Table des matières

Liste des figures et tableaux

Table des figures

Liste des tableaux

Tableau des acronymes

Abréviation	**Signification**
RSA	Rivest Shamir Adelman
SHA	Secure Hash Algorithm
OAEP	Optimal Asymmetric Encryption Padding
PKCS	public Key Cryptographic Standards
AES	Advanced Encryption Standard
PEM	privacy Enhauced Mail
ASN 1	Abstract Syntax Notation One
PRNG	PseudoRandom Number Generator
UF-CMA	UnForgability Chosen-Message-Attacks
OW	One Way
CPA	Chosen Plaintext Attack
CCA	Chosen Ciphertext Attack

.

Introduction Générale

Dans la technologie de transmission d'information, l'homme a besoin d'une fiabilité au niveau de la communication des données. Alors qu'on constate une évolution constante des techniques, visant à sécuriser l'échange de ces données ou des techniques mises au point pour contourner ces systèmes sécurisés. De ce fait nous faisons appel aux chiffrements et aux signatures Uni(Bi)directionnels des données. Le chiffrement ou la signature uni(bi)directionnel repose sur les "fonctions proxy", c'est-à-dire les fonctions qui transforment le texte chiffré correspondant à une clé en texte chiffré pour une autre clé sans révéler toute information sur le décryptage. Dans le cas de la signature, le mandataire convertit une signature valide pour une clé en une signature d'une autre clé sans divulguer la signature secrète. Nous étendons et généralisons cette notion comme suit :

-Intuitivement, les fonctions proxy permettent à un utilisateur de décrypter des cryptogrammes ou générer des signatures valables au nom d'un autre utilisateur sans détenir aucune information sur la clé secrète de ce dernier utilisateur. Les fonctions proxy peuvent être divisées en deux catégories : unidirectionnelle et bidirectionnelle. -Les fonctions proxy unidirectionnelles permettent à un utilisateur U1, de décrypter des cryptogrammes ou générer des signatures correspondant au secret d'un autre utilisateur U2 même si le premier utilisateur ne peut pas détenir cette clé secrète. Cependant, le propriétaire du secret U2 a besoin d'une fonction unidirectionnelle complètement différente s'il souhaite décrypter des cryptogrammes ou générer des signatures au nom du premier utilisateur U1. Par contre les fonctions proxy bidirectionnelles peuvent être utilisées par les deux utilisateurs pour décrypter des cryptogrammes ou générer des signatures, en transformant le cryptogramme d'une clé en un cryptogramme pour un autre clé ou la signature d'une clé pour une autre clé. En d'autres termes, les deux utilisateurs U1 et U2 peuvent utiliser la même fonction proxy bidirectionnelle pour transformer les cryptogrammes d'une clé à l'autre.

Dans ce mémoire, nous allons vous présentez les systèmes génériques de procuration bidirectionnelle et unidirectionnelle, des fonctions de cryptage à clé publique, et les systèmes de signature, d'où le «**schéma uni(Bi)directionnel**». Tous les schémas génériques peuvent être utilisés pour transformer toute primitive cryptographique standard en une fonction proxy, avec un ralentissement d'un facteur deux. Ce ralentissement est éliminé par les fonctions de procuration spécifiquement conçu pour quelques primitives cryptographiques comme RSA, RSA Hash-and-Sign. La notion de cryptographie par procuration peut être très utile dans les cas où un utilisateur doit effectuer des opérations sensibles (par exemple, décryptage de texte

chiffré, génération de signature) sans détenir les clés secrètes nécessaires.

Le chiffrement et les techniques associés sont également utilisés pour développer une sécurité renforcée des transactions financières et pour protéger les communications privées d'utilisateurs finaux. Ces technologies permettent par exemple de déterminer si des données ont été altérées (intégrité des données) et de renforcer la confiance des utilisateurs dans le fait qu'ils communiquent avec les destinataires prévus (authentification), et elles font partie des protocoles qui apportent la preuve que les messages ont été envoyés et reçus (non-répudiation).

Contexte

Devenu incontournable pour la protection et la confidentialité des données, le chiffrement et la signature des données sont connus par tous grâce à des algorithmes de chiffrement de données modernes (comme RSA) permettant aussi de vérifier l'origine d'un message, et de s'assurer que le contenu du message n'a pas été changé depuis son expédition. Dans ce sens, un utilisateur doit être capable de signer ou de décrypter un message au nom d'un autre utilisateur sans connaître sa clé secrète . Cependant, le résultat de la signature d'un message m est une toute nouvelle signature qui combine les identités de l'utilisateur original et le véritable signataire. Dans la signature unidirectionnelle et bidirectionnel basée sur RSA le système permet de partager la clé secrète entre un utilisateur et un serveur de sorte qu'aucun des deux ne peut créer une clé valide sans travailler ensemble. Les preuves de sécurité reposent sur le fait que le serveur est toujours digne de confiance, ce qui permet d'obtenir des niveaux de sécurité raisonnable.

Problématique

Aujourd'hui avec l'explosion de l'Internet et la dématérialisation du monde numérique, les réseaux informatiques jouent un rôle prépondérant dans la vie de tous les jours et dans notre société en général. Ouverts au monde extérieur les réseaux informatiques deviennent des cibles potentielles pour les personnes malveillantes qui tentent d'exploiter les vulnérabilités de ces systèmes. La disponibilité généralisée du chiffrement, ainsi que sa nature polyvalente et son utilisation par différents acteurs, présente un certain nombre de difficultés :

• Comment déléguer à une personne de signer à sa place sans qu'elle puisse connaitre la clé secrète ?

• Comment déléguer a une personne de déchiffré à sa place sans connaitre la clé secrète ?

• Comment mandater une personne sans que cette dernière ne puisse abuser ? (c'est a dire la personne ne peut pas signer ou déchiffrer sans le secret au complet)

Objectifs

Dans un monde où l'accès aux données informatiques est la préoccupation première des hackers, la question de la sécurisation des données devient une évidence. Le chiffrement et la signature des données sont des tactiques disponibles pour assurer cette sécurité.

Pour bien mener notre travaille ce mémoire est scindé en trois chapitres : D'abord dans le premier chapitre, nous présenterons les notions de la cryptographie et les concepts de chiffrement et de la signature Uni(Bi)directionnel.

Ensuite dans le second chapitre, nous étudierons les schémas de chiffrement et de la signature Uni(Bi)directionnel basé sur RSA.

Enfin dans le troisième chapitre, nous nous focaliserons sur l'implémentation des schémas de chiffrement et de signature Uni(Bi)directionnel basé sur RSA.

1 Proxy Unidirectionnel et Bidirectionnel

Introduction :
L'évolution rapide des réseaux informatiques, privés ou publics, engendre un volume toujours plus important de données sauvegardées et transmises, générant ainsi de nouveaux besoins en matière de sécurité. Dans un monde où l'entreprise dépend de plus en plus de son système informatique, la sécurité est donc devenue une préoccupation primordiale. Dans ce contexte, la cryptographie demeure une partie indispensable de la sécurité informatique moderne. Néanmoins, tout utilisateur non averti peut ne pas comprendre l'utilité du chiffrement et risque de s'en passer.

Etant une science de coder des messages secrets, la cryptographie préserve le texte en clair de l'indiscrétion des criminels informatiques, fournissant non seulement la confidentialité des données mais aussi l'authentification, l'intégrité, la non répudiation et le contrôle d'accès.

Nous présentons dans ce chapitre les fonctions proxy de chiffrement et la de signature Uni(Bi)directionnel, exposons les propriétés que nous voulons en obtenir notamment en matière de sécurité. Nous passons en revue les différentes solutions et décrivons celles que nous avons retenu et qui sont basées sur un serveur de chiffrement.

1.1 Rappels Mathématiques

1.1.1 Indicatrice d'Euler

• **Définition**

L'indicatrice d'Euler (ou la fonction indicatrice d'Euler ou Euler totient en anglais), notée avec la lettre grecque phi : ϕ(n) ou Φ(n) est le nombre représentant le nombre d'entiers inférieurs à n qui sont premiers avec n

• **L'indicatrice d'Euler**

Φ (n) (indicatrice Euler) se calcule de plusieurs manières, la formule la plus connue est

$\Phi(\mathrm{n}) = n \prod_{p/n} (1 - 1/p)$

où p est un facteur premier qui divise n.

• **Théorème d'Euler**

La fonction indicatrice d'Euler (phi) est utilisée en arithmétique modulaire. Elle est notamment utilisée dans le Théorème d'Euler : Soit n est un entier supérieur à 1 et a un entier premier avec n, alors

$a^{\phi(n)} \equiv 1 \mod n$

1.1.2 Théorème de Fermat

• Petit théorème de Fermat

Le théorème de Fermat est un corollaire immédiat du théorème d'Euler. Il s'agit du cas particulier où n est un nombre premier.

Si a n'est pas divisible par p premier, alors

$a^{p-1} \equiv 1 \mod p$

• Grand théorème de Fermat

Soit n ≥ 3 un entier. Les solutions de $x^n + y^n = z^n$, où x, y, et z sont des entiers, vérifient toutes xyz = 0.

1.1.3 Notion de Groupe

Définition 1 Un groupe est un ensemble non vide muni d'une loi de composition interne (G, $*$) tels que :

(1) $*$ est associative ;

(2) $*$ admet un neutre e_G ;

(2) tout élément de G est symétrisable (admet un symétrique) pour $*$.

Si $*$ est commutative, on dit que (G, $*$) est commutatif, ou encore abélien.

Définition 2

Un sous-groupe d'un groupe (G, $*$) est une partie non vide H de G telle que :

(1) $*$ induit sur H une loi de composition interne.

(2) Muni de cette loi, H est un groupe.

On note alors : H < G.

1.1.4 Notion d'anneau

Définition 1 Soit A un ensemble, + et . deux lois de composition interne, définies sur A. Le triplet **(A, +, .)** est un anneau si les trois conditions suivantes sont vérifiées.

(1) (A, +) est un groupe commutatif ;

(2) la loi . est associative et admet un élément neutre ;

(3) $\forall$ (x, y, z) $\in$ A, x(y + z) = xy + xz et (x + y)z = xz + yz. On dit que la loi . est distributive par rapport à +.

A est un anneau commutatif si la loi . est commutative. Si la loi . ne possède pas d'élément neutre, on parle alors d'anneau non unitaire.

Définition 2

Soit (A, +, .) un anneau et A' $\subset$ A. On dit que (A', +, .) est un sous-anneau de A si :

(1) (A', +) est un sous-groupe de (A, +) ;

(2) 1 $\in$ A' ;

(2) A' est stable pour la loi .

Définition 3

Soit A et A' deux anneaux, et f une application de A dans A'. On dit que f est un homomorphisme d'anneaux de A dans A' si :

(1) $(x, y) \in$ A, $f(x + y) = f(x) + f(y) et f(x.y) = f(x).f(y)$;

(2) $f(1) = 1$.

1.1.5 Corps

Définition 1

Soit (K, +, .) un anneau unitaire. Si, $\forall$ x $\in K^*$, il existe x' $\in$ K, tel que x.x' = x'.x = 1, alors (K, +, .) est un corps. Si $\forall$(x, x') $\in K^2$, x.x' = x'.x, alors K est un corps commutatif.

Définition 2

Soit L un corps et K $\subset$ L, K est un sous-corps de L si :

(1) K est un sous-anneau de L

(2) K^* est stable par « passage à l'inverse ».

Définition 3

Soit K et K' deux corps et ϕ une application de K dans K'. On dit que ϕ est un homomorphisme de corps si

(1) $\forall$(x, y) $\in$ K, ϕ(x + y) = ϕ(x) + ϕ(y) et ϕ(x.y) = ϕ(x).ϕ(y) ;

(2) $\phi(1) = 1$.

1. Schéma Unidirectionnel

Ce schéma nous montre comment un utilisateur U peut déléguer a une personne F de déchiffré à sa place sans connaitre la clé secrète KU

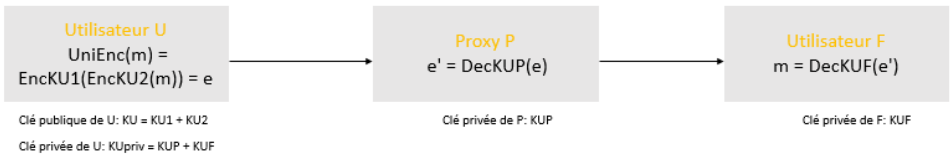


Figure 1 – Schéma du chiffrement Unidirectionnel

2. Schéma Bidirectionnel

Ce schéma nous montre comment un utilisateur U peut déléguer a une personne F de déchiffré à sa place sans connaitre la clé secrète KU et vice versa

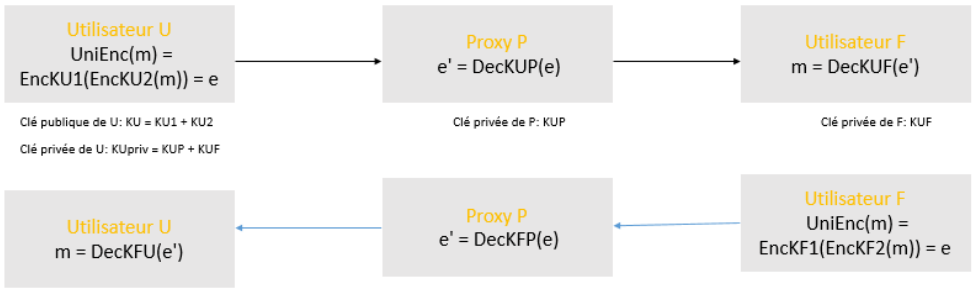


Figure 2 – Schéma du chiffrement Bidirectionnel

1.2 Généralité sur la Cryptographie

1.2.1 La cryptographie

La cryptographie[W3] permet la transformation, au moyen d'un algorithme de chiffrement, d'un message dit clair en un message chiffré. Grâce à ce procédé, deux interlocuteurs peuvent échanger de manière confidentielle et sécurisée, pourvu qu'ils possèdent la clé leur permettant de chiffrer ou de déchiffrer leurs messages. La cryptographie sert aussi d'autres applications

telles que l'authentification et la signature (numérique) des messages.Le chiffrement comprend le traitement,le stockage et la transmission sécurisée des donnéés Depuis longtemps, la transmission des données sensibles a nécessité l'utilisation d'un système de sécurisation performant.Les services secrets des grandes puissances économiques et Politiques ont développé tout d'abord des codages alpha-numériques simples, puis des techniques cryptographiques plus poussées, grâce à des l'outils mathématique pour rendre leurs données sensibles inviolables et inexploitables. La cryptologie, véritable science régissant le codage de l'information, a connu une réelle explosion avec le développement des systèmes informatiques, passant d'une ère artisanale et confidentielle à des systèmes de très hautes technologies nécessitant une importante puissance de calcul. Elle a connu un plus large essor encore avec l'arrivée des systèmes de communications modernes (internet, etc...) où il y a une nécessité absolue de protéger les données échangées pour respecter les individus.

Qu'entend-on par clé ?

On appelle clé une valeur utilisée dans un algorithme de cryptographie afin de chiffrer une donnée. Il s'agit en fait d'un nombre complexe dont la taille se mesure en bits. On peut imaginer que la valeur correspondante à 1024 bits est absolument gigantesque. Plus la clé est grande, plus elle contribue à élever la sécurité de la solution. Toutefois, c'est la combinaison d'algorithme complexe et de clés importantes qui seront la garantie d'une solution bien sécurisée. Les clés doivent être stockées de manière sécurisée de sort que seul leur propriétaire soit en mesure de les reçevoire et de les utiliser.

1.2.2 Terminologie de la cryptographie

• **Chiffrer** : transformer[3] à l'aide d'une clé de chiffrement un message en clair en un message incompréhensible sans la clé de déchiffrement

• **Déchiffrer** : retrouver à l'aide de la clé de déchiffrement le message en clair à l'origine du message chiffré

• **Chiffre** : algorithme utilisé pour le chiffrement

• **Cryptogramme** : message chiffré

• **Décrypter** : retrouver le message en clair correspondant à un cryptogramme sans connaitre la clé de déchiffrement ("casser" le code secret)

• **Cryptographie** : science de la protection des messages par des clés

• **Cryptanalyse** : science du décryptage

• **Cryptologie** : cryptographie et cryptanalyse

1.2.3 Symboles usuels

• Message en clair (plain-text) : P

- Cryptogramme (cipher-text) : C
- Clé de chiffrement (encryption-key) : ke
- Clé de déchiffrement (decryption-key) : kd
- Fonction de chiffrement (encrypt) : E(ke, P) = C
- Fonction de déchiffrement (decrypt) : D(kd, C) = P
- L'ensemble doit respecter : D(kd, E(ke, P)) = P
- Fonction de hachage (hash/message-digest) : M d(P) = H

1.2.4 Entités et canaux de communication

Un système[6] de communication sur lequel on doit appliquer la sécurité est composé au minimum de deux entités et d'un canal de transmission. Les notions utilisées dans ce système sont :

- **Une entité** : est une personne, organisation, ou dispositif qui envoie, reçoit ou manipule l'information.

- **Un émetteur** : est une entité légitime qui envoie l'information.

- **Un destinataire (ou récepteur)** : est une entité légitime qui reçoit l'information.

- **Un attaquant (ou adversaire)** : est une entité qui essaie de contrecarrer les mesures de sécurité. Les actions de l'attaquant peuvent être très variées en fonction de ses intentions et du système de communication utilisé. L'attaquant essaye par exemple de se faire passer pour le destinataire ou pour l'émetteur d'un message.

- **Un canal de communication** : est un media de transmission de l'information d'une entité à une autre.

- **Un canal physiquement sécurisé** : est un canal qui n'est pas physiquement accessible à l'attaquant.

- **Un canal public sécurisé** : est un canal qui ne doit pas être accessible à l'attaquant par des moyens cryptographiques. La figure 1 ci dessous , montre l'exemple d'un système de communication entre deux entités A et B, en présence d'un adversaire qui scrute le canal de transmission pour intercepter les messages transmis.

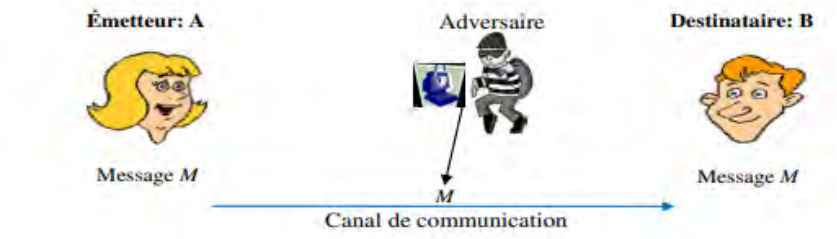


Figure 3 – Système de communication composé de deux entités et un adversaire

1.2.5 Fonctions cryptographiques et terminologies

Nous présentons[6] ci-dessous quelques définitions liées à la cryptographie.

- Espace de texte en clair ou message M, est un sous-ensemble de l'ensemble des chaînes binaires $\{0,1\}^*$. Par exemple chaque lettre de l'alphabet français peut être assignée à un mot de cinq bits (codage binaire).

- Espace de texte chiffré (ou cryptogramme) C, est également un sous-ensemble de $\{0,1\}^*$.

- Espace des clés K est aussi un sous-ensemble de $\{0,1\}^*$. Souvent $K = \{0,1\}^k$ où k est un paramètre de sécurité fixe représentant la clé.

- Fonction (ou algorithme) de chiffrement E, permet de calculer C avec différentes fonctions E, $C = E_K(M)$.

- Fonction (ou algorithme) de déchiffrement D, permet de retrouver le message en clair à partir du cryptogramme $M = D_k(C)$.

- Un cryptosystème est un procédé pour transformer un texte clair en un texte chiffré et inversement, il se compose de tous ce qui précède, c'est à dire (M, C, K, E, D). La figure, montre le schéma de principe d'un cryptosystème qui assure la confidentialité des communications entre Alice et Bob. Les deux entités commencent d'abord par l'échange d'une clé secrète par un moyen donné. Si Alice veut envoyer un message M à Bob, elle chiffre M avec la fonction E et la clé secrète k et l'envoie à Bob. Après avoir reçu le message chiffré C, Bob effectue le déchiffrement de ce message par la clé secrète déjà partagée et la fonction D afin de découvrir le message original M. L'attaquant peut intercepter le message C, et peut utiliser des techniques de cryptanalyse pour trouver le message M.

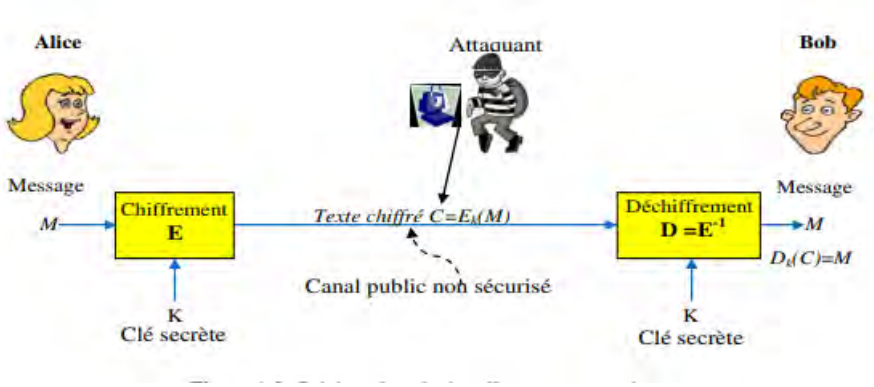


Figure 4 – Schéma de principe d'un cryptosystème

Le chiffrement symétrique est défini par $D_k(E_k(m)) = m$ Le chiffrement asymétrique est défini par $D_d(E_e(m)) = m$, où les clés e et d ne sont pas identiques, et, en plus d ne peut pas être dérivée facilement de e.

1.2.6 Principes et Objectifs de la Cryptographie

Principe de la Cryptographie

Le cryptage[W3] ou chiffrement des données est généralement décrit par une communication secrète d'information entre deux interlocuteurs. Dans un système informatique, cette confidentialité intervient en fait sous plusieurs formes, notamment dans la protection du stockage de l'information (copie de fichier), des accès (interrogation d'une base de données) et de sa transmission (écoute). Quelque soit son support physique, l'information est toujours représentée par un codage binaire. Ce codage est conforme a un standard qui permet de communiquer avec les dispositifs de la machine (clavier, écrans, imprimantes, traceurs. . . .). Lorsque les ensembles de données à stocker ou à transmettre sont très volumineux (en particulier les images), on fait appel a des technique de compression pour optimiser le volume et la vitesse de transmission. Le cryptage repose sur une transformation du code vers une forme non standard, de façon à limiter l'utilisation. La technique est très ancienne, mais elle a était largement développée et transformée avec l'utilisation intensive de l'informatique et des codages numériques. Ce domaine reste, pour des raisons évidentes, très secret, et les progrès les plus récents ne sont pas divulgués. Cette présentation se limite évidement au technique et outils du domaine public, qui sont ceux effectivement utilisés dans les grandes applications informatiques. Pour limiter les risques de divulgation, on doit pouvoir changer périodiquement de technique de cryptage.

