrement ou les optimisations du système.

L'algorithme RSA. Le cryptosystème RSA repose sur un principe élégant : la multiplication de deux grands nombres premiers p et q est facile à réaliser, mais retrouver p et q à partir de leur produit $N = pq$ est supposé difficile. Cette asymétrie permet de construire une paire de clés (N, e) et (N, d) telle que tout message chiffré par $M^e \bmod N$ ne peut être retrouvé qu'en calculant $C^d \bmod N$. Le théorème d'Euler garantit la correction de ce mécanisme. RSA permet également de produire des signatures numériques, assurant authenticité, intégrité et non-répudiation.

Les attaques. L'analyse de sécurité a montré que la robustesse mathématique de RSA ne suffit pas. Nous avons étudié plusieurs catégories d'attaques. La factorisation directe du module N, réalisable via la méthode ρ de Pollard, le Crible quadratique ou

le Crible algébrique, reste le principal obstacle à surmonter pour un attaquant. Les attaques élémentaires comme le module commun ou l'attaque par masquage exploitent des erreurs de déploiement. L'attaque de Bleichenbacher [10] a montré qu'un simple message d'erreur renvoyé par un serveur peut suffire à retrouver un message chiffré après environ 10^6 requêtes, ce qui a directement affecté SSL et TLS. Ces attaques illustrent un principe fondamental : la sécurité d'un système cryptographique dépend autant de la qualité de son implémentation que de ses fondements mathématiques.

Perspectives. Aujourd'hui, RSA reste déployé massivement dans les certificats numériques, les infrastructures à clés publiques et les protocoles TLS. Les clés de 2048 bits ou plus sont considérées comme sûres face aux méthodes classiques de factorisation. Cependant, l'algorithme de Shor, exécutable sur un ordinateur quantique, permettrait de factoriser N en temps polynomial, ce qui rendrait RSA entièrement cassable. Cette menace motive le développement actif de la cryptographie post-quantique, dont les premiers standards ont été publiés par le NIST en 2024. La transition vers ces nouveaux algorithmes constituera l'un des défis majeurs de la sécurité informatique dans les années à venir.

Références

[1] D. Boneh, *Twenty Years of Attacks on the RSA Cryptosystem*, Notices of the AMS, vol. 46, n° 2, pp. 203–213, 1999.

[2] R. L. Rivest, A. Shamir et L. Adleman, *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, Communications of the ACM, vol. 21, n° 2, pp. 120–126, 1978.

[3] D. R. Stinson, *Cryptographie : Théorie et Pratique*, 3e édition, Vuibert, 2000.

[4] W. Diffie et M. E. Hellman, *New Directions in Cryptography*, IEEE Transactions on Information Theory, vol. 22, n° 6, pp. 644–654, 1976.

[5] M. J. Wiener, *Cryptanalysis of Short RSA Secret Exponents*, IEEE Transactions on Information Theory, vol. 36, n° 3, pp. 553–558, 1990.

[6] P. C. Kocher, *Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems*, CRYPTO '96, LNCS vol. 1109, pp. 104–113, Springer, 1996.

[7] EMVCo, *EMV Integrated Circuit Card Specifications for Payment Systems*, Version 4.3, 2011. https://www.emvco.com

[8] National Institute of Standards and Technology, *Advanced Encryption Standard (AES)*, FIPS Publication 197,2001.

[9] Eric W. Weisstein, Pollard Rho Factorization Method, 2025, https://mathworld.wolfram.com/PollardRhoFactorizationMethod.html

[10] D. Bleichenbacher, *Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS#1*, CRYPTO '98, LNCS vol. 1462, pp. 1–12, Springer, 1998.

[11] H. Böck, J. Somorovsky et C. Young, *Return Of Bleichenbacher's Oracle Threat (ROBOT)*, USENIX Security Symposium, 2018. https://robotattack.org