Objectifs de la Cryptographie

L'objectif[W3] fondamental de la cryptographie est de Permettre à deux entités de communiquer sur un canal public peu sûr de telle sorte qu'une personne LAMDA ne puisse pas comprendre les données échangées. Un canal public est un canal de communication auquel tout le monde a accès et dont les communications sont susceptibles d'être écoutées par n'importe qui, sans trop de difficultés. Par exemple, le réseau téléphonique, le réseau Internet, mais aussi l'atmosphère (pour les fumées utilisées par les amérindiens pour communiquer à distance). La sécurité des systèmes d'information toute entière et la cryptographie poursuivent quatre objectifs essentiels qui, lorsqu'ils sont tous atteints, garantissent une sécurité optimale à l'utilisateur :

• Confidentialité : l'information ne peut être lue par une personne non autorisée.

• Authenticité : l'information est attribuée à son auteur légitime.

• Intégrité : l'information ne peut être modifiée par une personne non autorisée.

• Non-répudiation : l'information ne peut faire l'objet d'un déni de la part de son auteur.

Exemple : Alice[3] envoie un message confidentiel à Bob, Alice chiffre son message avec la clef publique de Bob C = E($kpub_B$, P) (cette clé est librement accessible).

• Elle transmet le message chiffré à Bob (l'interception de ce message est inexploitable).

• Bob utilise sa propre clé privée pour déchiffrer le message reçu D($kpriv_B$, C) = P (c'est le seul à pouvoir le faire).

Pour répondre : démarche réciproque

1.2.7 Systèmes de la cryptographie

Le chiffrement[W3] d'un message permet justement de garantir que seul l'émetteur et le(s) destinataire(s) légitime(s) d'un message en connaissent le contenu. C'est une sorte d'enveloppe scellée numérique. Une fois chiffré, faute d'avoir la clé spécifique, un message est inaccessible et illisible, que ce soit par les humains ou les machines. Il existe deux grandes familles de chiffrement : le chiffrement symétrique et le chiffrement asymétrique.

Chiffrement Symétrique Le chiffrement[W3] symétrique permet de chiffrer et de déchiffrer un contenu avec la même clé, appelée alors la « clé secrète ». Le chiffrement symétrique est particulièrement rapide mais nécessite que l'émetteur et le destinataire se mettent d'accord sur une clé secrète commune ou cela transmet par un autre canal. Celui-ci doit être choisi avec précautions, sans quoi la clé pourrait être récupérée par les mauvaises personnes, ce qui n'assurerait plus la confidentialité du message.

Dans le chiffrement symétrique les deux entités qui communiquent utilisent un algorithme de chiffrement/déchiffrement symétrique basé sur une même clé secrète k, comme montre la figure ci dessous. Les algorithmes de chiffrement symétrique sont divisés en deux classes principales : algorithmes de chiffrement par bloc et algorithmes de chiffrement par flux

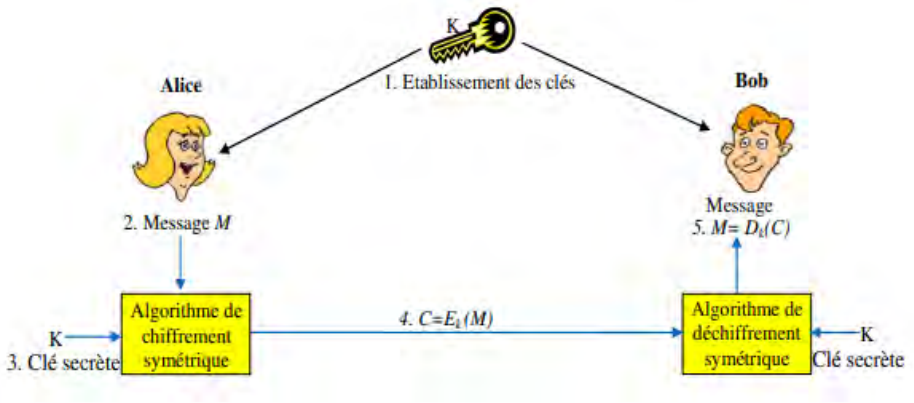


Figure 5 – Chiffrement/déchiffrement symétrique avec une clé secrète

Dans le chiffrement par bloc le texte en clair se décompose en blocs de longueur fixe et les blocs seront chiffrés les uns après les autres en utilisant une clé (de taille fixe). Ainsi, pour n'importe quelle clé fixée, le chiffrement par bloc est une bijection : $C = E_k(M)$, $M = D_k(C) = D_k(Ek_(m)) = m$. L'algorithme de chiffrement symétrique le plus connu et le plus robuste est l'AES (Advanced Encryption Standard). Il utilise une longueur de bloc de 128 bits et une taille (minimale) de clé de : 128, 192, et 256 bits. Les algorithmes de chiffrement par flux, traitent des données bit par bit ou mot par mot (un mot correspond à un octet ou un ensemble d'octets) par une technique simple, appelée one-time pad. A cet effet, un PRNG produit un masque binaire de taille égale au texte en clair pour effecteur le chiffrement. En outre, pour éviter la production du même masque (par l'utilisation d'une clé fixe), un compteur sera utilisé comme une entrée supplémentaire au générateur. Les algorithmes de chiffrement par flux sont considérés moins robustes, mais plus rapides que les algorithmes de chiffrement par bloc. Dans le cas du chiffrement symétrique, nous savons que les deux entités, Alice et Bob, avant d'échanger des messages en toute sécurité, doivent se mettre d'accord sur une clé secrète k. Ceci peut se faire suite à une rencontre physique entre Alice et Bob, mais cela n'est pas toujours possible. Donc, il se pose le problème important d'échange de clé en toute sécurité sans être interceptée.

Chiffrement Asymétrique L'idée[6] de chiffrement à clé publique est

simple : nous utilisons des clés différentes pour le chiffrement et le déchiffrement, une publique et l'autre privée. Seule la clé publique e doit être disponible à tout le monde, tandis que la clé privée d doit être gardée secrète par son propriétaire. Ces deux clés sont reliées mathématiquement, et il est pratiquement impossible de calculer la clé d à partir de la clé e. Dans la figure ci-dessous.

• Alice veut envoyer à Bob un message sécurisé avec un algorithme de chiffrement asymétrique.

• Au début, Bob choisi une paire de clés (e_B, d_B) et envoie sa clé publique e_B à Alice.

• Alice chiffre le message M avec la clé e_B et l'envoie à Bob.

• Lorsque Bob reçoit le message chiffré, il utilise sa clé privée, d_B, pour le déchiffrer.

Il est important de noter qu'il ne suffit pas que la clé de chiffrement e_B soit disponible au public, il faut en plus qu'elle soit authentifiée (garantie) par un organisme reconnu. Ceci est assuré par l'utilisation des certificats.

Plus besoin de partager une même clé secrète, le chiffrement asymétrique permet de s'en dispenser. Mais il est malheureusement plus lent.

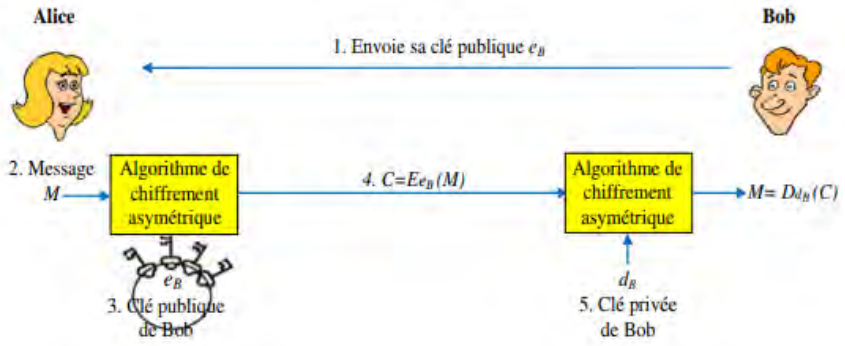


Figure 6 – Chiffrement à clé publique et déchiffrement à clé privée

Avec le chiffrement[W3] à clé publique, le problème de la gestion et d'échange des clés symétriques d'une façon sécurisée est résolu facilement.Par exemple Alice génére une clé symétrique aléatoire, elle la chiffre avec la clé publique de Bob et l'envoie à Bob. Même si ce message est intercepté, il ne peut pas être lu, car aucune personne à part Bob ne possède pas la bonne clé privée pour déchiffrer ce message. Bob déchiffre la clé symétrique avec sa clé privée. Maintenant les deux entités ont convenu sur une même clé secrète et peuvent commencer l'échange des messages en toute sécurité en utilisant le chiffrement symétrique. Par ailleurs, la mise en œuvre du chiffrement asymétrique permet également de réaliser la signature numérique de document.

Par exemple, Alice produit un document électronique et souhaite-le signer numériquement pour le protéger d'être modifié et prouver son authenticité. Pour cela, Alice calcule le condensé du document et le chiffre par sa clé privée d_A. Ensuite toute personne ayant accès à la clé publique d'Alice e_A peut ainsi vérifier la signature en déchiffrant avec e_A, en calculant, le condensé du document reçu et en comparant les deux valeurs obtenues. Si les valeurs sont identiques, la signature est validée, c'est à dire il est prouvé que le document n'a pas été modifié depuis qu'il a été signé, et c'est Alice qui l'a signé. Le chiffrement asymétrique est au moins 100 fois plus lent que le chiffrement symétrique et il nécessite des clés plus longues (taille courante de l'ordre de 1024 bits). Il est principalement utilisé pour chiffrer des messages courts, comme par exemple la clé secrète d'un algorithme de chiffrement symétrique.

Le principe de ce genre d'algorithme est qu'il s'agit d'une fonction unidirectionnelle à trappe. Une telle fonction a la particularité d'être facile à calculer dans un sens, mais difficile voire impossible dans le sens inverse. La seule manière de pouvoir réaliser le calcul inverse est de connaître une trappe. Une trappe peut par exemple être une faille dans le générateur de clés. Cette faille peut être soit accidentelle ou intentionnelle de la part du concepteur.

Cette fois, une clé secrète est déterminée par une des deux parties souhaitant communiquer et celle-ci est envoyée chiffrée par un chiffrement asymétrique. Une fois connue des deux parties, celles-ci communiquent en chiffrant symétriquement leurs échanges. Cette technique est notamment appliquée lorsque vous visitez un site dont l'adresse débute par « https ».

Nous allons[7] voir ici un processus de cryptographie à clé publique appelé RSA (d'après le nom de ses inventeurs Ronald Rivest, Adi Shamir et Leonard Adleman en 1977). Il est très utilisé dans le commerce électronique et pour échanger des données confidentielles sur Internet. Le chiffrement RSA est asymétrique. Il utilise deux clés (des nombres entiers) : une clé publique pour chiffrer et une clé privée pour déchiffrer les données. Une personne A (souvent nommée Alice) souhaite que B (souvent appelé Bob) lui envoie des données confidentielles. La personne A crée les deux clés, rend la clé publique accessible et conserve la clé privée. La clé publique est utilisée par son correspondant B pour chiffrer les données qui seront envoyées. La clé privée permettra à A de déchiffrer ces données. Ce procédé est souvent utilisé pour transmettre le mot-clé qui permettra alors de poursuivre l'échange de message par chiffrement cyclique comme vu ci-dessus :

• Bob envoie à Alice une clé de chiffrement symétrique (le mot-clé) qui peut ensuite être utilisée par Alice et Bob pour échanger des informations. Voyons ce processus plus en détails : supposons que B souhaite envoyer le mot-clé à A.

• A construit une clé privée et une clé publique. Elle envoie la clé publique

à B sans plus de précaution.

• B se sert de cette clé pour crypter le mot-clé. Il envoie le mot-clé crypté à A.

• Grâce à la clé privée, A parvient à décoder le mot-clé.

1.2.8 Certificats numériques

Pour qu'Alice[W5] puisse vérifier que la clé publique eB, qu'elle a obtenue, appartient vraiment à Bob, on doit utiliser une infrastructure à clé publique PKI (Public Key Infrastructure) et plus précisément le certificat numérique. Ce dernier permet d'associer une clé publique à une entité (une personne, une machine, ...) afin d'en assurer la validité. Le certificat est en quelque sorte la carte d'identité de la clé publique et est délivré par un organisme appelé autorité de certification (souvent noté CA pour Certification Authority). Le certificat est un document signé par une autorité de confiance et contenant des informations d'identification de l'entité (comme son non, son adresse e-mail, l'employeur, etc.) groupées avec la clé publique de cette entité. Ce certificat est délivré et signé numériquement par un tiers de confiance (TTP) indépendant appelé l'autorité de certification (CA). Mais ceci suppose que le CA a déjà vérifié (souvent physiquement) l'identité de l'entité en question. Rappelons que l'entité peut être une personne ou une organisation. Un certificat est essentiellement un véhicule pour le transport de clés publiques d'une façon vérifiable comme les certificats standard X.509 qui contiennent aussi le nom de l'émetteur du certificat (comme Verisign ou Thawte), des conditions de validité et d'autres attributs additionnels. Donc les certificats sont utilisés pour éviter les problèmes d'usurpation d'identité. Il y a des certificats personnels délivrés à des personnes et souvent appelés identifiants numériques, et d'autres délivrés à des organisations implémentés dans le serveur de l'organisation. Le certificat doit être installé dans l'ordinateur ou dans un équipement de communication. Parfois, la clé privée d'un utilisateur est compromise (par exemple volée), dans ce cas, le certificat doit être révoqué. Cela se fait souvent en incluant les certificats révoqués dans une liste des certificats révoqués (CRL(Certificate Revocation List)) qui est gérée et signée par le CA.

1.2.9 Efficacité des algorithmes de chiffrement

Un procédé[W2] est décrit par des algorithmes de cryptage et de décryptage, réalisés par des programmes. Pour être utilisable, un procédé de cryptage ne doit pas être trop élaborer : les algorithmes de codage et de décodage doivent être suffisamment rapide dont peu complexes, pour ne pas

ralentir excessivement leur exploitation. Inversement, pour garantir le secret du cryptage, il faut que celui-ci soit difficile. Cela suppose d'une autre part que le secret des clés soit bien gardé, d'autre part que les algorithmes de découverte des clés soit suffisamment complexe, donc suffisamment lents, pour décourager les recherches. La qualité d'un cryptage est donc essentiellement liée au temps d'utilisation (qui doit être court), et au temps de découverte, qui doit être long. Le choix du cryptage doit tenir compte des intentions et des moyens de l'adversaire, et des risques encourus. Le déchiffrage d'une donnée secrète donne accès à des informations confidentielles, la connaissance des clés de cryptage permet d'altérer des informations protégées ou d'en émettre de fausses (brouillage). La découverte d'un cryptage sera liée au temps disponible et à la puissance des ordinateurs utilisés par l'espion. La fréquence de chargement des clés doit évidement tenir compte de ces donnés. Les agressions ne sont pas toujours malveillantes, les donnés détruites ou divulguées sont le plus souvent le résultat d'erreur ou de négligence contre lesquelles il faut aussi se protéger. Les deux principaux procédés de cryptage connus sont les cryptages symétriques et asymétriques.

1.2.10 Fonctions de Hachage et Signature Numérique

Fonctions de Hachage Une fonction[W6] de hachage cryptographique fait partie d'un groupe de fonctions de hachage adapté aux applications cryptographiques telles que SSL /TLS. Comme les autres fonctions de hachage, les fonctions de hachage cryptographiques sont des algorithmes mathématiques à sens unique utilisés pour mapper des données de toute taille à une chaîne de bits de taille fixe. Les fonctions de hachage cryptographique sont largement utilisées dans les pratiques de sécurité de l'information, telles que les signatures numériques, les codes d'authentification de message et d'autres formes d'authentification.

Les fonctions de hachage cryptographique doivent avoir les propriétés suivantes :

1. Le même message donne toujours la même valeur de hachage (c'est à dire que la fonction est déterministe).

2. La valeur de hachage est calculée rapidement.

3. Il est impossible d'avoir deux messages avec la même valeur de hachage (appelée «collision»).

4. Il est impossible de créer intentionnellement un message qui produit une valeur de hachage donnée.

5. De légères modifications apportées au message doivent modifier considérablement la valeur de hachage résultante, afin qu'elle apparaisse non corrélée avec le hachage d'origine.

Les fonctions de hachage cryptographique les plus couramment utilisées incluent MD5, SHA-1 et SHA-2. L'unicité de chaque hachage est vitale pour l'intégrité de la fonction de hachage cryptographique. C'est ce qui distingue vraiment les fonctions de hachage cryptographiques des autres fonctions de hachage. Ils assurent qu'un message particulier est identifié de manière unique et est impossible à dupliquer.

Signature Numérique La signature[W4] numérique ou électronique (à ne pas confondre avec un certificat numérique), est une technique de validation mathématique de l'authenticité et de l'intégrité d'un message, d'un logiciel ou d'un document électronique. Equivalent électronique, par essence plus sécurisé, d'une signature manuscrite ou de l'apposition d'un sceau, la signature numérique est une solution à la falsification et l'usurpation d'identité dans les communications électroniques. Elle constitue un élément supplémentaire prouvant l'origine, l'identité et l'état d'un document électronique, d'une transaction ou d'un message et démontre le consentement éclairé du signataire. Dans de nombreux pays, notamment aux Etats-Unis, les signatures numériques ont la même valeur juridique que les documents signés classiques. L'imprimerie nationale américaine « United States Government Printing Office » publie des éditions électroniques des lois de droit privé, public et du budget et des projets de loi du Congrès portant sur signature numérique.

Fonctionnement des signatures numériques Les signatures[W6] numériques trouvent leur source dans le chiffrement à clé publique dit chiffrement asymétrique. Un algorithme à clé publique comme RSA permet de générer deux clés mathématiquement liées : une privée et une publique. Pour créer une signature numérique, le logiciel qui appose une signature, par exemple un logiciel de messagerie, crée un code de hachage unidirectionnel des données électroniques à signer. La clé privée sert ensuite à chiffrer le code de hachage. Ce code de hachage chiffré, avec d'autres informations comme l'algorithme de hachage, est la signature numérique. Le choix du chiffrement du hachage, plutôt que du message ou du document complet, s'explique par le fait qu'une fonction de hachage convertit n'importe quelle entrée en une valeur de longueur fixe, habituellement bien plus courte. C'est un gain de temps puisque le hachage est bien plus rapide que la signature. La valeur du code de hachage est propre aux données hachées. Toute modification apportée aux données, même si elle ne porte que sur un seul caractère, en changera la valeur. Cet attribut permet à d'autres de valider l'intégrité des données en utilisant la clé publique du signataire pour déchiffrer le code de hachage. Si le code déchiffré correspond à un second code de hachage des mêmes données, cela prouve que les données n'ont pas changé depuis la signature. Si les codes de hachage sont différents, l'alternative est que soit les données ont été falsifiées (intégrité), soit la signature a été créée à partir d'une clé privée qui

ne correspond pas à celle présentée par le signataire (authentification). N'importe quel message, chiffré ou non, peut contenir une signature numérique pour assurer le destinataire de l'identité de l'expéditeur et de l'intégrité du message. Le recours aux signatures numériques empêchera le signataire de nier avoir signé (non-répudiation), à moins que sa clé privée n'ait été compromise. En effet, la signature numérique est propre à la combinaison formée par ce document et le signataire, elle les lie l'un à l'autre. Un certificat numérique, document électronique signé numériquement par l'autorité émettrice, lie une clé publique à une identité. Il sert à vérifier qu'une clé publique appartient à une personne ou à une entité précise. La plupart des logiciels de messagerie modernes prennent en charge les signatures électroniques et les certificats numériques. Ils facilitent, par là, la signature des messages sortants et la validation des messages entrants signés électroniquement. Les signatures numériques servent aussi beaucoup à établir la preuve de l'authenticité, de l'intégrité des données et de la non-répudiation, des communications et des transactions effectuées sur internet. Si les deux valeurs de hachage sont identiques, le message n'a pas été falsifié et le destinataire sait qu'il provient bien de l'expéditeur.

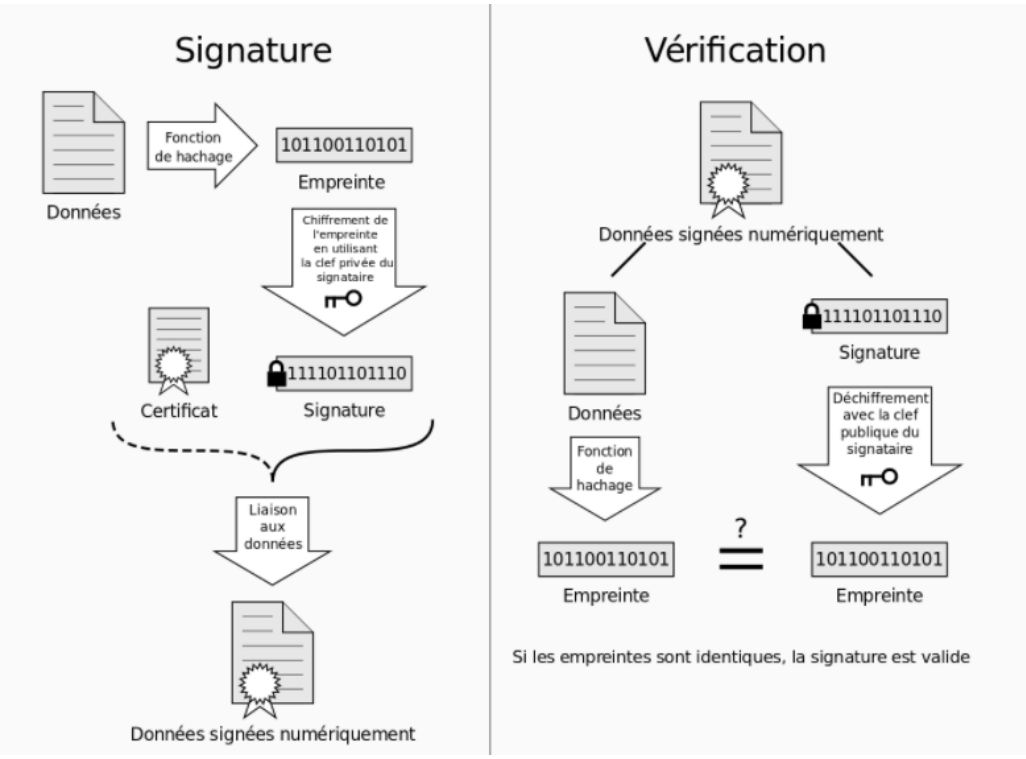


Figure 7 – Schéma de signature et de vérification

1.3 Notion de proxy

1.3.1 Fonction proxy de cryptage Unidirectionnel

La fonction proxy[2] de cryptage Unidirectionnel est définie comme un tuple $E = (UniGen, UniEnc, UniDec, PDec, FDec)$. L'algorithme de génération de clé UNIGEN génére des clés pour chaque utilisateur U. Ensuite, pour chaque utilisateur U, il génére deux clés supplémentaires pour le proxy P et l'utilisateur F. Les utilisateurs généraux chiffrent des méssages en clair utilisant l'algorithme UniEnc et pour décrypter ils utilisent l'algorithme UniDec. Chaque fois que l'utilisateur F veut décrypter un texte chiffré e, il demande l'aide du mandataire P. Le mandataire P utilise le PDec pour transformer le texte chiffré e en un texte chiffré e' et l'envoyé à l'utilisateur F. L'utilisateur F applique la fonction FDec au texte chiffré reçu e' et obtient le texte clair original m.

1.3.2 Fonction proxy de la signature Unidirectionnel

La fonction proxy[2] de la signature est définie comme un tuple S = (UniGen, UniSig, UniVer, PSig, FSig). Comme dans le cas du cryptage, l'algorithme de génération de clé génére pour chaque utilisateur général U.

Ensuite, pour chaque utilisateur U, il génére deux autres clés pour le proxy P et l'utilisateur F. Les utilisateurs généraux signent les messages en utilisant l'algorithme UniSig et les vérifient à l'aide de l'algorithme UniVer. A Chaque fois que l'utilisateur F veut signer un message m au nom d'un certain utilisateur U, il demande de l'aide au proxy P. Tout d'abord, l'utilisateur F utilise FSig pour générer une signature partielle du message m. Le mandataire P transforme la signature partielle en une autre signature en appliquant le PSig sur la signature partielle.

La notion de cryptographie par procuration peut être très utile dans les cas où un utilisateur doit effectuer des opérations sensibles (par exemple, décryptage de texte chiffré, génération de signature) sans détenir les clés secrètes. Par exemple, le président d'une société peut déléguer ses droits de signature en donnant une clé de procuration à son assistant. La clé de procuration transforme une signature créer par le vice-président en la signature du président, permettant ainsi à l'assistant de cosigner uniquement si le document a été signé au préalable par le vice-président.

1.3.3 Fonction proxy de cryptage bidirectionnel

La fonction[2] de cryptage est définie comme un tuple E = (BiGen, BiEnc, BiDec, Π). L'algorithme de génération de clés BiGen crée des clés pour

	Bidirectional	*Unidirectional*
Encryption	$U(sk_U)$ $P(\pi)$ $F(sk_F)$ $e=Enc_U(m) \longrightarrow e'=\pi(e) \longrightarrow e'=Enc_F(m)$	$U(DK_U)$ $P(DK_P)$ $F(DK_F)$ $e=Enc_U(m) \longrightarrow PDec(e) \longrightarrow FDec(PDec(e))=m$
Signature	$F(sk_F)$ $P(\pi)$ $U(sk_U)$ $s'=Sig_F(m) \longrightarrow s'=\pi(s) \longrightarrow s=Sig_U(m)$	$U(sk)$ $F(sk_F)$ $P(sk_P)$ $m \longrightarrow FSig(m) \longrightarrow s=Psig(FSig(m))$

Table 1 – Cryptage et de signature uni(bi)directionnel

tous les utilisateurs du système, y compris l'utilisateur F. Pour chaque paire de clés (kU, kF), l'algorithme BiGen génére une clé bidirectionnelle π. Les utilisateurs généraux U cryptent et décryptent les messages à l'aide de BiDec. Afin de décrypter un texte chiffré appartenant à un utilisateur général U, l'utilisateur F demande de l'aide au mandataire P. Le mandataire P utilise la fonction bidirectionnelle Π et la clé bidirectionnelle π pour transformer le texte chiffré pour l'utilisateur U en texte chiffré pour l'utilisateur F. Après cela, l'utilisateur F peut décrypter le nouveau texte chiffré avec sa propre clé et obtenir le message en clair.

1.3.4 Fonction proxy de la signature Bidirectionnel

La fonction[2] de signature est définie comme un tuple S = (BiGen, BiSig, BiVer, Π). Comme dans le cas du cryptage bidirectionnel, l'algorithme de génération BiGen crée des clés pour tous les utilisateurs dans le système, y compris l'utilisateur F. Pour chaque paire de clés (kU, kF), l'algorithme BiGen génère une clé bidirectionnelle π. Les utilisateurs généraux U signent les messages en utilisant BiSig et vérifient les signatures à l'aide de BiVer. Chaque fois que l'utilisateur F veut générer une signature valide pour un message m au nom d'un utilisateur U, il génère d'abord une signature avec sa propre clé et puis demande au mandataire P de l'aider. Le mandataire P utilise le système bidirectionnel Π et la clé bidirectionnelle π pour transformer la signature générée par l'utilisateur F en une autre signature avec la clé de l'utilisateur. Le tableau ci dessous reflète la façon dont les clés unidirectionnelles et bidirectionnelles fonctionnent à la fois pour le cryptage et les signatures.

Même si les proxy bidirectionnels et unidirectionnels de l'Union européenne atteignent le même objectif, il y a quelques différences entre eux. Tout d'abord, les fonctions de proxy unidirectionnel ne peuvent être utilisées

que d'une seule manière, d'un utilisateur à l'autre, le sens inverse nécessite une autre fonction. Les fonctions de proxy bidirectionnels peuvent être utilisé dans les deux sens. Les systèmes bidirectionnels suppose que l'organisme chargé de l'application de la loi (utilisateur F) a sa propre clé. Les schémas unidirectionnels ne font pas cette hypothèse mais payent un prix plus élevé pour les besoins de stockage car l'utilisateur F doit stocker une clé pour chaque utilisateur dans le système. Dans les deux cas, le problème d'espace se pose pour le proxy P car il faut garder une clé pour chaque utilisateur. Ce problème peut être très important dans les systèmes où le nombre d'utilisateurs généraux est extrêmement important. Une solution est offerte par les primitifs basés sur l'identité, où le mandataire ne doit conserver qu'une partie du secret. Dans les deux cas, la révocation est facilement obtenue pour faire en sorte que le tiers P refuse d'aider. Les schémas unidirectionnels et bidirectionnels exigent que le mandataire P assiste en permanence l'organisme chargé de l'application de la loi F lors du décryptage du texte chiffré ou de générer des signatures valables au nom d'un utilisateur.

1.4 Chiffrement et Signature Unidirectionnel

1.4.1 Fonction de Chiffrement Unidirectionnel

La présente[W1] invention se rapporte à un système et à un procédé cryptographique utilisant de nouvelles familles de fonctions de chiffrement unidirectionnelles.

Des fonctions de chiffrement unidirectionnelles sont simples à calculer mais difficiles à inverser, la « difficulté » faisant référence ici à la complexité moyenne de la tâche d'inversion.

À l'origine, la voie de communication protégée devait servir uniquement pour un chiffrement unidirectionnel (c'est-à-dire un chiffrement des données transmises de l'ordinateur de l'utilisateur aux serveurs d'un ministère).

Sensiblement toutes les primitives cryptographiques impliquent l'existence de fonctions de chiffrement unidirectionnelles, et beaucoup d'entre elles peuvent être construites sur la base de l'existence de fonctions de chiffrement unidirectionnelles ou sur la base de versions connexes de cette supposition.

La clé d'identification est générée à l'aide d'une séquence de nombre générée par l'identification de utilisateur de manière à sélectionner et à combiner **l'ensemble clé de diversification**. La clé aléatoire est générée sur la base du principe d'une fonction de chiffrement unidirectionnelle selon la clé publique commune . Des informations se présentant sous la forme d'un code, d'identification personnel contenant le nom d'un utilisateur autorisé, d'accès chiffré au moyen d'une fonction de chiffrement unidirectionnel, d'authenticité

formé par un code d'accès au système et au moins, d'identification personnel, sont enregistrés sur un disque souple.

Afin d'atteindre l'objectif visé, la présente invention se rapporte à un dispositif de traitement de données qui : génére N messages sur la base d'un uplet (F = (f1, ..., fm)) de polynômes multivariés. ces derniers étant définis sur un anneau (K) et un vecteur s (s appartenant à Kn) entre un document (M) et les N messages dans une fonction de chiffrement unidirectionnelle. Cette fonction sélectionne N éléments de premières données, génére N éléments de secondes données correspondant respectivement aux N éléments de premières données, et fournit une signature électronique contenant les N éléments de premières données et les N éléments de secondes données à un vérificateur qui connaît l'uplet susmentionné (F) de polynômes multivariés et un vecteur y ($y = (y1, ..., ym) = (f1(s), ..., fm(s))$).

Le dispositif comporte une première mémoire, une clé de chiffrement mémorisée dans la première mémoire et une fonction unidirectionnelle pour une application à la clé de chiffrement.

Elle consiste essentiellement à chiffrer l'identifiant utilisateur et/ou son mot de passe, et une indication de synchronisation, de préférence un intervalle de temps fixe, selon une fonction unidirectionnelle secrète, puis de transmettre le message chiffré, que l'on nomme "identifiant utilisateur dynamique", à "l'autorité locale" de l'utilisateur où celui-ci est enregistré.

En utilisation, la clé de chiffrement est récupérée dans la première mémoire avant son application à la fonction unidirectionnelle et le dispositif est configuré pour limiter, à un seuil prédéterminé, le nombre de fois que la clé de chiffrement peut être récupérée dans la mémoire non volatile.

1.4.2 Chiffrement Unidirectionnel

Nous présentons[2] d'abord une technique générique unidirectionnelle qui transforme un schéma de cryptage général $E = (Enc-Gen, Enc, Dec)$ en un chiffrement unidirectionnel schéma $E' = (UniGen, UniEnc, UniDec, PDec, FDec)$. L'algorithme de génération de clés UniGen génére deux paires de clés (EK1,DK1, EK2,DK2) en faisant tourner le Enc-Gen algorithme deux fois. L'utilisateur U conserve les deux clés, tandis que le proxy P et l'utilisateur F obtiennent respectivement (EK1, EK2,DK1) (EK1, EK2,DK2). Nous définissons DKP = DK1 et DKF = DK2. L'algorithme de cryptage UniEnc est équivalent à crypter le message avec les deux clés EK1 et EK2 : UniEnc(m) = Enc1(Enc2(m)) = e. L'unidirectionnel algorithme de décryptage UniDec décrypte le texte chiffré e en appliquant l'algorithme de décryptage original Dec deux fois m = Dec2(Dec1(e)). Le proxy P utilise la fonction PDec pour transformer le texte chiffré e en texte chiffré e' en décryptant une fois avec sa clé DKP =

DK1 e' = Dec1(e). L'utilisateur F utilise le FDec pour transformer le texte chiffré e' en un message m (ou invalide) en le décryptant une fois avec sa clé DKF = DK2 m = Dec2(e'). Le double cryptage peut également être défini en utilisant deux schémas de cryptage E1 = (Enc-Gen1, Enc1,Dec1), E2 = (Enc-Gen2, Enc2,Dec2). Dans ce cas, la fonction unidirectionnelle schéma de cryptage E' = (UniGen, UniEnc, UniDec, PDec, FDec) est défini comme :

- UniGen(1^k) = (Enc-GenE1(1^k), Enc-GenE2 (1^k))
- UniEnc(m) = EncE1(EncE2(m))
- UniDec(e) = DecE2(DecE1 (e))
- PDec(e) = DecE1 (e)
- FDec(e') = DecE2(e')

Par souci de simplicité, nous supposons que les deux systèmes de cryptage sont identiques. Ensuite, nous prouvons que le système de cryptage est sécurisé selon nos définitions. Nous partons du principe que le système de cryptage initial est CCA2 et montre que le nouveau système de cryptage est également le CCA2. Si E est sécurisé par CCA2, alors E' est également CCA2 sécurisé contre le proxy P, l'utilisateur F, et tout utilisateur U. Le système générique est deux fois plus lent que le système original. Afin d'éliminer ce ralentissement, ils ont développé un cryptage unidirectionnel basé sur RSA.

Théoréme

Considérons un schéma de chiffrement standard E = (Enc-Gen, Enc,Dec). Sur la base de E, nous construisons un schéma de chiffrement unidirectionnel $E' = (UniGen, UniEnc, UniDec, PDec, FDec)$. Si E est sécurisé par CCA2 alors E' est également sécurisé CCA2 contre le proxy P, l'utilisateur F, et tous les utilisateurs U.

Preuve

1. Supposons[2] que E' n'est pas sécurisé par CCA2 contre le mandataire P. Cela signifie que :

• Le mandataire P ne peut pas utiliser son système de sécurité.

• Le proxy P peut casser E'

avec une probabilité de succès supérieure à 1/2. Nous supposons que PDec est un algorithme déterministe et que le proxy P ne soumet jamais PDec(UniEncEK(mb)) à l'oracle FDec. Sur la base de P, nous construisons un adversaire B capable de casser le schéma de chiffrement original E.

• L'adversaire B reçoit en entrée la clé publique EK2 du schéma de chiffrement original E.

• L'adversaire B simule les conditions nécessaires pour que le mandataire P puisse casser le cryptage unidirectionnel E' en choisissant de manière aléatoire une paire de clés publiques/privées $(EK1, DK1)$ et en la transmettant au proxy P avec EK2.

• L'adversaire B commence à faire fonctionner le proxy P.

• Chaque fois que le proxy P a besoin de faire une requête à l'oracle FDec, B simule l'oracle FDec en prenant la paire de clés EK1 et DK1.

• L'oracle FDec en prenant la requête q du proxy P la transmet à l'oracle Dec.

• B envoie au proxy P la sortie de l'oracle Dec. A un moment, P conteste le schéma de chiffrement unidirectionnel E' en choisissant deux messages $(m0, m1)$ et en les envoyant à B.

• B choisit les mêmes deux messages pour contester le schéma de chiffrement original E.

• Quand B est présenté avec le défi Enc2(mb), où m est choisi au hasard parmi les deux messages (m0,m1), B applique Enc1(Enc2(m)) et envoie le défi comme UniEnc(mb) au proxy P. Nous avons supposé que le proxy P peut casser le schéma de cryptage unidirectionnel avec une probabilité supérieure à 1/2. Ainsi, B peut casser le schéma de chiffrement standard avec une probabilité supérieure à 1/2.

2. Supposons que E' n'est pas CCA2 securisé contre l'utilisateur F. Cela signifie que :

• L'utilisateur F peut casser E' avec une probabilité de succès supérieure à 1/2. En se basant sur l'utilisateur F, nous construisons un adversaire B capable de briser le schéma de cryptage original E.

• L'adversaire B reçoit en entrée la clé publique EK1 du schéma de chiffrement original E.

• Tout d'abord, B simule les conditions nécessaires à l'utilisateur F pour casser le cryptage unidirectionnel E' en choisissant au hasard une paire de clés publique/privée (EK2,DK2) et en la transmettant à l'utilisateur F avec la clé EK1.

• L'adversaire B commence à exécuter la requête de l'utilisateur F.

• Lorsque l'utilisateur F effectue une requête q = Enc1(Enc2(m)) à l'oracle PDec, l'adversaire B prend la requête e, la transmet à l'oracle PDec et PDec la transmet à son propre oracle Dec et envoie directement la réponse Enc2(m) à F.

• Lorsque l'utilisateur F défie l'adversaire B, il choisit deux messages (m0,m1) et les envoie à B.

• B chiffre ces deux messages en utilisant la clé messages à l'aide de la clé EK2 et les envoie à l'adversaire.

• Quand on présente à B le défi Enc1(mb), où mb est choisi au hasard parmi deux messages (Enc2(m0), Enc2(m1)), B envoie le défi à F.

Nous avons supposé que l'utilisateur F peut casser le schéma de cryptage unidirectionnel avec une probabilité supérieure à 1/2. Ainsi, B peut casser le schéma de chiffrement standard avec une probabilité supérieure à 1/2.

Generic	$\mathsf{Succ}_{\mathsf{P},\mathcal{S}'} \stackrel{\text{def}}{=} \Pr\left[\mathsf{UniVer}(m,s) = succeed \;\middle|\; \begin{array}{c}(\mathsf{VK}_1,\mathsf{SK}_1,\mathsf{VK}_2,\mathsf{SK}_2) \leftarrow \mathsf{UniGen}(1^k)\\ (m,s) \leftarrow \mathsf{P}^{\mathsf{FSig}}(\mathsf{VK}_1,\mathsf{VK}_2,\mathsf{SK}_1)\end{array}\right]$ $\mathsf{Succ}_{\mathsf{F},\mathcal{S}'} \stackrel{\text{def}}{=} \Pr\left[\mathsf{UniVer}(m,s) = succeed \;\middle|\; \begin{array}{c}(\mathsf{VK}_1,\mathsf{SK}_1,\mathsf{VK}_2,\mathsf{SK}_2) \leftarrow \mathsf{UniGen}(1^k)\\ (m,s) \leftarrow \mathsf{F}^{\mathsf{FSig}}(\mathsf{VK}_1,\mathsf{VK}_2,\mathsf{SK}_2)\end{array}\right]$
RSA-Hash	$\mathsf{Succ}_{\mathsf{P},\mathcal{S}} \stackrel{\text{def}}{=} \Pr\left[\mathsf{UniVer}(m,s) = succeed \;\middle|\; (\mathsf{SK},\mathsf{VK}) \leftarrow \mathsf{UniGen}(1^k), (m,s) \leftarrow \mathsf{P}^{\mathsf{FSig}}(\mathsf{SK}_\mathsf{P},\mathsf{VK})\right]$ $\mathsf{Succ}_{\mathsf{F},\mathcal{S}} \stackrel{\text{def}}{=} \Pr\left[\mathsf{UniVer}(m,s) = succeed \;\middle|\; (\mathsf{SK},\mathsf{VK}) \leftarrow \mathsf{UniGen}(1^k), (m,s) \leftarrow \mathsf{F}^{\mathsf{PSig}}(\mathsf{SK}_\mathsf{F},\mathsf{VK})\right]$

Table 2 – Chiffrement unidirectionnel avec le schéma générique et RSA-hash

1.4.3 Signature Unidirectionnel

Soit S = (Sig-Gen, Sig, Ver) une norme du système de signature. La nouvelle signature générique unidirectionnelle[2] est S' = (UniGen, UniSig, UniVer, PSig, FSig). L'algorithme de génération UniGen génére deux paires de clés (SK1, VK1, SK2, VK2) pour chaque utilisateur U. La clé secrète du système de signature unidirectionnelle est SK = (SK1, SK2). La procuration P obtient (VK1, VK2, SKP = SK1) et l'utilisateur F obtient (VK1, VK2, SKF = SK2). La signature UniSig génére une signature valide pour un message m appartenant à M en apposant deux fois la signature Sig : s = (s1, s2) = Sig1(m)Sig2(m). De même, l'algorithme UniVer vérifie si les signatures générées par UniSig sont valables en appliquant la vérification originale de l'algorithme Ver deux fois Ver1(s1)Ver2(s2). Le proxy P utilise PSig pour générer une partie du signature unidirectionnelle Sig1(m), tandis que l'utilisateur F utilise FSig = Sig2(m) pour générer la signature entière avec PSig. Selon nos définitions, le terme système de signature générique unidirectionnel est sûr si le théorème suivant s'est avéré être vrai.

Théorème Soit S = (Sig-Gen, Sig, Ver) une norme du système de signature. Considérons S' = (UniGen, UniSig, UniVer, PSig, FSig) un système de signature unidirectionnel construit comme décrit ci-dessus, sur la base de S. Si S est UFCMA alors S' est UF-CMA contre le proxy P, l'utilisateur F, et tout utilisateur U. Le système de signature générique unidirectionnel comporte deux principaux inconvénients en termes de performances.

Premièrement, la taille du secret et l'augmentation du nombre de clés. Chaque utilisateur n'a plus une, mais deux clés.

Deuxièmement, le nombre d'opérations effectuées lorsque la signature et la vérification doublent.

Afin d'améliorer ils ont développé une signature unidirectionnelle efficace basé sur le RSA-Hash.

Preuve

1. Supposons que S' n'est pas UF-CMA contre le proxy P. Cela signifie que |SuccP,S' (1^k)| n'est pas négligeable. Nous supposons que le proxy P n'est pas autorisé à demander à l'oracle FSig FSig(m). Sur la base de S' nous construisons un faussaire B capable de briser le schéma de signature original S. • Le faussaire B reçoit une copie de l'oracle FSig.

• Le faussaire B reçoit en entrée la clé publique VK2 et essaie de générer une signature valide d'un message m sous la clé secrète SK2.

• Le faussaire B choisit au hasard une paire de clés publiques/privées (VK1, SK1) et la transmet à P avec VK2.

• Le faussaire B commence à faire fonctionner le proxy P.

• Lorsque P effectue une requête sur l'oracle de hachage pour un message m', le faussaire B fait suivre la requête à son propre oracle de hachage et envoie la réponse au proxy P.

• Lorsque P demande à l'oracle FSig de produire une signature pour un message m', B demande à son propre oracle de signature de produire une signature pour m' sous SK2 et envoie le résultat au proxy P.

À un moment donné, le proxy P génére une signature unidirectionnelle valide pour un message m avec une probabilité non-négligeable, où m est un message complètement nouveau, B prend la signature unidirectionnelle UniSig(m) = Sig1(m)Sig2(m), en supprimant la première partie et produit Sig2(m) comme une signature valide de m.

2. Supposons que S' n'est pas UF-CMA contre F. Cela signifie que |SuccP, S' (1^k)| n'est pas négligeable. Nous supposons que F n'est pas autorisé à demander à l'oracle PSig au sujet de m. En se basant sur S', nous construisons un faussaire B capable de briser le schéma de signature original S.

• Le faussaire B reçoit en entrée la clé publique VK1 et essaie de générer une signature valide d'un message m sous la clé secrète SK1.

• Le faussaire B choisit au hasard une paire de clés publique/privée (VK2, SK1) et la transmet à l'utilisateur F avec VK1.

• Le faussaire B commence à exécuter F.

• Lorsque l'utilisateur F fait une requête sur l'oracle de hachage pour un message m', le faussaire B fait suivre la requête à son propre oracle de hachage et renvoie la réponse à F.

• Lorsque F demande à l'oracle PSig de produire une partie de la signature pour un message m', B demande à son propre oracle de signature de produire une signature pour m' sous SK1.

• Après cela, B envoie à F Sig1(m').

• A un moment donné, F génére une signature unidirectionnelle valide pour un message m avec une probabilité non-négligeable, où m est un message complètement nouveau.

• B prend la signature unidirectionnelle UniSig(m) = Sig1(m)Sig2(m), supprime la deuxième partie et sort Sig1(m). comme une signature valide de m.

1.4.4 Primitives de cryptage unidirectionnel

Un système de cryptage[2] unidirectionnel comprend cinq algorithmes, E = (UniGen, UniEnc, UniDec, PDec, FDec). L'algorithme de génération UniGen(1^k) produit un tuple de (EK,DK) pour chaque utilisateur général U. EK est la clé de cryptage et DK est la clé de décryptage. Pour chaque clé secrète DK, l'algorithme de génération de clés crée deux clés secrètes DKP et DKF pour le mandataire P, et respectivement l'utilisateur F. Le tableau 2 montrent que l'algorithme de génération de clé ne produit que des clés d'utilisateur,même s'il construit également les clés du mandataire P et de l'utilisateur F. UniEncEK est l'algorithme de cryptage. Il crypte un message m ∈ M (par ex,$\{0,1\}^k$) comme e = UniEncEK(m). L'algorithme de décryptage UniDecDK est un algorithme déterministe qui prend le cryptogramme e, la clé secrète DK et produit m ∈ M (ou invalide dans le cas où e était un cryptogramme inapproprié). La propriété d'exactitude du chiffrement indique que UniDec(UniEnc(m)) = m, pour tout message m et paire de clés (EK,DK). La fonction PDec utilise la clé secréte du proxy P, DKP, pour transformer un texte chiffré e en texte chiffré e'. La fonction FDec prend ce texte chiffré e' et la clé secrète DKF de l'utilisateur F et produit le message original m ∈ M ou invalide si le texte chiffré e' n'était pas correct. La propriété d'exactitude précise que FDec(PDec(UniEnc(m))) = m, pour tout m et (EK,DK).

Officieusement, un schéma de cryptage unidirectionnel est considéré d'être en sécurité si aucune des entités participantes (utilisateur F, mandataire P, utilisateur U) ne peuvent le briser même s'ils détiennent des secrets supplémentaires. Par souci de simplicité, les définitions présentées dans le tableau2 seront spécifiques à la sécurité CCA2 pour la clé publique de cryptage. Dans nos définitions, l'agent mandataire P ne reçoit que l'accès oracle au FDec. En fait, P n'a pas besoin d'accéder à UniDec car il peut le simuler par lui-même. Les seules conditions nécessaire sont que PDec soit un algorithme déterministe et le proxy P ne soumet jamais PDec (UniEncEK (mb)) à l'oracle FDec. Pour chacun des schémas unidirectionnels décrits ci-dessous, nous prouverons qu'ils sont sécurisés selon les trois définitions.

Soit E = (UniGen, UniEnc, UniDec, PDec, FDec) un schéma de chiffrement unidirectionnel.

1. E est CCA2 protégé contre la procuration P si | $Succ_{P,E}$ (1^k)-1/2 | est négligeable. $Succ_{P,E}$ est défini comme suit, PDec est un algorithme

déterministe et le proxy P ne soumet jamais PDec(UniEncEK(mb)) à l'oracle FDec.

$$\mathsf{Succ}_{\mathsf{P},\mathcal{E}} \stackrel{\text{def}}{=} \Pr\left[b = \tilde{b} \;\middle|\; \begin{array}{c} (\mathsf{EK},\mathsf{DK}) \leftarrow \mathsf{UniGen}(1^k), (m_0, m_1) \leftarrow \mathsf{P}^{\mathsf{FDec}}(\mathsf{EK}, \mathsf{DK}_{\mathsf{P}}), \\ b \leftarrow \{0,1\}, \tilde{b} \leftarrow \mathsf{P}^{\mathsf{FDec}}(\mathsf{EK}, \mathsf{DK}_{\mathsf{P}}, \mathsf{UniEnc}_{\mathsf{EK}}(m_b)) \end{array} \right]$$

2. E est CCA2 sécurisé contre l'utilisateur F si | $Succ_{F,E}$ (1^k)-1/2 | est négligeable. $Succ_{F,E}$ est défini comme suit, l'utilisateur F ne peut pas soumettre le défi UniEncEK(mb) à l'oracle PDec.

$$\mathsf{Succ}_{\mathsf{F},\mathcal{E}} \stackrel{\text{def}}{=} \Pr\left[b = \tilde{b} \;\middle|\; \begin{array}{c} (\mathsf{EK},\mathsf{DK}) \leftarrow \mathsf{UniGen}(1^k), (m_0, m_1) \leftarrow \mathsf{F}^{\mathsf{PDec}}(\mathsf{EK}, \mathsf{DK}_{\mathsf{F}}), \\ b \leftarrow \{0,1\}, \tilde{b} \leftarrow \mathsf{F}^{\mathsf{PDec}}(\mathsf{EK}, \mathsf{DK}_{\mathsf{F}}, \mathsf{UniEnc}_{\mathsf{EK}}(m_b)) \end{array} \right]$$

3. E est CCA2 sécurisé contre tout utilisateur U si | $Succ_{U,E}$ (1^k) - 1/2 | est négligeable. $Succ_{U,E}$ est défini comme suit, l'utilisateur U ne peut pas soumettre le défi UniEncEK(mb) à l'oracle de décryptage UniDec.

$$\mathsf{Succ}_{\mathsf{U},\mathcal{E}} \stackrel{\text{def}}{=} \Pr\left[b = \tilde{b} \;\middle|\; \begin{array}{c} (\mathsf{EK},\mathsf{DK}) \leftarrow \mathsf{UniGen}(1^k), (m_0, m_1) \leftarrow \mathsf{U}^{\mathsf{UniDec}}(\mathsf{EK}), \\ b \leftarrow \{0,1\}, \tilde{b} \leftarrow \mathsf{U}^{\mathsf{UniDec}}(\mathsf{EK}, \mathsf{UniEnc}_{\mathsf{EK}}(m_b)) \end{array} \right]$$

1.4.5 Primitives de signature unidirectionnelle

Un système[2] de signature unidirectionnelle comprend cinq algorithmes : $S = (UniGen, UniSig, UniVer, PSig, FSig)$.L'algorithme de génération UniGen(1k) produit un tuple de clés (SK,VK) pour chaque utilisateur U. VK est la vérification et SK est la clé de signature. SK est utilisé pour générer les clés SKP et SKF données au mandataire P, à l'utilisateur F respectivement. L'algorithme de signature UniSig signe un message m ∈ M (par exemple $\{1,0\}^k$), $s = UniSigSK(m)$ en utilisant la clé secrète SK. La signature est formée par le tuple (m, s). L'algorithme de vérification UniVer utilise la clé public pour vérifier la validité d'une signature (m, s). La vérification de l'algorithme réussit si la signature est correcte et échoue autrement. La propriété d'exactitude exige que UniVer $(UniSig(m)) = réussir$. Le proxy P utilise la fonction PSig pour générer une signature partielle d'un message m ∈ M basé sur la SKP. L'utilisateur F utilise FSig pour générer une signature partielle d'un message m ∈ M basé sur SKF. PSig(FSig(m)) forme une signature complète du message m. Nous définissons le schéma de signature unidirectionnelle comme étant sûr si aucune entité (proxy P, utilisateur F, utilisateur U) ne peut générer seule des signatures valables sous la clé SK, même s'ils connaissent la SKP ou SKF et toutes les informations publiques disponibles.

Soit S = (UniGen, UniSig, UniVer, PSig, FSig) est un schéma de signature unidirectionnel.

1. S est UF-CMA par rapport à la procuration P si $|Succ_{P,S}\ (1^k)|$ est négligeable. $Succ_{P,S}$ est défini comme suit, la procuration P n'est pas autorisée à demander l'oracle FSig pour UniSig(m).

$$\mathsf{Succ}_{\mathsf{P},\mathcal{S}} \stackrel{\text{def}}{=} \Pr\left[\mathsf{UniVer}(m,s) = succeed \;\middle|\; (\mathsf{SK},\mathsf{VK}) \leftarrow \mathsf{UniGen}(1^k), (m,s) \leftarrow \mathsf{P}^{\mathsf{FSig}}(\mathsf{SK}_\mathsf{P},\mathsf{VK})\right]$$

2. S est UF-CMA contre l'utilisateur F si $|Succ_{F,S}\ (1^k)|$ est négligeable. $Succ_{F,S}$ est défini comme suit, l'utilisateur F est n'est pas autorisé à demander l'oracle de signature pour UniSig(m).

$$\mathsf{Succ}_{\mathsf{F},\mathcal{S}} \stackrel{\text{def}}{=} \Pr\left[\mathsf{UniVer}(m,s) = succeed \;\middle|\; (\mathsf{SK},\mathsf{VK}) \leftarrow \mathsf{UniGen}(1^k), (m,s) \leftarrow \mathsf{F}^{\mathsf{UniSig}}(\mathsf{SK}_\mathsf{F},\mathsf{VK})\right]$$

3. S est UF-CMA contre tout utilisateur U si $|Succ_{U,S}\ (1^k)|$ est négligeable pour tout adversaire U. $Succ_{U,S}$ est défini comme ci-dessous, l'utilisateur U n'est pas autorisé à demander l'oracle de signature pour UniSig(m).

$$\mathsf{Succ}_{\mathsf{U},\mathcal{S}} \stackrel{\text{def}}{=} \Pr\left[\mathsf{UniVer}(m,s) = succeed \;\middle|\; (\mathsf{SK},\mathsf{VK}) \leftarrow \mathsf{UniGen}(1^k), (m,s) \leftarrow \mathsf{U}^{\mathsf{UniSig}}(\mathsf{VK})\right]$$

1.5 Chiffrement et Signature Bidirectionnel

1.5.1 Chiffrement Bidirectionnel

Dans cette[2] section, nous présentons une mise en œuvre générique d'un système de cryptage bidirectionnel basé sur un cryptage standard de l'UE. Supposons que nous disposions d'un schéma de cryptage E = (Enc-Gen, Enc,Dec). Nous transformons E dans un schéma de cryptage bidirectionnel E' = (BiGen, BiEnc, BiDec, Π) en suivant les prochaines étapes. Pour chaque Utilisateur U, l'algorithme de génération BiGen exécute l' algorithme de génération Enc-Gen pour générer trois paires de clés (k1, k2, k3), où chaque Ki = (EKi,DKi). Les utilisateurs U, P et F reçoivent chacun deux paires de clés de telle sorte que deux entités quelconques n'ont qu'une seule paire de clés en commun. Par exemple, les clés de l'utilisateur U sont EKU = (EK1, EK2), DKU = (DK1,DK2), les clés du proxy P sont EKP = (EK2, EK3), DKP = (DK2,DK3), et les clés de l'utilisateur F sont EKF = (EK1, EK3), DKF = (DK1,DK3). Dans le contexte de cryptage bidirectionnel, nous disons que les deux utilisateurs U et F ont les clés privées, tandis que le mandataire P a la clé bidirectionnelle π. L'algorithme de cryptage BiEnc effectue un double cryptage e = BiEncU(m) = Enc1(Enc2(m)). De même, l'algorithme de décryptage π est défini comme un double décryptage m = BiDecU(e) =

Dec2(Dec1(e)). La fonction proxy Π transforme le texte chiffré crypté avec la clé de l'utilisateur U en texte chiffré crypté avec la clé de l'utilisateur F. La première étape consiste à décrypter le texte chiffré $e = BiEncU(m) = Enc1(Enc2(m))$ en exécutant $e' = \text{Dec1(e)}$. Ensuite, il crypte e' avec l'autre moitié de la clé et obtient $e'' = \text{Enc3}(e')$. Le résultat est $e'' = \text{BiEncF(m)}$. Le schéma de chiffrement bidirectionnel générique décrit ci-dessus est sécurisé si aucun adversaire (proxy P, utilisateur F, utilisateur U) n'est capable de le briser. Supposons que le cryptage initial est sécurisé par le système CCA2. Nous montrerons que dans ce cas, le cryptage bidirectionnel est également sécurisé par le CCA2. La preuve se trouve dans[2] l'annexe C.1.

Théoréme

Considérons un schéma de chiffrement standard E = (Enc-Gen, Enc,Dec). En se basant sur E, nous construisons un schéma de de chiffrement unidirectionnel E' = (UniGen, UniEnc, UniDec, PDec, FDec). Si E est sécurisé par CCA2, alors E' est aussi sécurisé par CCA2 contre le mandataire P, l'utilisateur F, et tout utilisateur U.

1.5.2 Signature Bidirectionnel

Tout d'abord, considérons[2] un schéma de signature standard S = (Sig-Gen, Sig, Ver). Le paragraphe suivant explique comment construire un système de signature générique bidirectionnel S' = (BiGen, BiSig, BiVer,Π) du régime initial S. L'algorithme de génération de clés BiGen utilise le schéma original, l'algorithme de génération de clés Sig-Gen, pour générer trois clés (k1, k2, k3) où ki = (SKi, VKi), et les donne à aux utilisateurs U, F et P, de sorte qu'ils aient en commun une seule paire de clé. Par exemple, l'utilisateur U obtient VKU = (VK1, VK2), SKU = (SK1, SK2), l'utilisateur F obtient VKF = (VK1, VK3), SKF = (SK1, SK3) et le mandataire P obtient VKP = (VK2, VK3), SKP = (SK2, SK3). L'algorithme de signature BiSig calcule la signature du message m ∈ M en appliquant l'algorithme de signature standard deux fois, une fois pour chaque clé. Par exemple, l'utilisateur U signe un message (s1, s2) = Sig1(m)Sig2(m). BiVer vérifie si une signature généré avec BiSig est correct, en exécutant la norme algorithme de vérification deux fois Ver1(s1) et Ver2(s2). La fonction de procuration transforme une signature valide générée par BiSig pour une paire de clés dans une signature valide générée avec une autre paire de clés : Π(BiSigU(m)) = BiSigF(m). Le schéma de signature bidirectionnelle générique est sécurisé si le prochain théorème est vrai. La preuves se trouvent dans[2] à l'annexe D.1.

Théorème

Considérons un schéma de signature standard $S = (Sig-Gen, Sig, Ver)$. A partir de S, nous construisons un schéma de signature bidirectionnel S' =

(BiGen, BiSig, BiVer, Π). Si S est UF-CMA,' alors S' est UF-CMA contre le mandataire P, l'utilisateur F, et tous autres utilisateurs.

1.5.3 Primitives de cryptage bidirectionnel

Un système[2] de cryptage bidirectionnel comprend quatre algorithmes : E = (BiGen, BiEnc, BiDec,Π). L'algorithme de génération de clés BiGen produit une paire de clés (EKU,DKU) pour chaque utilisateur U. En outre, il génére des clés pour l'utilisateur F, (EKF,DKF). Ensuite, il crée une clé bidirectionnelle π pour chaque utilisateur U. Les clés bidirectionnelles sont données au proxy P. L'algorithme de cryptage BiEncEK prend comme entrée un message crypté et une clé publique EK et produit le texte chiffré e = BiEncEK(m). BiDecDK est l'algorithme de décryptage déterministe qui prend le texte chiffré e, la clé secrète DK correspondante à la clé publique EK pour retrouver le message m ∈ M (ou invalide dans le cas où e était un cryptogramme inapproprié) . La propriété d'exactitude du cryptage indique que BiDec(BiEnc(m)) = m, pour tout m et (EK,DK). Π est la fonction bidirectionnelle qui transforme des cryptogrammes avec une clé (EKU) en cryptogrammes avec une autre clé (EKF). Nous pouvons dire que les schémas de cryptage bidirectionnels sont sécurisés si ni le proxy P ni les utilisateurs (U, F) ne peuvent attaquer le système. Pour des raisons techniques, nous supposons qu'il existe un algorithme qui évalue la relation R_π(e, e') par rapport au vrai ou faux, où $e = BiEnc(m)$ est le cryptogramme original et $e' = \pi$(e) est le cryptogramme modifié calculé par le proxy P. La sortie de l'algorithme est vraie, si DecEKU(e) = DecEKF(e'). Ayant un tel algorithme, nous permettons au mandataire P d'avoir accès à l'oracle du BiDecF car il peut simuler l'accès de l'oracle à BiDecU par lui-même. En outre, nous limitons l'accès du mandataire P au BiDecF en bloquant toute soumissions à l'oracle d'un texte chiffré e' tel que R_π(e, e') = vrai.

Soit E = (BiGen, BiEnc, BiDec, Π) un schéma de chiffrement bidirectionnel.

1. E est CCA2 sécurisé contre le proxy P si $|Succ_{P,E}\ (1^k)$ - 1/2| est négligeable, $Succ_{P,E}$ est défini comme ci-dessous, et BiEnc(mb) n'est jamais soumis à l'oracle de décryptage BiDec.

$$\mathsf{Succ}_{\mathsf{P},\mathcal{E}} \stackrel{\text{def}}{=} \Pr\left[b = \tilde{b} \;\middle|\; \begin{array}{c} (\mathsf{EK_U}, \mathsf{DK_U}, \mathsf{EK_F}, \mathsf{DK_F}, \pi) \leftarrow \mathsf{BiGen}(1^k), (m_0, m_1) \leftarrow \mathsf{P}^{\mathsf{BiDec}}(\mathsf{EK_U}, \mathsf{EK_F}, \pi), \\ b \leftarrow \{0,1\}, \tilde{b} \leftarrow \mathsf{P}^{\mathsf{BiDec}}(\mathsf{EK_U}, \mathsf{EK_F}, \pi, \mathsf{BiEnc}(m_b)) \end{array} \right]$$

2. E est CCA2 sécurisé contre l'utilisateur F si $|Succ_{F,E}\ (1^k)$ - 1/2| est négligeable, $Succ_{F,E}$ est défini comme ci-dessous, et BiEnc(mb) n'a jamais été soumis à l'oracle P.

$$\mathsf{Succ}_{\mathsf{F},\mathcal{E}} \stackrel{\text{def}}{=} \Pr\left[b = \tilde{b} \;\middle|\; \begin{array}{c} (\mathsf{EK_U}, \mathsf{DK_U}, \mathsf{EK_F}, \mathsf{DK_F}, \pi) \leftarrow \mathsf{BiGen}(1^k), (m_0, m_1) \leftarrow \mathsf{F}^{\Pi}(\mathsf{EK_U}, \mathsf{EK_F}, \mathsf{DK_F}), \\ b \leftarrow \{0,1\}, \tilde{b} \leftarrow \mathsf{F}^{\Pi}(\mathsf{EK_U}, \mathsf{EK_F}, \mathsf{DK_F}, \mathsf{BiEnc}(m_b)) \end{array} \right]$$

3. E est CCA2 sécurisé contre tout utilisateur U si $|Succ_{U,E}\ (1^k)$ - $1/2|$ est négligeable, pour tout adversaire A, nous définissons $Succ_{U,E}$ comme ci-dessous, et BiEnc(mb) n'a jamais été soumis à l'oracle de décryptage BiDec.

$$\mathsf{Succ}_{\mathcal{A},\mathcal{E}} \stackrel{\text{def}}{=} \Pr\left[b = \tilde{b} \;\middle|\; \begin{array}{c} (\mathsf{EK_U}, \mathsf{DK_U}, \mathsf{EK_F}, \mathsf{DK_F}, \pi) \leftarrow \mathsf{BiGen}(1^k), (m_0, m_1) \leftarrow \mathcal{A}^{\mathsf{BiDec}}(\mathsf{EK_U}, \mathsf{EK_F}), \\ b \leftarrow \{0,1\}, \tilde{b} \leftarrow \mathcal{A}^{\mathsf{BiDec}}(\mathsf{EK_U}, \mathsf{EK_F}, \mathsf{BiEnc}(m_b)) \end{array} \right]$$

1.5.4 Primitives de signature bidirectionnelle

Un schéma[2] de signature bidirectionnelle comprend quatre algorithmes S = (BiGen, BiSig, BiVer, Π). L'algorithme de génération de clés BiGen(1^k), où k est le paramètre de sécurité, génère des clés pour tous les utilisateurs, y compris l'utilisateur F. Par exemple, l'utilisateur U obtient les clés (SKU, VKU) et l'utilisateur F obtient (SKF, VKF). L'algorithme de génération calcule également une clé bidirectionnelle π pour chaque utilisateur et la donne au proxy P. L'algorithme de signature BiSig signe un message m ∈ M (par exemple $\{1,0\}^k$), s = BiSigSK(m) en utilisant une clé secrète SK. La signature est formée par le tuple (m, s). La signature (m, s) est vérifiée par l'algorithme de vérification BiVer. L'algorithme de vérification algorithme de vérification produit un résultat positif si la signature est correcte et un échec dans le cas contraire. La propriété d'exactitude exige que BiVer(BiSig(m)) = succès. La fonction proxy utilise la clé bidirectionnelle pour transformer une signature générée avec une clé secrète en une signature générée avec une autre clé secrète. Un schéma de chiffrement bidirectionnel défini comme ci-dessus est considéré comme sûr s'il ne peut être attaqué avec succès par aucun utilisateur quelconque (U, F) ou par un tiers (P). Les définitions formelles sont présentées ci-dessous.

Soit S = (BiGen, BiSig, BiVer,) un système de signature bidirectionnel.

1. S est UF contre le mandataire P si $|Succ_{P,S}(1^k)|$ est négligeable, où $Succ_{P,S}$ est défini comme ci-dessous et P n'est pas autorisé à demander BiSig(m) à l'oracle de signature.

$$\mathsf{Succ}_{\mathsf{P},\mathcal{S}} \stackrel{\text{def}}{=} \Pr\left[\mathsf{BiVer}(m,s) = succeed \;\middle|\; \begin{array}{c} (\mathsf{SK_U}, \mathsf{VK_U}, \mathsf{SK_F}, \mathsf{VK_F}, \pi) \leftarrow \mathsf{BiGen}(1^k) \\ (m,s) \leftarrow \mathsf{P}^{\mathsf{BiSig_F}}(\mathsf{VK_U}, \mathsf{VK_F}, \pi) \end{array} \right]$$

2. S est UF contre l'utilisateur F si $|Succ_{F,S}(1^k)|$ est négligeable, $Succ_{F,S}$ est défini comme ci-dessous et F n'est pas autorisé à demander à P Π(m).

$$\mathsf{Succ}_{\mathsf{F},\mathcal{S}} \stackrel{\text{def}}{=} \Pr\left[\mathsf{BiVer}(m,s) = succeed \;\middle|\; \begin{array}{c}(\mathsf{SK_U},\mathsf{VK_U},\mathsf{SK_F},\mathsf{VK_F},\pi) \leftarrow \mathsf{BiGen}(1^k)\\ (m,s) \leftarrow \mathsf{F}^{\Pi}(\mathsf{VK_U},\mathsf{VK_F},\mathsf{SK_F})\end{array}\right]$$

3. S est UF contre tout utilisateur U pour tout adversaire A, si $|Succ_{U,S}(1^k)|$ est négligeable, où $Succ_{U,S}$ est défini comme ci-dessous, et U n'est pas autorisé à demander à l'oracle de signature BiSig(m).

$$\mathsf{Succ}_{\mathcal{A},\mathcal{S}} \stackrel{\text{def}}{=} \Pr\left[\mathsf{BiVer}(m,s) = succeed \;\middle|\; \begin{array}{c}(\mathsf{SK_U},\mathsf{VK_U},\mathsf{SK_F},\mathsf{VK_F},\pi) \leftarrow \mathsf{BiGen}(1^k)\\ (m,s) \leftarrow \mathcal{A}^{\mathsf{BiSig_U}}(\mathsf{VK_U},\mathsf{VK_F})\end{array}\right]$$

1.6 Sécurité des schémas Uni(bi)directionnel

1.6.1 Notion de Sécurité

Sécurité parfaite et sécurité au sens de la complexité

La première[4] étape dans l'évaluation de la sécurité est, bien évidemment, la compréhension de la notion de « sécurité ». À ce jour, on en connaît deux définitions majeures. La première, au sens de la théorie de l'information, a été initialement évoquée par Shannon. Elle concerne l'« information » sur le texte clair « contenue » dans le texte chiffré : un schéma de chiffrement est parfaitement sûr si le texte chiffré ne contient aucune information au sujet du texte clair correspondant. Dans le cadre d'un schéma de chiffrement symétrique, il a été démontré que la sécurité parfaite n'est atteinte que si la clé utilisée est aussi longue que le message. Cette condition est une sérieuse limite en pratique. La deuxième approche, au sens de la théorie de la complexité, est actuellement utilisée dans l'analyse des schémas de chiffrement. Elle ne s'intéresse pas au contenu du texte chiffré mais met l'accent sur la « difficulté » d'extraire de l'information sur le texte clair à partir du texte chiffré. Cette difficulté est considérée au sens de la complexité et elle est souvent comparée à la difficulté d'un problème bien défini : la factorisation, par exemple. Remarquons que la notion de « sécurité parfaite » n'est pas applicable au chiffrement asymétrique car le texte chiffré contient toujours de l'information sur le texte clair. En effet, tout attaquant de puissance illimitée peut retrouver le texte clair à partir du texte chiffré grâce à une recherche exhaustive dans l'espace des textes clairs et éventuellement, dans un espace d'aléas (dans le cas d'un chiffrement probabiliste). Dans ce qui suit, nous présentons les différentes notions de sécurité d'un schéma de chiffrement qui sont caractérisées par deux éléments : la difficulté d'extraire de l'information et la puissance dont un attaquant dispose pour casser le schéma.

Réduction en terme de complexité

Dans[4] les premiers schémas de chiffrement à clé publique comme RSA ou Merkle- Hellman , la nature de la sécurité n'a pas été prouvée, faute d'une définition de la sécurité. En 1979, Rabin a introduit un schéma où la capacité d'un attaquant d'extraire le message complet à partir d'un texte chiffré est mathématiquement équivalente à la factorisation. Le schéma de chiffrement de Rabin est décrit comme suit : chaque utilisateur choisit deux nombres premiers de même taille p, q (servant de clé secrète) et publie la clé publique n = pq. Le chiffrement d'un message m est $c = m^2 \mod$ n. Le recepteur, grâce à la connaissance de p, q, peut facilement calculer les quatre racines carrées du chiffrement c et en choisir le message approprié (une façon pratique de conduire à un bon choix est de mettre de la redondance dans le message initial). La sécurité considérée pour ce schéma est dite à sens-unique (dénotée OW, par one-way en anglais). Un schéma est « à sens-unique » si, à partir d'un challenge chiffré, l'attaquant ne peut retrouver le texte clair correspondant. Pour prouver la sécurité, on effectue une réduction d'un problème algorithmique à un problème de sécurité. Dans ce schéma, les deux problèmes sont respectivement une solution de la factorisation et l'extraction du message complet à partir d'un texte chiffré. En entrée, une instance n du problème de factorisation est donnée (n est la multiplication de deux nombres premiers). Le but est de résoudre le problème de factorisation pour l'instance n en utilisant un attaquant contre le schéma. L'attaquant, étant capable d'extraire le message complet à partir d'un texte chiffré, joue le rôle de boîte noire, c'est à dire il reçoit une entrée et en retourne une sortie sans que l'on ne connaisse le mécanisme. Pour utiliser cet attaquant, il nous faut donc construire un simulateur de l'environnement d'attaque. Le simulateur créera un environnement parfaitement similaire (ou au moins indistinguable aux yeux de l'attaquant) à l'environnement réel de l'attaque. Dans le cas du chiffrement de Rabin, un tel simulateur peut être construit de façon simple. En effet, il publie le nombre n comme la clé publique, puis, choisit un texte aléatoire m et le chiffre comme $c = m^2 \mod n$. Le simulateur donne la clé publique et le chiffré c à l'attaquant. Comme il s'agit d'un attaquant passif (qui ne demande pas d'information supplémentaire au système), le simulateur n'a pas à simuler les informations à échanger. La simulation est alors parfaite. L'attaquant retournera les quatre racines carrées correspondant au chiffré c. Le simulateur choisit m' parmi ces quatre racines carrées tel que $m' \neq \pm$m mod n. Alors, un des deux facteurs de n est le plus petit diviseur commun de n et $m - m'$, d'où la factorisation de n. Le schéma de Rabin est ainsi le premier système dans lequel une hypothèse algorithmique, de la difficulté de la factorisation au sens de la complexité, a été réduite à un problème de sécurité : la capacité d'extraire le message complet à partir d'un texte chiffré. Une telle réduction s'appelle aujourd'hui « une preuve de sécurité ».

Fonctions à sens-unique

Dans[4] ce premier exemple, on peut traduire l'« hypothèse algorithmique » de la difficulté de la factorisation en propriété « à sens-unique » de la fonction de multiplication de deux nombres premiers, c'est à dire elle est facile à calculer mais difficile à inverser. Cette notion de fonction à sens-unique est fondamentale en cryptographie. Dans les schémas asymétriques, elle est souvent couplée avec une propriété supplémentaire : l'inverse peut être facilement calculé grâce à quelques informations additionnelles, c'est à dire une trappe. Un des candidats pour les permutations à sens-unique à trappe est la fonction RSA où l'inversion est conjecturée difficile mais devient facile avec la connaissance de la factorisation du module utilisé. Une permutation à sens-unique à trappe implique naturellement un schéma OW contre les attaquants passifs. Cependant, la recherche en cryptographie considère d'autres notions de sécurité plus subtiles que la propriété à sens-unique et d'autres types d'attaquants plus puissants qu'un attaquant passif. À ce stade, une question importante est la suivante : « la cryptographie peut-elle être fondée uniquement sur les hypothèses générales telles que l'existence de fonctions (permutations) à sens-unique ou de fonctions (permutations) à sens-unique à trappe ? ».

Sécurité sémantique et indistinguabilité

Dans[4] les années 80, les chercheurs ont formellement défini les notions de sécurité pour les primitives cryptographiques (notamment pour la signature et pour le chiffrement). Goldwasser et Micali ont introduit la notion de la sécurité sémantique et ont proposé la construction d'un chiffrement probabiliste qui peut garantir la sécurité sémantique. La notion de sécurité sémantique est une notion très forte, elle couvre l'exigence de ne pas pouvoir extraire une information, même partielle (ne serait-ce qu'un seul bit) sur le clair à partir du chiffré. Elle coïncide avec la notion de la sécurité parfaite limitée aux attaques polynomiales probabilistes. La sécurité sémantique est la propriété désirée pour tout schéma de chiffrement, mais, en réalité, on étudie souvent la notion de l'indistinguabilité (dénotée IND), plus simple à analyser.Pour Goldwasser et Micali (l'indistinguabilité implique la sécurité sémantique) et Micali, Rackoff et Sloan (la sécurité sémantique implique l'indistinguabilité) ont démontré qu'elle était équivalente à la notion de la sécurité sémantique. Avec l'indistinguabilité, l'attaquant ne peut pas trouver deux textes clairs m0 et m1 dont il pourrait distinguer les chiffrements : il reçoit un challenge c, chiffré de m0 ou m1 et ne doit pas pouvoir deviner à quel texte clair c correspond. Cette condition implique immédiatement que le chiffrement soit probabiliste. De ce fait, les chiffrements déterministes de RSA et de Rabin n'atteignent pas la sécurité sémantique. Les premiers schémas qui ont atteint cette notion forte sont le schéma de Goldwasser et Micali

qui repose sur la difficulté du problème de la résiduosité quadratique, et le schéma de Yao , fondé sur l'hypothèse générale de l'existence de fonctions injectives à sens-unique à trappe. Cependant, ces deux schémas sont totalement inefficaces car leurs chiffrements sont « bit par bit », i.e. chaque bit est chiffré de manière indépendante, elles conduisent à des chiffrés très larges. Un schéma plus efficace (à chiffrés plus courts) ayant atteint la propriété d'indistinguabilité est celui de Blum et Goldwasser . Il est fondé sur la difficulté du problème de la factorisation. L'idée dans ce schéma est d'utiliser le générateur Blum-Blum-Shub pour générer des bits aléatoires qui masqueront ensuite le clair.

Attaques à Chiffrés Choisis(Chosen Ciphertext Attack) Ainsi[4], l'indistinguabilité a été initialement définie dans le scénario de base où l'attaquant n'a accès qu'à l'information publique et, de ce fait, peut chiffrer des clairs de son choix, d'où le nom d'« attaques à clairs choisis » (dénotée CPA, par chosen plaintext attack en anglais). Désormais, on dénote un schéma qui est sûr au sens XXX (IND, par exemple) face aux attaques YYY (CPA, par exemple), un schéma $XXX - YYY$ sûr (IND - CPA sûr, Naor et Yung ont étudié la notion d'attaque à chiffrés choisis dans le cas où l'attaquant peut accéder à un oracle (dit l'oracle de déchiffrement) qui retourne, pour chaque chiffré, le clair correspondant. Face à la puissance de ce type d'attaque, tous les schémas précédents deviennent vulnérables. Naor et Yung ont construit le premier schéma résistant à ce type d'attaque en exploitant la notion de preuve à divulgation de connaissance nulle d'appartenance à un langage (dénotée preuve NIZK, par non-interactive zero-knowledge proof en anglais), introduite par Blum, Feldman et Micali. Dans, Blum et. al ont déjà considéré des attaques à chiffrés choisis. Ils ont même suggéré une manière de construire un schéma résistant aux attaques à chiffrés choisis.

Blum et. al ont proposé le schéma (sans construction formelle) suivant : au lieu d'envoyer le chiffré $c = E(m)$, il faut envoyer c et σ, une preuve non-interactive zeroknowledge, justifiant que le déchiffrement de c est connu de l'expéditeur. Bien que la notion de NIZK soit proposée dans le même article, remarquons qu'il s'agit plutôt d'une preuve de connaissance dans la construction. Le déchiffrement vérifie d'abord si σ est convaincante, auquel cas il retourne m, sinon, il ne retourne rien. De cette façon, l'oracle de déchiffrement semble inutile : pour que l'oracle retourne le message, l'attaquant doit soumettre un chiffré avec une preuve convaincante de connaissance du déchiffrement, c'est à dire il doit déjà connaître le clair. Ainsi, une fois que le chiffrement E est sémantiquement sûr face aux attaques passives, le chiffrement modifié l'est également face aux attaques à chiffrés choisis. Cependant, Blum et. al n'ont pas expliqué les étapes de construction d'une telle preuve de connaissance et n'ont pas formellement prouvé la sécurité. Le schéma proposé

par Naor et Yung s'est inspiré de l'idée de Blum et. al avec une utilisation judicieuse des preuves NIZK. Le point de départ était toujours un chiffrement E IND - CPA sûr.

Le chiffrement d'un message m retourne un couple ($c_1 = E(e1, m)$, $c_2 = E(e2, m)$) avec deux clés publiques différentes e1, e2 (les deux clés secrètes correspondantes sont d1, d2) ainsi qu'une preuve NIZK σ que c_1 et c_2 sont deux chiffrés du même message.

Le déchiffrement ne retourne le message que si la preuve est convaincante. L'idée est que la connaissance d'une des deux clés secrètes d1, d2 est suffisante pour le déchiffrement. Alors, le simulateur, en choisissant une de ces deux clés, disons d2, pourra bien simuler le déchiffrement et réduire une attaque à chiffrés choisis contre le nouveau schéma à une attaque passive contre $E(e1)$.

Le schéma résiste ainsi aux attaques à chiffrés choisis. Cependant, le modèle d'attaque considéré par Naor et Yung est à chiffrés choisis non adaptative (dénoté CCA1) au sens où les requêtes ne doivent pas dépendre du challenge. En effet, si l'attaquant peut faire des requêtes qui dépendent du challenge, la construction de Naor et Yung ne résiste plus aux attaques à chiffrés choisis. Le schéma de Naor et Yung joue cependant un rôle très important en théorie car c'est le premier schéma à considérer des attaques à chiffrés choisis. De plus, il ne repose que sur une hypothèse générale. En effet, puisque l'existence d'une permutation à sens-unique à trappe est suffisante pour construire un schéma IND - CPA sûr et une preuve NIZK, elle est aussi suffisante pour le schéma IND - CCA1 sûr de Naor et Yung.

Le modèle d'attaque à chiffrés choisis adaptative (dénoté CCA2) a été étudié pour la première fois par Rackoff et Simon . Dans ce modèle, l'attaquant peut soumettre des requêtes à l'oracle de déchiffrement à tout instant, avant et après la réception du challenge, avec pour seule restriction de ne pas soumettre de requête sur le challenge. Rackoff et Simon ont formalisé la notion de preuve non-interactive zero-knowledge de connaissance, c'est à dire la version non-interactive de la notion de preuve zero-knowledge de connaissance. En examinant cette dernière, Rackoff et Simon ont proposé un schéma IND - CCA2 sûr, fondé sur l'hypothèse générale de l'existence de permutations à sens-unique à trappe. Cependant, chaque expéditeur et chaque destinataire dans ce schéma doivent faire confiance à un centre, un troisième participant qui génére et leur distribue la clé publique ainsi que la clé secrète. Par conséquent, ce schéma ne rentre pas dans la catégorie classique des schémas de chiffrement à clé publique

Non-malléabilité

La[4] construction d'un schéma IND - CCA2 sûr est réalisée grâce à une nouvelle notion de sécurité : la non-malléabilité (dénotée NM). Introduite par Dolev, Dwork et Naor, elle est considérée comme une extension de la sécurité

sémantique (une autre notion de non-malléabilité équivalente et plus simple à étudier sera ultérieurement introduite par Bellare, Desai, Pointcheval et Rogaway), elle est en général plus forte que l'indistinguabilité. Bellare et. al ont pourtant prouvé que la non-malléabilité et l'indistinguabilité sont équivalentes face aux attaques à chiffrés choisis adaptatives. La non-malléabilité exclut les attaques qui, en possession d'un chiffré, peuvent produire un nouveau chiffré tel que les clairs correspondants soient clairement reliés sans pour autant les connaître. Dolev et. al ont construit un schéma complexe qui considère l'interaction entre plusieurs chiffrements et repose sur des preuves NIZK ainsi que des signatures à usage unique (« one-time signature » en anglais). Ce schéma est prouvé non-malléable contre des attaques à chiffrés choisis adaptatives. Leur schéma est donc le premier schéma IND - CCA2 fondé sur l'hypothèse générale de l'existence de permutations à sens-unique à trappe. Une autre construction, beaucoup plus simple, se fondant sur la même hypothèse, est due à Sahai. Avec une nouvelle notion de preuve non-malléable NIZK [109], Sahai a montré qu'une telle preuve peut être déduite d'une preuve NIZK. Il montrera ensuite qu'avec le remplacement de la preuve NIZK σ par une preuve non-malléable NIZK σ' dans le schéma IND - CCA1 de Naor et Yung, on peut rendre ce schéma résistant aux attaques à chiffrés choisis adaptatives.

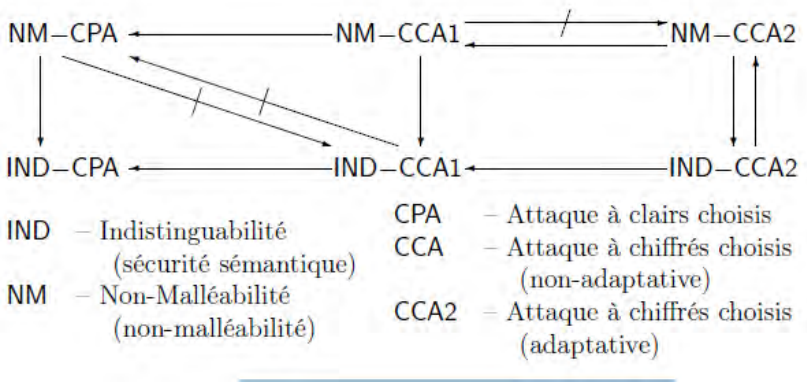


Figure 8 – Relations entre les notions de sécurité

1.6.2 Sécurité pour le chiffrement Uni(bi)directionnel

Chosen Ciphertext Attacks (CCA1)

Pendant[4] l'attaque du texte chiffré choisi, un cryptanalyste peut analyser tous les textes chiffrés choisis avec leurs textes en clair correspondants. Son objectif est d'acquérir une clé secrète ou d'obtenir autant d'informations que possible sur le système attaqué.

L'attaquant a la capacité de forcer la victime (qui connaît évidemment la clé secrète) à déchiffrer tout texte chiffré et à lui renvoyer le résultat. En analysant le texte chiffré choisi et le texte en clair reçu correspondant, l'intrus tente de deviner la clé secrète qui a été utilisée par la victime.

Les attaques par texte chiffré choisi sont généralement utilisées pour briser des systèmes avec un cryptage à clé publique. Par exemple, les premières versions du chiffrement RSA étaient vulnérables à de telles attaques. Ils sont moins souvent utilisés pour attaquer des systèmes protégés par des chiffrements symétriques. Certains chiffrements de flux auto-synchronisés ont également été attaqués avec succès de cette manière.

Adaptive Chosen Ciphertext Attacks (CCA2)

L'attaque de texte chiffré adaptatif est une sorte d'attaque de texte chiffré choisi, au cours de laquelle un attaquant peut amener le système attaqué à déchiffrer de nombreux textes chiffrés différents. Cela signifie que les nouveaux textes chiffrés sont créés en fonction des réponses (textes en clair) reçues précédemment. L'attaquant peut demander le déchiffrement de nombreux textes chiffrés.

Il existe assez peu d'attaques pratiques de texte chiffré adaptatif choisi. Ce modèle est plutôt utilisé pour analyser la sécurité d'un système donné. Prouver que cette attaque ne brise pas la sécurité confirme que toute attaque

réaliste de texte chiffré choisi ne réussira pas.

Chosen-Plaintext Attacks (CPA)

Pendant l'attaque en clair choisi, un cryptanalyste peut choisir des données en clair arbitraires à chiffrer, puis il reçoit le texte chiffré correspondant. Il essaie d'acquérir la clé de chiffrement secrète ou, alternativement, de créer un algorithme qui lui permettrait de déchiffrer tout message chiffré à l'aide de cette clé (mais sans connaître réellement la clé secrète).

C'est une situation plutôt confortable pour l'attaquant. Il peut obtenir plus d'informations sur la clé secrète et sur l'ensemble du système attaqué, car il est capable de choisir n'importe quel texte à traiter par le chiffrement. Il peut analyser le comportement du système et produire un texte chiffré, basé sur tout type de données d'entrée.

Lors de la rupture de chiffrements déterministes avec la clé publique, l'intrus peut facilement créer une base de données avec des textes chiffrés populaires, par exemple avec des requêtes courantes sur le serveur. Après cela, il pourra trouver la signification de nombreux messages cryptés interceptés, en les comparant simplement avec ses propres entrées de base de données.

Les attaques en clair choisis les plus connues ont été effectuées par les cryptanalystes alliés pendant la Seconde Guerre mondiale contre les chiffrements Enigma allemands .

Attaque Adaptive-Chosen-Plaintext

Dans ce type d'attaque en clair choisi, l'intrus a la capacité de choisir du texte en clair pour le chiffré plusieurs fois. Au lieu d'utiliser un gros bloc de texte, il peut choisir le plus petit, recevoir son texte chiffré, puis en fonction de la réponse, en choisir un autre, et ainsi de suite. Cela lui permet d'enquêter sur le système attaqué plus en détail.

1.6.3 Sécurité pour la signature Uni(bi)directionnel

Unforgability against chosen-message attacks(UF-CMA)

Un code d'authentification[6] de messages (MAC) MAC = (K, T) avec un espace de messages MsgSp associé est défini par deux algorithmes :

- L'algorithme de génération de clé aléatoire K renvoie une clé secrète k.

- L'algorithme déterministe de marquage MAC T prend en entrée la clé secrète K et un texte en clair M $\in$ MsgSp pour renvoyer une étiquette mac pour M. Pour une paire message-étiquette (M, σ), on dit que σ est une étiquette valide pour M si $\sigma = \sigma'$

$\sigma' \leftarrow$ TK (M). La définition de sécurité suivante exige qu'aucun adversaire ne puisse forger une étiquette valide pour un nouveau message.

Soit MAC = (K, T) un schéma MAC. On dit qu'il est infalsifiable contre les attaques par messages choisis ou UF-CMA sécurisé si tout adver-

saire A disposant de ressources raisonnables ne peut réussir l'expérience suivante qu'avec une faible probabilité, appelée l'avantage UF-CMA du schéma MAC.Mais peut réussir l'expérience suivante seulement avec une petite probabilité, appelée l'avantage UF-CMA de A. Dans l'expérience, la clé aléatoire k est tout d'abord générée par K. L'adversaire a accès à l'oracle de marquage TK(-). Il réussit s'il peut produire une paire message-étiquette (M, σ) tel que M $\in$ MsgSp, σ est une étiquette valide pour M sous K, et M n'a pas été interrogé à l'oracle d'étiquetage. Une autre définition de sécurité (plus forte) exige que la sortie du MAC soit indiscernable d'une chaîne aléatoire.

Conclusion :

La cryptographie a défini les notions de sécurité et prouvé la sécurité des cryptosystèmes de chiffrement, de codes d'authentification de messages et de signatures numériques.

Dans ce document, nous avons décrire les fonctions unidirectionnelles et les fonction bidirectionnelle. La notation unidirectionnelle est justifiée par le fait que le mandataire P doit aider l'utilisateur F pour chaque message qui doit être décrypté ou signé. Pour les fonction bidirectionnelle le mandataire p doit aider l'utilisateur F et tout autre utilisateur pour chaque messages qui doit être décrypté.

2 RSA Unidirectionnel et Bidirectionnel

Introduction :

Le chiffrement[1] RSA nommé par les initiales de ses trois inventeurs,**Ronald Rivest, Adi Shamir et Leonard Adleman** est un algorithme de cryptographie asymétrique, très utilisé dans le commerce électronique, et plus généralement pour échanger des données confidentielles sur Internet. Cet algorithme a été décrit en 1977. RSA a été breveté par le Massachusetts Institute of Technology (MIT) en 1983 aux États-Unis. Le brevet a expiré le 21 septembre 2000. Le chiffrement RSA est considéré comme l'une des procédures de clé publique les plus sûres et les mieux décrites. Un mécanisme à clé publique comme le RSA autorise la production d'une signature numérique. Il suffit, pour s'engager, d'élever le document que l'on souhaite signer à la puissance exposant privé modulo n. Le résultat constituera la signature du document. Quiconque pourra vérifier la signature en l'élevant à la puissance

exposant public modulo n, mais personne ne pourra produire la signature sans la connaissance de l'exposant privé.

Nous allons voir dans ce chapitre le chiffrement et la signature uni(bi)directionnel basé sur RSA . Il est très utilisé dans le commerce électronique et pour échanger des données confidentielles sur Internet. Le chiffrement RSA[1] est asymétrique.

2.1 Chiffrement RSA

Fonctionnement

Le chiffrement RSA[W6] est asymétrique : il utilise une paire de clés (des nombres entiers) composée d'une clé publique pour chiffrer et d'une clé privée pour déchiffrer des données confidentielles. Les deux clés sont crées par une personne, souvent nommée par convention Alice, qui souhaite que lui soient envoyées des données confidentielles. Alice rend la clé publique accessible. Cette clé est utilisée par ses correspondants (Bob, etc.) pour chiffrer les données qui lui sont envoyées. La clé privée est quant à elle réservée à Alice, et lui permet de déchiffrer ces données. La clé privée peut aussi être utilisée par Alice pour signer une donnée qu'elle envoie, la clé publique permettant à n'importe lequel de ses correspondants de vérifier la signature.

Une condition indispensable est qu'il soit « impossible par calcul » de déchiffrer à l'aide de clé publique, en particulier de reconstituer la clé privée à partir de la clé publique, c'est-à-dire que les moyens de calcul disponibles et les méthodes connues au moment de l'échange (et le temps que le secret doit être conservé) ne le permettent pas.

Le chiffrement RSA est souvent utilisé pour communiquer une clé de chiffrement symétrique, qui permet alors de poursuivre l'échange de façon confidentielle : Bob envoie à Alice une clé de chiffrement symétrique qui peut ensuite être utilisée par Alice et Bob pour échanger des données.

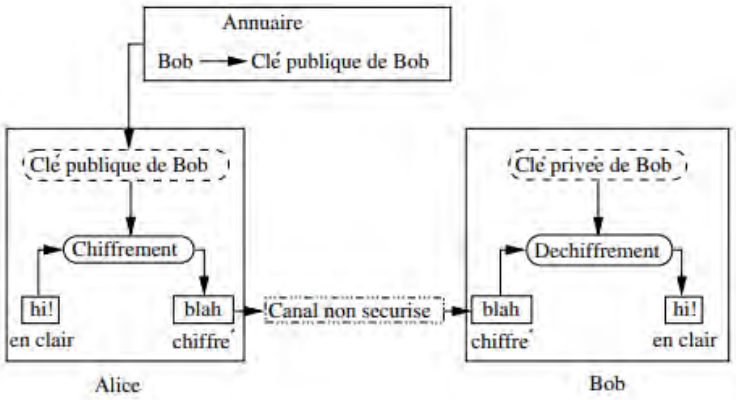


Figure 9 – Schéma de chiffrement et de déchiffrement entre Alice et Bob

2.2 Principe du chiffrement

Il est basé sur le calcul exponentiel[W5]. Sa sécurité repose sur la fonction unidirectionnelle suivante : le calcul du produit de 2 nombres premiers est aisé. La factorisation d'un nombre en ses deux facteurs premiers est beaucoup plus complexe. Il s'agit du système le plus connu et le plus largement répandu, basé sur l'élévation à une puissance dans un champ fini sur des nombres entiers modulo un nombre premier. Il emploie de grands nombres entiers (par exemple représentés sur 1024 bits). Ce cryptosystème utilise deux clés d et e, interchangeables. Le chiffrement se fait selon

$C = M^e \mod n$

et le déchiffrement par

$M = C^d \mod n$

On possède une paire de clés, l'une publique (e, n) et une privée (d, n). La première étape revient à choisir n. Il doit s'agir d'une valeur assez élevée, produit de 2 nombres premiers très grands p et q. En pratique, si p et q ont 100 chiffres décimaux, n possèdera 200 chiffres. Selon le niveau de sécurité souhaité, la taille de n peut varier : 512 bits, 768, 1024 ou 20483. Dans un second temps, on choisira un très grand entier e, relativement premier à $\phi(n)$=(p-1) *(q-1). La clé publique sera formée par (e,n). On choisira ensuite un d tel que $e * d = 1 \mod (\phi(n))$

La clé privée sera donnée par (d, n). Dernière phase : on jette p et q. Le cryptanalyste devant retrouver ces valeurs, il faut les détruire pour éviter les fuites. On peut donc dire que RSA a pour but :

- Génération de 2 nombres premiers p et q

- Calcul de n = p*q
- Déterminer e tel que $3 < e < \phi(n)$ et (e, ϕ(n)) = 1
- Calculer d tel que $e * d = 1 \mod (\phi(n))$
- Clé publique : (e,n)
- Clé privée : (d,n)
- p et q doivent rester secrets, voire supprimés
- $C = M^e \mod n$
- $M = C^d \mod n$

Il n'est pas très astucieux de choisir d'aussi petites valeurs car on peut retrouver d très facilement. En pratique, il faut prendre de très grandes valeurs de p et q. Pour retrouver ces grandes valeurs, il faudra alors utiliser le Jacobien et le test de Solovay-Strassen par exemple.

2.3 Signature RSA

Le cryptosystème[5] à clé publique RSA fournit un schéma de signature numérique cryptographiquement sécurisé (signe + vérifier), basé sur les mathématiques des exponentiations modulaires et des logarithmes discrets et sur la difficulté du problème de factorisation d'entiers (IFP). Le processus de signature / vérification RSA fonctionne comme suit :

• L'algorithme de signe RSA calcule un hachage de message, puis chiffre le hachage avec l'exposant de clé privée pour obtenir la signature. La signature obtenue est un nombre entier (le hachage du message chiffré RSA).

• L'algorithme de vérification RSA calcule d'abord le hachage du message, puis déchiffre la signature du message avec l'exposant de clé publique et compare le hachage déchiffré obtenu avec le hachage du message signé pour s'assurer que la signature est valide.

Les signatures RSA sont déterministes (le même message + la même clé privée produisent la même signature). Une variante non déterministe des signatures RSA est facile à concevoir en complétant le message d'entrée avec quelques octets aléatoires avant de signer. Les signatures RSA sont largement utilisées dans la cryptographie moderne, par ex. pour signer des certificats numériques, pour protéger les sites Web. Par exemple (à partir de novembre 2018), le site Web officiel de Microsoft utilise Sha256RSA pour son certificat numérique. Néanmoins, la tendance au cours de la dernière décennie est de passer de RSA et DSA à des signatures basées sur des courbes elliptiques (comme ECDSA et EdDSA). Les cryptographes et développeurs modernes préfèrent les signatures ECC pour leur longueur de clé plus courte, leur signature plus courte, une sécurité plus élevée (pour la même longueur de clé) et de meilleures performances.

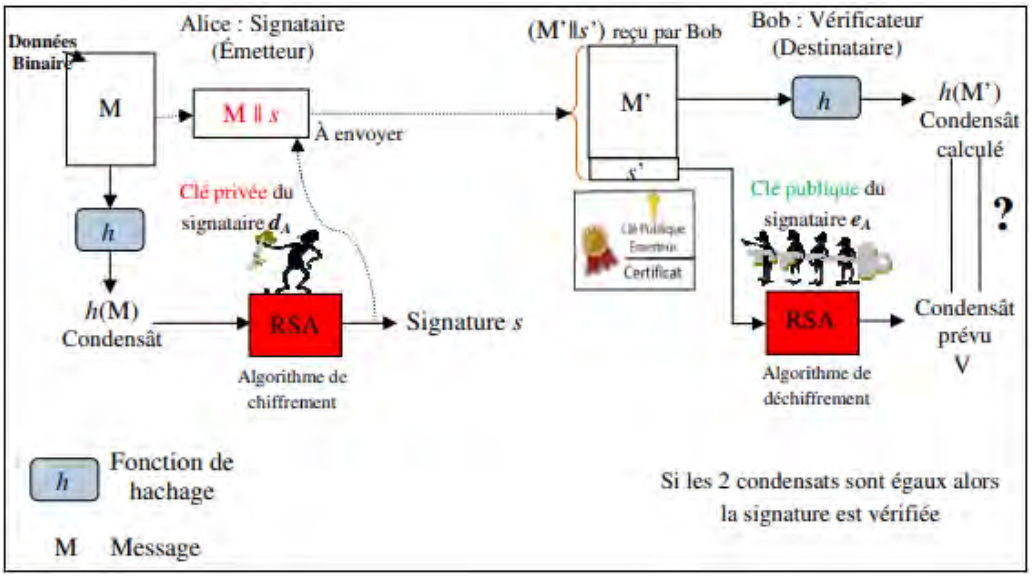


Figure 10 – Processus de génération et de vérification de la signature en utilisant l'algorithme RSA

2.4 Algorithmes de chiffrement et de signature asymétriques

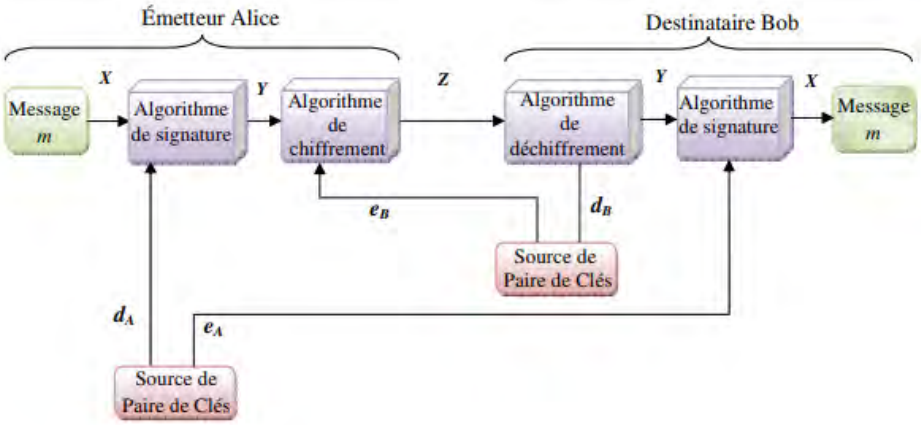


Figure 11 – Cryptosystème à clé publique assurant la confidentialité et l'authenticité des messages

Dans cette figure 9, Alice veut envoyer un message signé et chiffré à Bob. Alice utilise sa clé privée d_A pour signer le message, et la clé publique de Bob e_B pour chiffrer le message signé. Lorsque Bob reçoit le message, il le

déchiffre avec sa clé privée d_B et vérifie la signature d'Alice par la clé publique e_A d'Alice.

2.5 Efficacité et robustesse de RSA

Grâce aux algorithmes[5] tel-que : El Gamal, Menezes-Vanstonne, McEliece et Chor-Rivest, il est facile de générer des grands nombres premiers, tout au moins en acceptant un taux d'erreur dans le cas de RSA, l'erreur n'est pas trop grave. En effet, si l'on commet une erreur en croyant que p et q sont premiers, le destinataire se rendra rapidement compte que les nombres ne sont pas premiers : soit la clef d n'est pas inversible, soit certains blocs du message décrypté sont incompréhensibles. Dans ce cas, on peut procéder à un changement de système RSA (recalcul de p et q). - Le calcul du couple (e, d) est extrêmement facile, il suffit d'appliquer l'algorithme d'Euclide étendu . - Enfin, le chiffrement et le déchiffrement sont réalisés par exponentiation modulaire. Nous avons vu que cette exponentiation pouvait être réalisée assez efficacement. La sécurité fournie par RSA repose essentiellement sur la difficulté à factoriser de grands entiers. En effet, si un attaquant peut factoriser le nombre n = p*q, de la clé publique, il peut alors déduire directement (p-1) *(q-1) et donc calculer la clé privée K_d à partir de la clé publique K_e par l'algorithme d'Euclide étendu. Donc, si l'on dispose d'un algorithme rapide pour factoriser de grands entiers, casser RSA devient facile aussi. Après 20 ans de recherche, aucun moyen plus efficace que la factorisation de n n'a été publié pour casser RSA. Cependant, la réciproque : « si factoriser de grands entiers est dur alors casser un message crypté par RSA est dur » n'a pas été prouvée. Autrement dit, on ne sait pas aujourd'hui si casser RSA est aussi difficile que factoriser n. On peut cependant montrer l'équivalence entre « déterminer d à partir de (n, e) » et « factoriser n ». Un sens est trivial : connaissant p et q, on peut facilement calculer d puisque c'est ce qui est fait dans le cadre de la génération de clé. Réciproquement, calculer p et q à partir de (n, e, d) peut être obtenu : - par un algorithme déterministe si e est petit

- par un algorithme probabiliste (une variante de l'algorithme de Miller-Rabin) dans le cas général. La robustesse du RSA est basée sur la difficulté pratique de factoriser un grand nombre entier n (qui est le produit de deux grands nombres premiers p et q) dans un temps raisonnable. Il utilise généralement des clés de 1024 bits.

2.6 Un exemple

Alice génére ses clés RSA en choisissant deux nombres premiers :$p = 11$ et $q = 13$. Le module $n = p.q = 143$. L'indicatrice d'Euler en n $\phi(n) =$

$(p-1).(q-1) = 120$. Elle choisit 7 comme clé publique RSA e et calcule sa clé privée RSA en utilisant l'algorithme d'Euclide étendu, ce qui lui donne 103. Bob veut envoyer à Alice un message chiffré M ; il lui demande donc sa clé publique RSA (n, e) qui, dans cet exemple, est (143, 7). Son message en texte clair est simplement le chiffre 9. Il est chiffré dans le cryptogramme C comme suit :

M^e mod n = $9^7 mod 143 = 48 = C$

Lorsque Alice reçoit le message de Bob, elle le déchiffre à l'aide de sa clé privée RSA (d, n) comme suit :

$C^d mod n = 48^{1}03 mod 143 = 9 = M$

Si elle souhaite utiliser les clés RSA pour signer numériquement un message, Alice doit créer un hachage ou condensé de son message, chiffrer la valeur de hachage à l'aide de sa clé privée RSA puis l'ajouter au message. Bob peut alors vérifier que le message a bien été envoyé par Alice et qu'il n'a pas subi de modification en déchiffrant la valeur de hachage à l'aide de la clé publique de Alice. Si cette valeur correspond au hachage du message d'origine, seule Alice peut l'avoir envoyé (authentification et non-répudiation) et le message est exactement tel qu'elle l'a rédigé (intégrité). Bien sûr, Alice pourrait chiffrer son message à l'aide de la clé publique RSA de Bob (confidentialité) avant de le lui envoyer. Un certificat numérique renferme des informations qui identifient le propriétaire du certificat et contient également la clé publique de ce propriétaire. Les certificats sont signés par l'autorité de certification qui les émet. Ils contribuent à simplifier le processus d'obtention des clés publiques et à vérifier leur propriétaire.

2.7 Sécurité de RSA

Comme indiqué[5] plus haut, la sécurité de RSA repose sur la difficulté que représente la factorisation de grands entiers. Cependant, à mesure que la puissance de traitement augmente et que des algorithmes de factorisation plus efficaces sont découverts, il devient possible de factoriser des nombres de plus en plus élevés. La puissance du chiffrement est directement liée à la taille de la clé. De ce fait, un doublement de la longueur de la clé renforce le chiffrement de façon exponentielle, au détriment toutefois des performances.

Les clés RSA font généralement 1024 ou 2048 bits, mais les experts pensent que les clés de 1024 bits pourraient être déchiffrées à brève échéance. C'est la raison pour laquelle l'administration et le secteur privé commencent à adopter des clés d'une longueur minimale de 2048 bits. A moins d'une percée imprévue en informatique quantique, de nombreuses années devraient s'écouler avant que des clés plus longues soient nécessaires, mais la cryptographie à courbe elliptique (ECC, Elliptic Curve Cryptography) fait de plus

en plus d'adeptes parmi les spécialistes de la sécurité pour mettre en oeuvre le chiffrement à clé publique en remplacement de RSA.

En effet, cette technologie permettra de créer des clés de chiffrement à la fois plus rapides, plus petites et plus efficaces. La plupart des matériels et logiciels actuels sont déjà compatibles ECC et la popularité de la discipline devrait encore augmenter, car elle offre une sécurité équivalente pour une puissance de traitement et une consommation d'énergie moindres, ce qui la rend plus adaptée que RSA aux applications mobiles.

Enfin, une équipe de chercheurs parmi lesquels Adi Shamir, co-inventeur de RSA, a réussi à déchiffrer une clé RSA de 4096 bits en utilisant la cryptanalyse acoustique. Cependant, n'importe quel algorithme de chiffrement est vulnérable à ce type d'attaque.

Les inventeurs de l'algorithme RSA ont fondé RSA Data Security en 1983. La société a ensuite été acquise par Security Dynamics, à son tour rachetée par EMC Corporation en 2006. L'algorithme RSA a été diffusé dans le domaine public par RSA Security en 2000.

2.8 RSA Unidirectionnel

Système de cryptage unidirectionnel basée sur RSA Aujourd'hui, RSA[W2] est considéré comme le premier algorithme, publié scientifiquement, qui permet la transmission de données chiffrées sans échange de clé secrète.

Le chiffrement RSA utilise un algorithme basé sur la multiplication de grands nombres premiers. Bien qu'il n'y ait généralement pas de problème à multiplier deux nombres premiers par 100, 200 ou 300 chiffres, il n'y a toujours pas d'algorithme efficace capable de décomposer le résultat d'une telle opération de calcul en ses facteurs premiers dans l'étape inverse. La factorisation en nombres premiers peut être illustrée par l'exemple ci-dessous.

Si on multiplie les nombres premiers 14.629 et 30.491, on obtient le produit 446.052.839, qui n'a que quatre diviseurs : 1, lui-même et les deux nombres premiers qui ont été multipliés. Si on supprime les deux premiers diviseurs, puisque chaque nombre est divisible par 1 et par lui-même, on obtient les valeurs initiales 14.629 et 30.491.

Ce schéma est la base de la génération des clés RSA. Les clés publiques et privées représentent deux paires de nombres entiers :

Clé publique : (e, N)

Clé privée : (d, N)

N est le module RSA. Ceci est contenu dans les deux clés et les résultats du produit de deux nombres premiers très grands p et q ($N = p \times q$) sélectionnés au hasard.

Pour générer la clé publique, on a besoin de e, qui est un nombre choisi au hasard avec certaines restrictions. Si on combine N et e, on obtient la clé publique accessible à chaque abonné de communication en texte clair.

Pour générer la clé privée N et d sont nécessaires. d est une valeur résultant des nombres premiers p et q générés aléatoirement ainsi que du nombre aléatoire e, qui sont calculés sur la base de la fonction phi d'Euler (ϕ).

Les nombres premiers inclus dans le calcul de la clé privée doivent être tenus secrets afin de garantir la sécurité du cryptage RSA. Le produit des deux nombres premiers, d'autre part, peut être utilisé en texte clair dans la clé publique, puisqu'il est supposé qu'il n'y a pas d'algorithme efficace aujourd'hui qui peut décomposer le produit de nombres premiers très grands en ses facteurs dans le temps pertinent. Il n'est donc pas possible de calculer p et q à partir de N. Sauf si nous pouvons raccourcir le calcul. Une telle trappe représente la valeur d, qui n'est connue que du propriétaire de la clé privée.

La sécurité de l'algorithme RSA dépend fortement du progrès technique. La puissance de calcul des ordinateurs double tous les deux ans. Il n'est donc pas exclu qu'une méthode efficace de factorisation primaire soit disponible dans un avenir prévisible à l'échelle habituelle. Dans ce cas, RSA offre la possibilité d'adapter l'algorithme au développement technique en utilisant des nombres premiers encore plus grands pour calculer les clés.

Supposons que nous ayons le schéma de cryptage RSA E = (Enc-Gen, Enc, Dec). L'algorithme de génération de clé RSA produit la clé publique EK = (e,N) et la clé secrète SK = $(d, N, \phi(N))$, où $e * d = 1 \bmod \phi(\mathrm{N})$, N = pq, p,q sont deux grands nombres premiers et ϕ est la fonction Euler. Le cryptage est défini comme suit : EncEK(m) = m^e mod N = c. L'algorithme de décryptage est DecEK(c) = c^d mod N = m. L'algorithme de génération de clé unidirectionnelle E' = (UniGen, UniEnc, UniDec, PDec, FDec). Pour chaque utilisateur U, l'algorithme de génération de clés UniGen(1^k) génère une paire de clés publiques (EK,DK) et divise la clé secrète en deux parties d1 et d2 telles que d = d1+d2 mod ϕ(N). Le mandataire P obtient DKP = d1 et l'utilisateur F obtient DKF = d2. La fonction de cryptage UniEnc et les algorithmes UniDec sont identiques aux algorithmes originaux Enc et Dec. La transformation des fonctions PDec et FDec exécutent le décryptage avec les clés d1 et d2 respectivement. La fonction d'exactitude du RSA unidirectionnel est donnée par l'égalité FDecd2(PDecd1(Ence(m))) = m.

2.9 Signature unidirectionnelle basée sur RSA

RSA-Hash unidirectionnel Régime de signature Supposons que[2] nous ayons S = (Sig-Gen, Sig, Ver) une signature standard RSA-Hash (Full

Domain Hash). Sig-Gen génére la clé publique VK = (e,N) et la signature clé secrète SK = (d, N). La fonction de signature est définie comme $Sig = hash(m)^d modN = s$, où hash est une fonction de hachage associée à Sig. La vérification renvoie réussie si $s^e = hash(m) modN$. Sinon, elle renvoie échouée. Le schéma de signature standard RSA-Hash S est transformé dans un système de signature unidirectionnel S' = (UniGen, UniSig, UniVer, PSig, FSig). L'algorithme de génération de clés UniGen génére des clés pour tous les utilisateurs U en exécutant Sig-Gen et en divisant ensuite chaque clé secrète d en deux parties d = d1 + d2 mod ϕ(N). d1 devient la clé SKP du proxy P et d2 devient la clé SKF de l'utilisateur F. Les algorithmes de signature et de vérification UniSig et UniVer sont identiques respectivement aux algorithmes Sig et Ver. L'utilisateur F utilise la fonction FSig pour générer une partie de la signature unidirectionnelle RSA-Hash en calculant s' = Sigd2(m). Le proxy P utilise la fonction PSig pour générer l'autre partie de l'unidirectionnel Signature RSA en calculant s = Sigd1(m). La fonction de signature est formée par s et s'. Le schéma standard de signature RSA-Hash est infalsifiable contre les attaques de messages choisis. Ainsi, nous prouvons formellement dire dans le prochain théorème que le système RSA-Hash présente le même niveau de sécurité.

Théorème

Soit S = (Sig-Gen, Sig, Ver,) un classique RSA-Système de signature électronique. Considérons que S' = (UniGen, UniSig, UniVer, PSig, FSig) est un système unidirectionnel RSA-Système de signature électronique construit comme ci-dessus. S' est UF-CMA contre le proxy P, l'utilisateur F, et tous les utilisateurs U. Le RSA-Hash probabiliste décrit de meilleures sécurité que la fonction RSA-Hash utilisée ci-dessus, et peut être transformée en une primitive unidirectionnelle si nous permettons à l'utilisateur F de générer le caractère aléatoire nécessaire.

Preuve

1. Supposons que P puisse casser le schéma RSA-Hash unidirectionnel. Cela signifie que |SuccP,S(1^k)| est non négligeable. En se basant sur le proxy P, nous construisons un faussaire B capable de casser le schéma de cryptage RSA. Le faussaire B reçoit en entrée une clé publique (N, e) et essaie d'inverser $x = f^-1(y)$, où f est la fonction RSA définie par N et e. L'adversaire B commence à faire fonctionner le proxy P pour cette clé publique et un nombre d1 choisi au hasard et donné comme clé secrète. Lorsque le proxy P effectue la i-ème requête de hachage, l'adversaire cherche à voir si le message mi a déjà été demandé. Si ce n'est pas le cas, il choisit aléatoirement xi, définit $h(mi) = xi^e$ avec une probabilité p et $h(mi) = y * xi^e$ avec une probabilité de 1 - p. Si le proxy P demande à FSig un message mi, l'adversaire renvoie xi si mi a été demandé auparavant. Sinon, il échoue. Au final, P produit

une signature RSA-Hash unidirectionnelle correcte (m, s) pour tout nouveau message m. Si le message m n'a pas été haché auparavant, l'adversaire calcule sa valeur hash. Si $h(m) = y * xi^e$ alors l'adversaire renvoie $y^d = s/xi^e$ comme $x = f^-1(y)$.

2.10 RSA Bidirectionnel

2.10.1 Schéma de chiffrement bidirectionnel basée sur RSA

Dans un schéma[W5] bidirectionnel , le schéma de chiffrement est réversible c'est-à-dire que la clé de cryptage peut être utilisée pour traduire des messages de Bob à Charlie, ainsi que de Charlie à Bob. Cela peut avoir diverses conséquences sur la sécurité, selon l'application. Une caractéristique notable des schémas bidirectionnels est que le délégant et la partie déléguée (par exemple, Charlie et Bob) doivent combiner leurs clés secrètes pour produire la clé de rechiffrement.

Le schéma bidirectionnel RSA est judicieux pour la communication bidirectionnelle client / serveur car si un client se connecte à un serveur, le serveur et le client générent tous deux des clés RSA et se transmettent les clés publiques. Si le client souhaite envoyer un message, il le crypte avec la clé publique du serveur, l'envoie et le serveur le déchiffre. Vice versa avec serveur vers client.

Si les nouvelles clés publiques RSA ne sont pas authentifiées, on est pas en sécurité contre les attaquants actifs, qui pourraient intercepter la connexion (ou remplacer l'un des partenaires). Ceux-ci pourraient lire tout le contenu ou le remplacer par le leur.

Si nous utilisons des clés publiques authentifiées (par exemple des certificats), la confidentialité serait assurée (c'est-à-dire qu'un attaquant ne pourrait pas lire les messages). Mais les messages ne sont pas d'authentifiés, et un attaquant actif pourrait remplacer les messages par les siens, insérer les siens ou supprimer les messages.

2.10.2 Système de signature bidirectionnelle basée sur RSA

Dans un schéma de signature de proxy, un proxy semi-fiable reçoit des informations qui lui permettent de transformer la signature d'Alice sur un message m en signature de Bob sur m, mais le proxy ne peut pas générer des signatures pour Alice ou Bob.

Les signatures proxy[2] permettent à Alice de déléguer ses droits de signature à Bob seulement si le proxy coopère. En pratique, Bob et le proxy peuvent générer conjointement une signature sur des messages arbitraires au

nom d'Alice. Ceci est généralement accompli en divisant le secret d'Alice en deux actions qui sont distribué à Bob et au proxy (chacun ne reçoit qu'un seul partage). Une signature d'Alice sur un message est généré en combinant deux signatures partielles sur le même message calculé par Bob et le procuration sous leurs propres actions, respectivement. Dans la signature par proxy, un proxy «traduit» une signature parfaitement valide et vérifiable publiquement d'Alice sur un certain message, σ_A (m), en une autre signature de Bob sur le même message, σ_B (m). Autrement dit, dans la signature par proxy, les deux signatures, l'une d'Alice et l'autre de Bob telles que générées par le proxy, peuvent coexister et les deux peuvent être vérifiés publiquement comme étant deux signatures des deux personnes distinctes sur le même message. De plus, le proxy peut convertir une seule signature en signatures multiples de plusieurs signataires distincts, et vice-versa ! Toute signature de proxy peut être utilisée pour créer une signature de proxy, mais l'inverse n'est pas nécessairement vrai. Par exemple, il est possible de construire une signature proxy basée sur RSA (comme dans[1], en fractionnant le secret d'Alice d en d1 et d2 tel que d = d1 + d2) mais il n'est pas possible d'avoir un schéma de démission de proxy qui se traduit entre deux signatures RSA publiquement vérifiables partageant le même module (en raison du problème / attaque de module commun). Une autre propriété unique de la démission par procuration est que la «traduction» d'une signature à une autre peut être effectuée en séquence et plusieurs fois par des mandataires distincts sans même nécessiter l'intervention des entités signataires. Ainsi, les précieuses clés secrètes peuvent rester hors ligne. Les signatures générées dans ce processus toutes des signatures valides et vérifiables publiquement sur le même message provenant d'entités distinctes.

Conclusion :

Dans ce chapitre nous avons présenté le cryptosystème RSA, le chiffrement et la signature unidirectionnel et bidirectionnel basé sur RSA.

Le monde actuel de la cryptographie est en perpétuelle évolution afin de pouvoir répondre aux besoins de sécurisation des données qui ne cessent d'augmenter.

La sécurité fournie par RSA se repose essentiellement sur la difficulté à factoriser de grands entiers. RSA effectue donc le chiffrement avec de nombres premier c'est un de ces plus grands avantages. Mais la factorisation d'un très grand nombre en deux autres nombres premiers prend plus de temps ce qui rend le cryptage plus lent. L'inconvénient avec cette méthode est qu'il faut utiliser de grandes clés, et leur génération prend quand même un certain

temps

3 Application de RSA Standard et RSA Unidirectionnel

Introduction :

Pour réussir cette phase nous allons utiliser les outils et technologies nécessaires, ci-dessous, à l'implémentation de notre application.

- **Python**

Figure 12 – Logo Python

Python est un langage de programmation open source interprété côté serveur et non compilé. Créé par Guido van Rossum, il est utilisé pour le développement web, de jeux vidéo et autres logiciels ainsi que pour les interfaces utilisateurs graphiques. Il a notamment été utilisé dans la création d'Instagram, de YouTube et de Spotify, et est l'un des langages de programmation officiels de Google. L'interpréteur Python peut être facilement étendu par de nouvelles fonctions et types de données implémentés en C ou C++ (ou tout autre langage appelable depuis le C). Python est également adapté comme langage d'extension pour personnaliser des applications.

- **Django**

Figure 13 – Logo de django

Django est un framework python open-source consacré au développement web 2.0 . Les concepteurs de Django lui ont attribué le slogan suivant : " Le framework web pour les perfectionnistes sous pression ". Il est donc clairement orienté pour les développeurs ayant comme besoin de produire un projet solide rapidement et sans surprise ... c'est à dire à tous les développeurs !

Comme il est toujours compliqué de partir de rien, Django nous propose une base de projet solide. Django est donc une belle boîte à outils qui aide et oriente le développeur dans la construction de ses projets.

Django a été créer en 2003 et a été publié sous licence BSD en juillet 2005.

• **Pycharm**

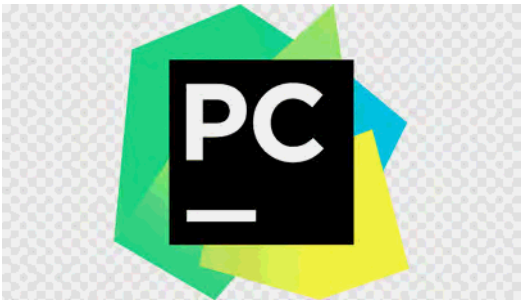


Figure 14 – Logo pycharm

PyCharm est un environnement de développement intégré dédié aux langages Python et Django. Ce logiciel existe en deux versions disponibles en téléchargement, une version libre et gratuite appelée Community Edition et une version complète payante Professional Edition.

nous disposerons d'une foule de fonctionnalités d'édition comme la coloration syntaxique, la complétion automatique du code ainsi que sa factorisation. Le logiciel s'avère utile pour analyser et réparer automatiquement le code source Python et Django.

3.1 Implémentation de RSA Standard

Montrons comment les algorithmes RSA fonctionnent en Python. Le code ci-dessous générera une paire de clés RSA aléatoire, cryptera un message qu'on veut chiffré et le décryptera à sa forme d'origine, en utilisant le schéma de remplissage RSA-OAEP.

Tout d'abord, installons le package pycryptodome, qui est une puissante bibliothèque Python de primitives cryptographiques de bas niveau (hachages, codes MAC, dérivation de clé, chiffrements symétriques et asymétriques, signatures numériques) :

```
(myenv) mamy@mamy-ThinkPad-X240:~/Documents/appliRSA$ pip install pycryptodome
```

Figure 15 – Installation du bibliothèque pycryptodome

Importation des packages pour l'algorithme RSA en python

```
from pathlib import Path
from django.shortcuts import render
from django import template
from django.http import HttpResponse

from django.views.decorators.csrf import csrf_exempt
import json
import zlib
import base64
import ast
register = template.Library()
from .models import *
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
```

```
from Crypto.Cipher import PKCS1_OAEP
from Crypto.Cipher import PKCS1_v1_5
import binascii
from hashlib import sha512
```

Figure 16 – Importation des packages

3.1.1 Chiffrement/déchiffrement

Maintenant, écrivons le code Python. Tout d'abord, générons les clés RSA (3072 bits) et récuperons la clé publique dans un fichier qu'on nomme **"public.pem"** et la clé privée dans un fichier appelé **"private.pem"** au format PEM

```
def index(request):
    new_key = RSA.generate(3072)
    private_key = new_key.exportKey("PEM")
    public_key = new_key.publickey().exportKey("PEM")
    private_key_path = Path('private.pem')
    private_key_path.touch(mode=0o600)
    private_key_path.write_bytes(private_key)
    public_key_path = Path('public.pem')
    public_key_path.touch(mode=0o664)
    public_key_path.write_bytes(public_key)
```

```
private_key = Path('private.pem')
public_key = Path('public.pem')
rsa_public_key = RSA.importKey(public_key.read_bytes())
rsa_public_key = PKCS1_OAEP.new(rsa_public_key)
rsa_private_key = RSA.importKey(private_key.read_bytes())
rsa_private_key = PKCS1_OAEP.new(rsa_private_key)

print(f"Public key:  (n={hex(new_key.n)}, e={hex(new_key.e)})")
print(f"Private key: (n={hex(new_key.n)}, d={hex(new_key.d)})")
```

Figure 17 – Génération des clés

Voici la clé publique dans le fichier **"public.pem"**

```
-----BEGIN PUBLIC KEY-----
MIIBojANBgkqhkiG9w0BAQEFAAOCAY8AMIIBigKCAYEA54do46Z16k3mupDADaxU
QcYwFi4eKPr5W3MGQwD0hy1TMV0Ph++QZgND0lEZvQvtIejfUwokNu3y4q59dzwX
mzUR6JSlwukMjwYGZjge4ZNF1mm0nNKgDjPEmdVUOnWATIxONJsQVjAy9Jfrh5/7
ee43lqUrLWFg5tahAwdvuVoHXmR24DK6blF825xGAYLc6OH+LhBdy7CbAB5+aSjK
SRd7MANIK6t4nPZ+bgnqL7pUvApalyALyZIDCBUwczCFCpLjJ/nCiEmVJoIlg1EW
egvora36Is6CkQ0ic5PKrzXWNWYdiusxRVbDtdjo13XlKc5+b8FPRDaX+rFGX4fh
IQBse/BN9H6e1KCPdB1wHwpieLeo8gldgs+i9wcy9/Xq0R1b7yjtxR17I0z/Ama5
tnox++3ba7COhllVNx1GiTQ7EQV8zM7WhpVtOVXQc6DTYJ/NI7jNeBo6p/kf2OS8
Vw3cvnTbbxVf2Nr551UK8dwe2t2+z/RWIfwDedrFqzxTAgMBAAE=
-----END PUBLIC KEY-----
```

Figure 18 – La clé publique

La clé privée est stockée dans le fichier **"private.pem"**

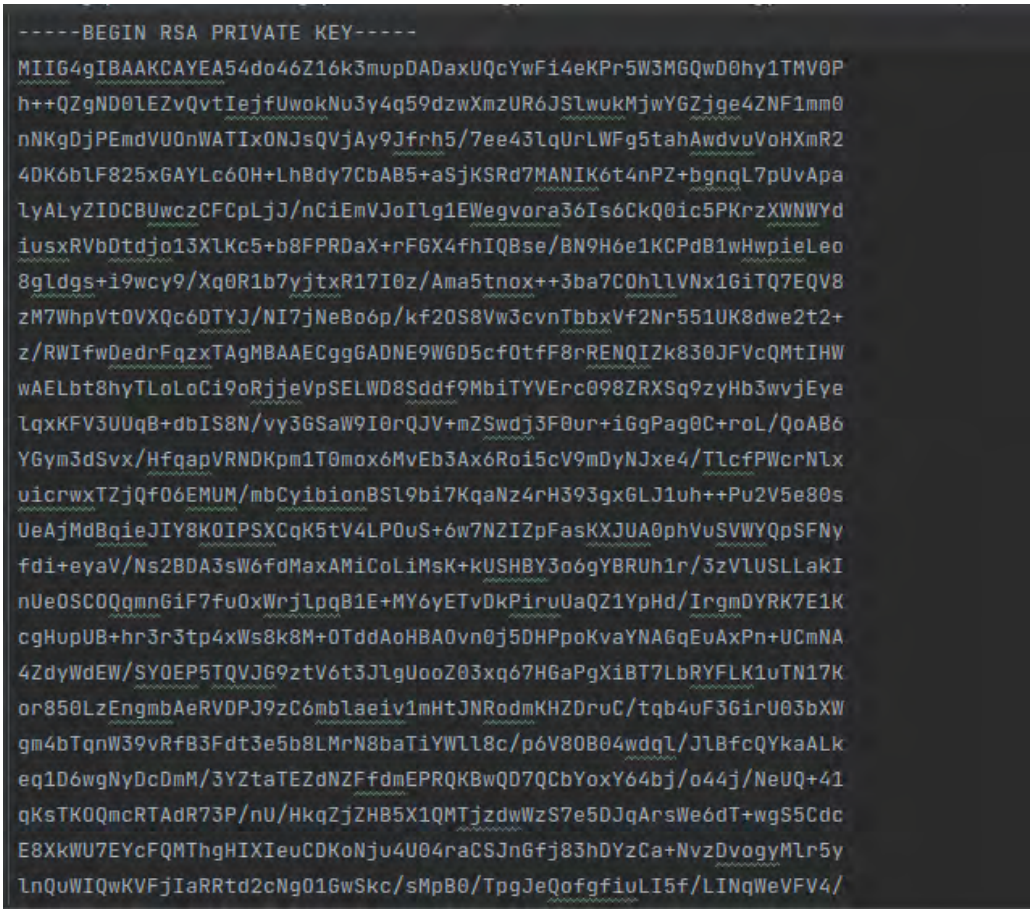


Figure 19 – La clé Privée

On récupére les nombres (n, e) qui forment la clé publique

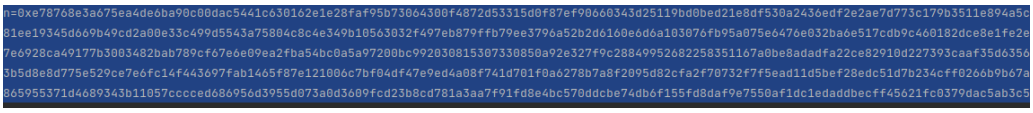


Figure 20 – Le nombre n

Figure 21 – Le nombre e

Et les nombres (n, d) qui forment la clé privée.

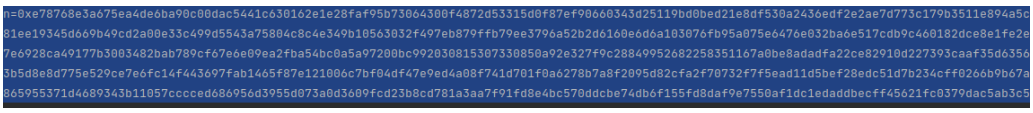


Figure 22 – Le nombre n

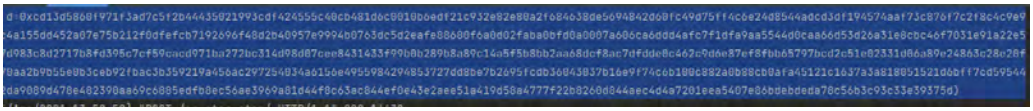


Figure 23 – Le nombre d

Ensuite, on crypte le message au moyen de la clé publique RSA (ici supposée être disponible localement dans le fichier **"public.pem"**) à l'aide du schéma de cryptage RSA-OAEP (RSA avec remplissage OAEP PKCS1) et le text chiffré est stocké dans le fichier **"encrypt.txt"**.

```
if request.method == 'POST' and request.FILES:
    if request.POST.get("form_type") == 'form_encrypt':
        myfile = request.FILES['input_file_message']
        msg = myfile.read()
        encrypted = rsa_public_key.encrypt(msg)
        encrypted_message = binascii.hexlify(encrypted)
        f = open('encrypt.txt', 'wb')
        f.write(encrypted)
        f.close()
```

Figure 24 – Le chiffrement du text en claire

Enfin, on décrypte le message en utilisant RSA-OAEP avec la clé privée RSA qui est stockée dans le fichier **"private.pem"**, le message décrypté se trouve dans le fichier **decrypt.txt**.

```
elif request.POST.get("form_type") == 'form_decrypt':
    myfile = request.FILES['input_file_encrypt']
    fenc = myfile.read()
    print("fenc", fenc)
    try:
        decrypted = rsa_private_key.decrypt(fenc)
        decrypted_message = binascii.hexlify(decrypted)
        f = open('decrypt.txt', 'wb')
        f.write(str(decrypted))
        f.close()
    except ValueError as err:
        print("errorr ", err)
```

Figure 25 – Le déchiffrement du message chiffré

3.1.2 Signature/Vérification

Maintenant, signons un message, en utilisant la clé privée RSA (n, d). Calculons son hachage et augmentons le hachage à la puissance d modulo n (chiffrons le hachage avec la clé privée). Nous utiliserons le hachage SHA-512, il s'adaptera à la taille de clé RSA actuelle (3072). En Python, nous avons l'exponentiation modulaire comme fonction intégrée pow (x, y, n).

La signature numérique obtenue est un entier compris entre zéro et la longueur de la clé RSA [0 ... n] qui est stocké dans le fichier **"sign.txt**, comme illustré ci-dessous

```
elif request.POST.get("form_type") == 'form_sign':
    myfile = request.FILES['input_file_sign']
    message = myfile.read()
    print("message", message)
    hash = int.from_bytes(sha512(message).digest(), byteorder='big')
    signature = pow(hash, new_key.d, new_key.n)
    f = open('sign.txt', 'wb')
    f.write(str(signature))
    f.close()
```

Figure 26 – La signature du message en claire

Ensuite, vérifions la signature en déchiffrant la signature à l'aide de la clé publique (élevons la signature à e modulo n) et comparons le hachage obtenu de la signature(hastFromSignature) au hachage du message signé à l'origine(hash)

```
    elif request.POST.get("form_type") == 'form_verify':
        f = open('sign.txt', 'r')
        signature = f.read()
        myfile = request.FILES['input_file_verify']
        msg = myfile.read()
        hash = int.from_bytes(sha512(msg).digest(), byteorder='big')
        hashFromSignature = pow(int(signature), new_key.e, new_key.n)
        valid = hash == hashFromSignature

return render(request, 'rsa_standard.html', locals())
```

Figure 27 – La vérification de la signature

3.2 Implémentation de RSA Unidirectionnel

3.2.1 Chiffrement/Déchiffrement

Pour l'implémentation de RSA unidirectionnel on crée d'abord deux paires de clés de taille 3072 et 4096. chaque clé est stockée dans un fichier nommé respectivement par "**publicuni1.pem**", "**privateuni1.pem**", "**publicuni2.pem**",et "**privateuni2.pem**". On a donc deux clés publiques et deux clés privées

```
def rsa_uni(request):
    new_key = RSA.generate(3072)
    private_key_uni1 = new_key.exportKey("PEM")
    public_key_uni1 = new_key.publickey().exportKey("PEM")
    private_key_uni1_path = Path('privateuni1.pem')
    private_key_uni1_path.touch(mode=0o600)
    private_key_uni1_path.write_bytes(private_key_uni1)
    public_key_uni1_path = Path('publicuni1.pem')
    public_key_uni1_path.touch(mode=0o664)
    public_key_uni1_path.write_bytes(public_key_uni1)

    private_key_uni1 = Path('privateuni1.pem')
    public_key_uni1 = Path('publicuni1.pem')
    rsa_public_key_uni1 = RSA.importKey(public_key_uni1.read_bytes())
    rsa_public_key_uni1 = PKCS1_OAEP.new(rsa_public_key_uni1)
    rsa_private_key_uni1 = RSA.importKey(private_key_uni1.read_bytes())
    rsa_private_key_uni1 = PKCS1_OAEP.new(rsa_private_key_uni1)

    print(f"Public key:  (n={hex(new_key.n)}, e={hex(new_key.e)})")
    print(f"Private key: (n={hex(new_key.n)}, d={hex(new_key.d)})")
```

```
keypair= RSA.generate(4096)
private_key_uni2 = keypair.exportKey("PEM")
public_key_uni2 = keypair.publickey().exportKey("PEM")
private_key_uni2_path = Path('privateuni2.pem')
private_key_uni2_path.touch(mode=0o600)
private_key_uni2_path.write_bytes(private_key_uni2)
public_key_uni2_path = Path('publicuni2.pem')
public_key_uni2_path.touch(mode=0o664)
public_key_uni2_path.write_bytes(public_key_uni2)

private_key_uni2 = Path('privateuni2.pem')
public_key_uni2 = Path('publicuni2.pem')
rsa_public_key_uni2 = RSA.importKey(public_key_uni2.read_bytes())
rsa_public_key_uni2 = PKCS1_OAEP.new(rsa_public_key_uni2)
rsa_private_key_uni2 = RSA.importKey(private_key_uni2.read_bytes())
rsa_private_key_uni2 = PKCS1_OAEP.new(rsa_private_key_uni2)

print(f"Public key:  (n={hex(new_key.n)}, e={hex(new_key.e)})")
print(f"Private key: (n={hex(new_key.n)}, d={hex(new_key.d)})")
```

Figure 28 – Génération des clés

On récupére les nombres n, e et d pour chaque clé
• Pour les clés de tailles 3072 on a :
- les nombres (n,e) qui forment la clé publique ci-dessous

```
(n=0xecc76886c8c7742d202e11ab8967e6b430a6c2f18cc72be5abd024d8c49b2b1adeb981ea77b860dd6490aa8b8f25f365ac52856c035d8832f36433515981f268337d16e
58e213c82f4c82dd5d4b84bb549ba98a7bda5d8bb63006c6446bba61739f34fdb5cdd54dc2ee1a24e629995266f9032a479fc04b1d88155f1adf9f2ba076f0d41c893b0f46242
3364b1ccc5ebdf915c335b92596c66ee67c76bdb64bc69db9e893ffdcf2fbbcb46bd0e27df01b3b82d82f78bce8908385a8e12dd868a5da6cf85e374dd56e3d5121dab62e3e91
c809428372c7ed7e5c8033be87372a86808ac7ae5cb07276cb909642a86e8370adb14cabec5ad570f7a714207bcd32500bcf5f022dfc17d02b5ed32e348450b5a36989a115aaf
67dd38da9844a6c0439c58954a0884bd91df5f4139901a6c091c28184c012e76b68b5cfaa6fcf9c2a032d3185c31a70f78269ff2af121844eb4d14fc5e6c34f48aca1c2015ebf
```

Figure 29 – Le nombre n

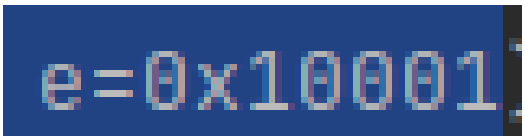


Figure 30 – Le nombre e

La clé publique

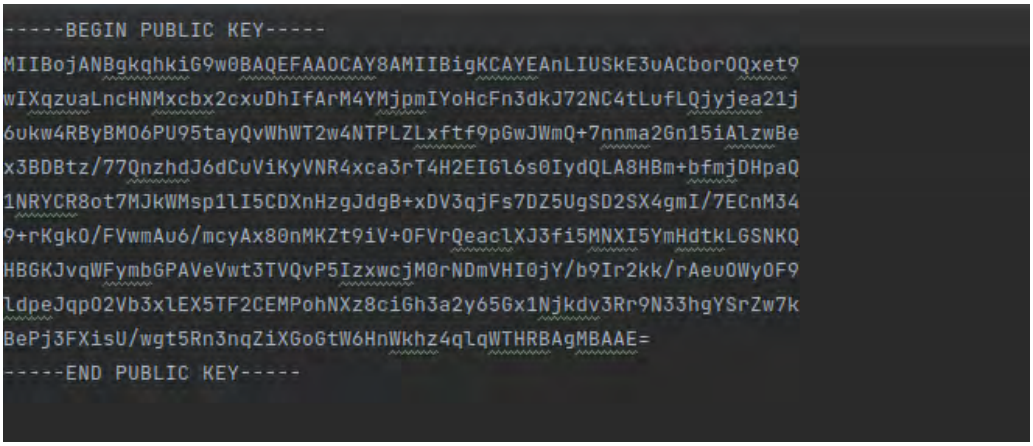


Figure 31 – La clé publique 1

- Les nombres (n,d) qui forment la clé privée ci-dessous

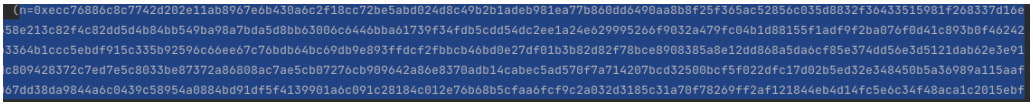
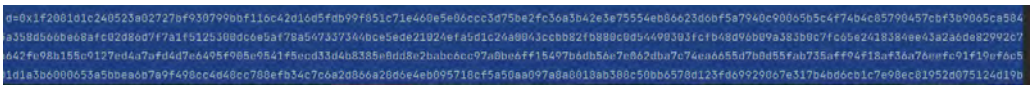


Figure 33 – Le nombre n



La clé privée

Figure 34 – La clé privée 1

• Pour les clés de tailles 4092 on a :
- les nombres (n,e) qui forment la clé publique ci-dessous

Figure 35 – Le nombre n

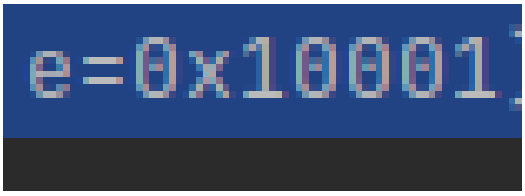


Figure 36 – Le nombre e

La clé publique

Figure 37 – La clé publique 2

- Les nombres (n,d) qui forment la clé privée ci-dessous

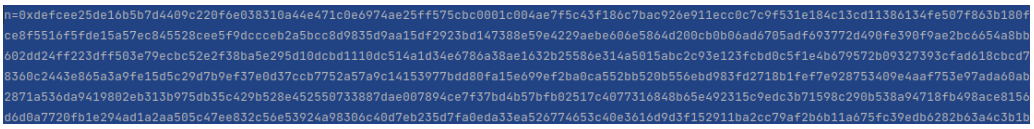


Figure 38 – Le nombre n

Figure 39 – Le nombre d

La clé privée

Figure 40 – La clé privée 2

Ensuite on crypte le message en claire avec la première clé publique *rsa_public_key_uni*1 qui est stockée dans le fichier **publicuni1.pem** et on obtient un message chiffré nommé "encrypted" se trouvant dans le fichier "encryptuni1.txt". Ce dernier sera transformé avec la deuxiéme clé publique **rsa_public_key** qui se trouve dans le fichier **publicuni2.pem** on obtient un second chiffré nommé rsa_encrypted stocké dans le fichier "encryptuni2.txt"

```
if request.method == 'POST' and request.FILES:
  if request.POST.get("form_type") == 'form_encrypt':
    myfile = request.FILES['input_file_message']
    msg = myfile.read()
    encrypted = rsa_public_key_uni1.encrypt(msg)
    encrypted_message = binascii.hexlify(encrypted)
    f = open('encryptuni1.txt', 'wb')
    f.write(encrypted)
    f.close()

    msg1 = encrypted
    rsa_encrypted = rsa_public_key_uni2.encrypt(msg1)
    # print("uni", rsa_encrypted)
    f = open('encryptuni2.txt', 'wb')
    f.write(str(rsa_encrypted))
    f.close()
```

Figure 41 – Le chiffrement des messages

En fin on déchiffre le message "rsa_encrypted" avec la clé privée $rsa\_private\_key\_uni2$ on obtient un déchiffré nommé "decrypted" qu'on stocke dans le fichier "decrypt1.txt". On decrypt le message "decrypted" avec la clé privée $rsa\_private\_key\_uni1$, on obtient "rsa_decrypted" se trouvant dans le fichier "decrypt2.txt" qui contient le message en claire.

```
elif request.POST.get("form_type") == 'form_decrypt':
    myfile = request.FILES['input_file_encrypt']
    msg = myfile.read()
    encrypted = rsa_public_key_uni1.encrypt(msg)
    msg1 = encrypted
    rsa_encrypted = rsa_public_key_uni2.encrypt(msg1)
    decrypted = rsa_private_key_uni2.decrypt(rsa_encrypted)
    print('Decrypted:', decrypted)
    f = open('decrytp1.txt', 'wb')
    f.write(str(decrypted))
    f.close()

    rsa_decrypted = rsa_private_key_uni1.decrypt(decrypted)
    f = open('decrypt2.txt', 'wb')
    f.write(str(rsa_decrypted))
    f.close()
```

Figure 42 – Le déchiffrement des messages chiffrés

3.3 Test et Validation

3.3.1 RSA standard

a. **Page de gérération des clés**

On clic sur le bouton "Générer la paire de clé" et les clés sont générées dans les fichiers **public.pem** et **private.pem** ci-dessus.

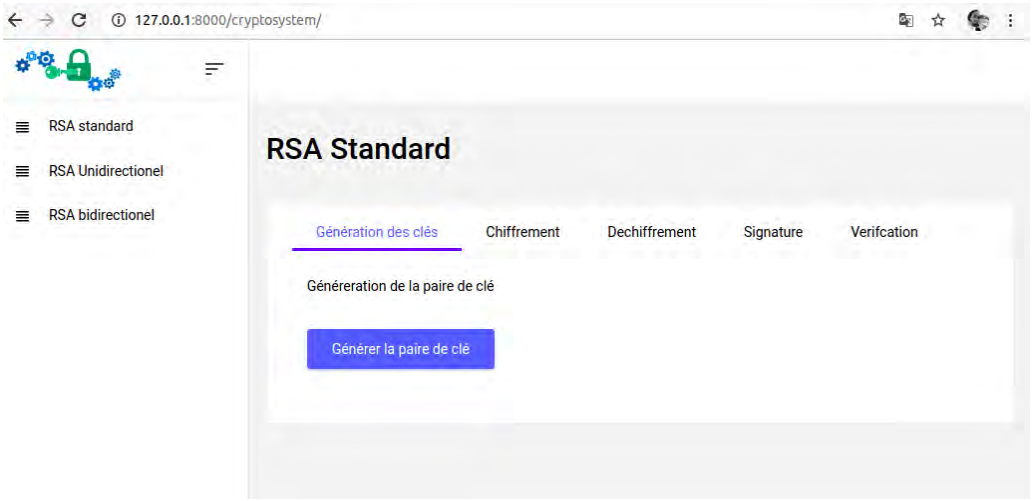


Figure 43 – Page de génération des clés

b. **Page de chiffrement du message**

On veut chiffrer le message stocké dans le fichier "plaintext.txt" ci-dessous avec la clé public générée dans le fichier "public.pem"

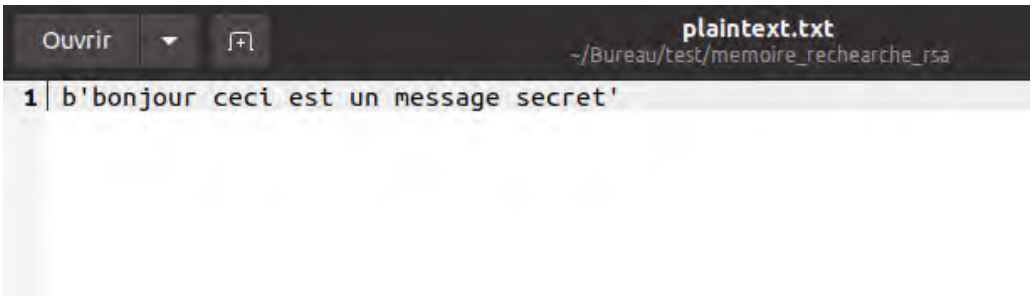


Figure 44 – Le fichier plaintext.txt

D'abord on clic sur le bouton "Upload" et on choisit le fichier "plaintext.txt" contenant le text a chiffré. Ensuite on clic sur "Chiffrer" et le message est chiffré et gardé dans le fichier "encrypt.txt" comme illustré ci-dessous.

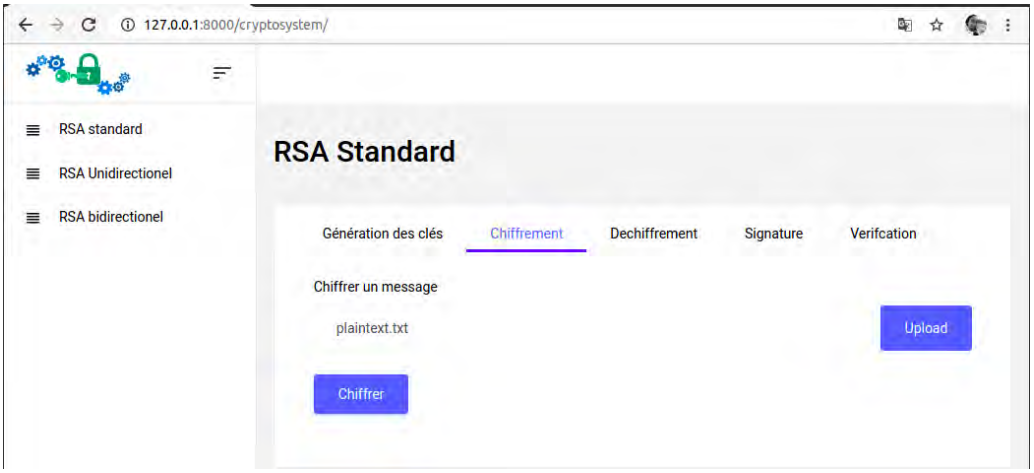


Figure 45 – Page de chiffrement du message

Figure 46 – Le fichier encrypt.txt

c. **Page de déchiffrement du message**

On clic sur le bouton "Upload", ensuite on choisit le fichier contenant le chiffré "encrypt.txt" et enfin on clic sur le bouton "Déchiffrer". Le message est déchiffré à l'aide de la clé privée stockée dans "private.pem" et gardée dans "decrypt.txt".

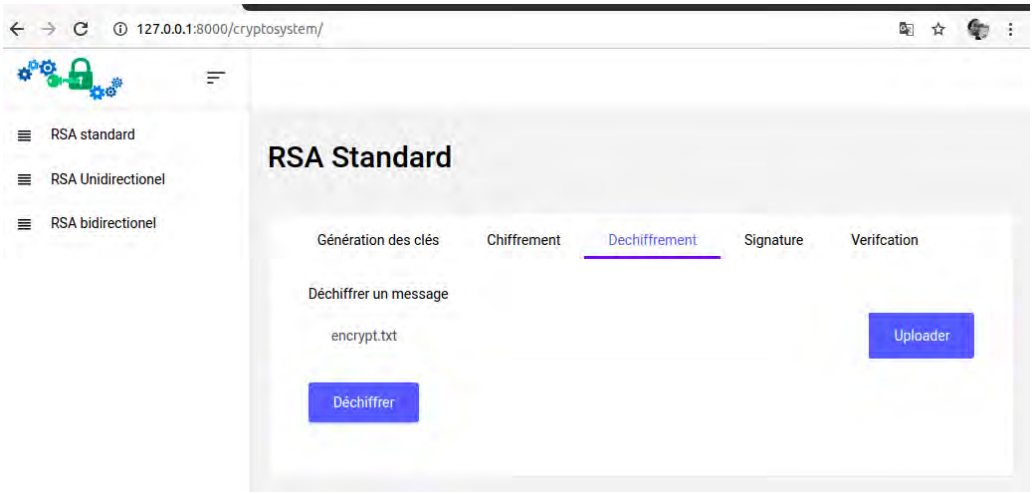


Figure 47 – Page de déchiffrement du message

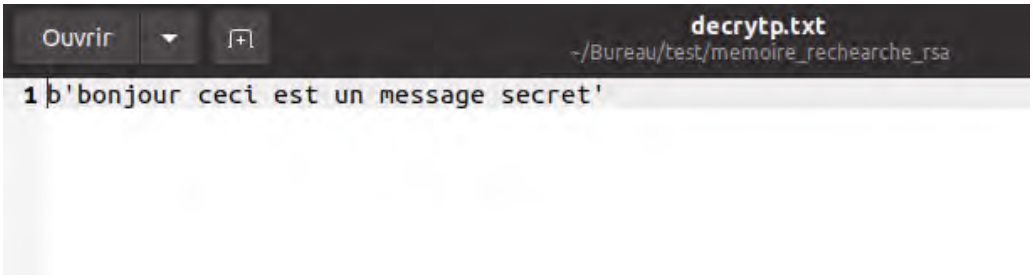


Figure 48 – Le fichier decrypt.txt

d. **Page de signature du message**

on veut signer le message en claire, se trouvant dans le fichier "plaintext.txt", avec la clé privée.

on clic sur "Uplaod" on choisit le fichier "plaintext.txt" et on clic sur le bouton "Signer". Le message signé est gardé dans le fichier **sign.txt**.

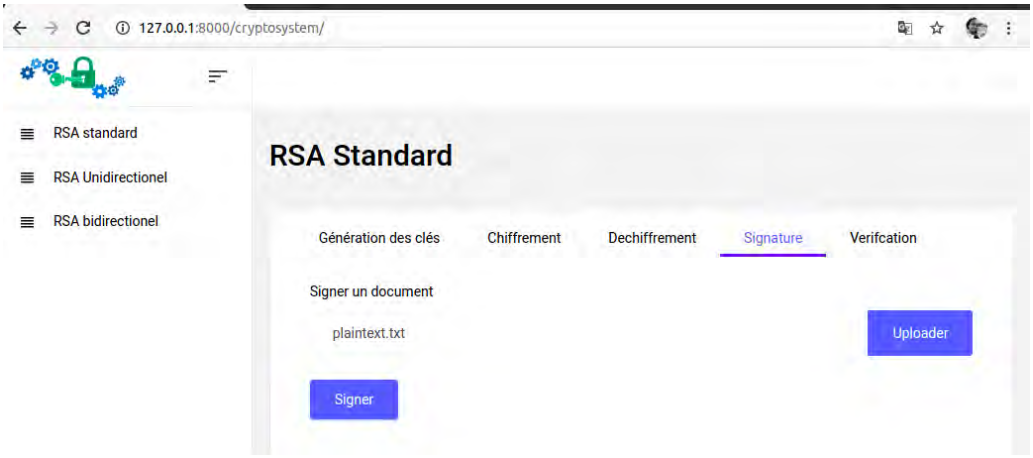


Figure 49 – Page Signature du message

Figure 50 – Le fichier sign.txt

e. **Page de verification de la signature**

Maintenant on va vérifier la signature, on clic le bouton "Uplaod" on choisit le fichier **sign.txt**. Ensuite on clic sur le bouton "Verifier" et on obtient le resultat ci-dessous :

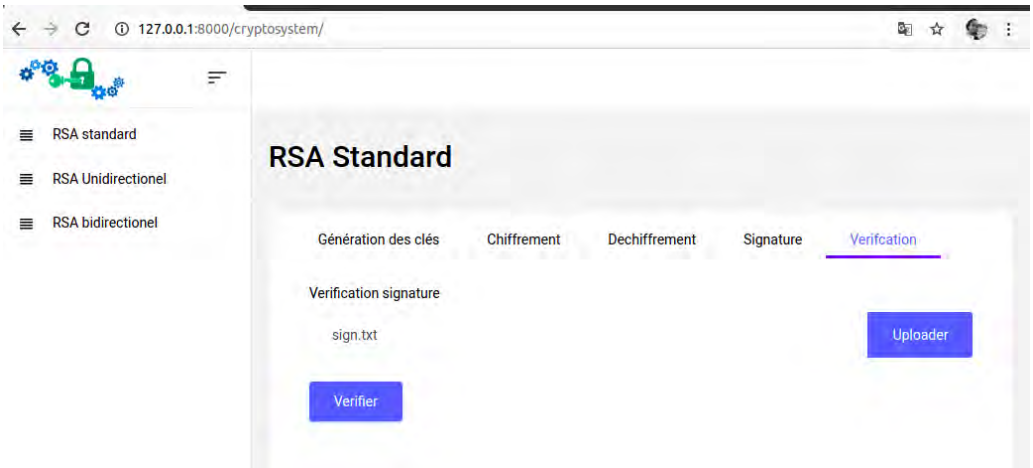


Figure 51 – Page Vérification de la signature

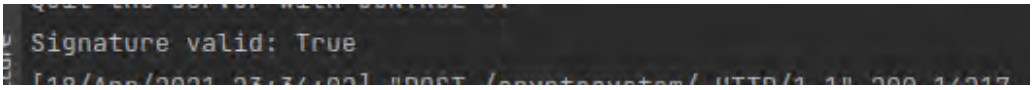


Figure 52 – Résultat de la signature

La signature est valide et retourne **true** car le message reçu est effectivement celui qui a été signé. Dans le cas contraire le résultat serait **false** .

3.3.2 RSA Unidirectionnel

a. **Page de gérération des clés**

On clic sur le bouton "Générer la paire de clé" deux fois et les clés sont générée dans les fichiers **publicuni1.pem**, **privateuni1.pem**, **publicuni2.pem** et **privateuni2.pem** ci-dessus.

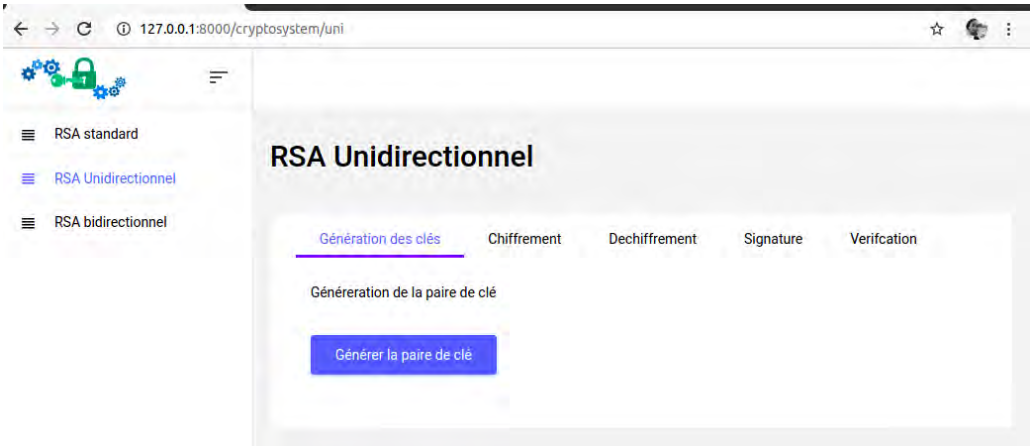


Figure 53 – Page de génération des clés

b. **Page de chiffrement du message**

On veut chiffrer le message se trouvant dans le fichier "plaintext.txt" avec les deux clés publiques

Maintenant on choisit le fichier "plaintext.txt" en cliquant sur le bouton "Uplaod", on clic sur le bouton "Chiffrer" et le message est chiffré et gardé dans le fichier "encrypt1.txt" ci-dessous avec la première clé public qui se trouve dans le fichier **publicuni1.pem**

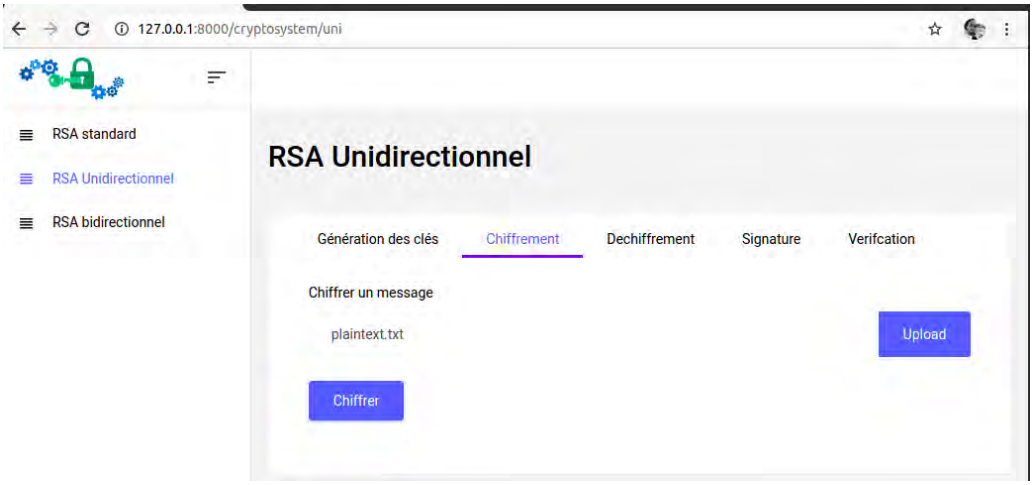


Figure 54 – Page de chiffrement du message stocké dans plaintext.txt

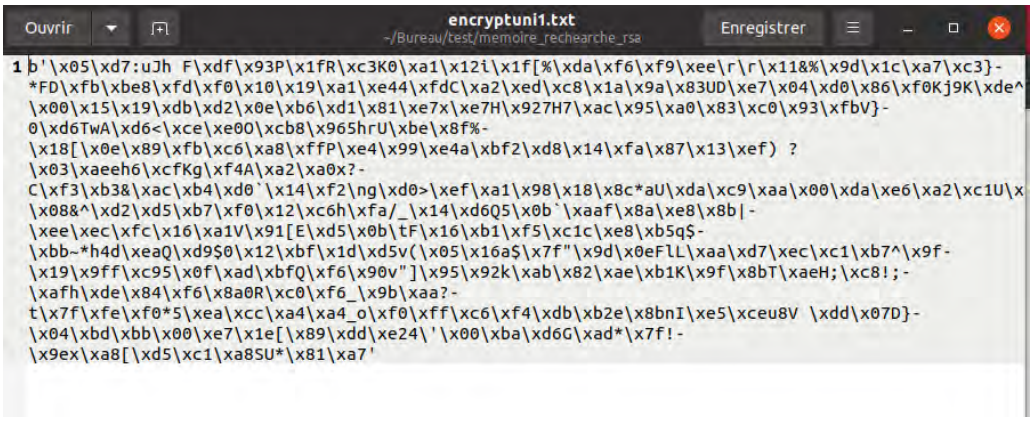


Figure 55 – Le fichier encryptuni1.txt

Ensuite on clic sur le bouton "Upload" et on choisit le fichier "encryptuni1.txt", on clic sur "Chiffrer" et le contenu du fichier est chiffré de nouveau avec la deuxième clé public stocké dans "publicuni2.txt" et est gardé dans le fichier "encryptuni2.txt".

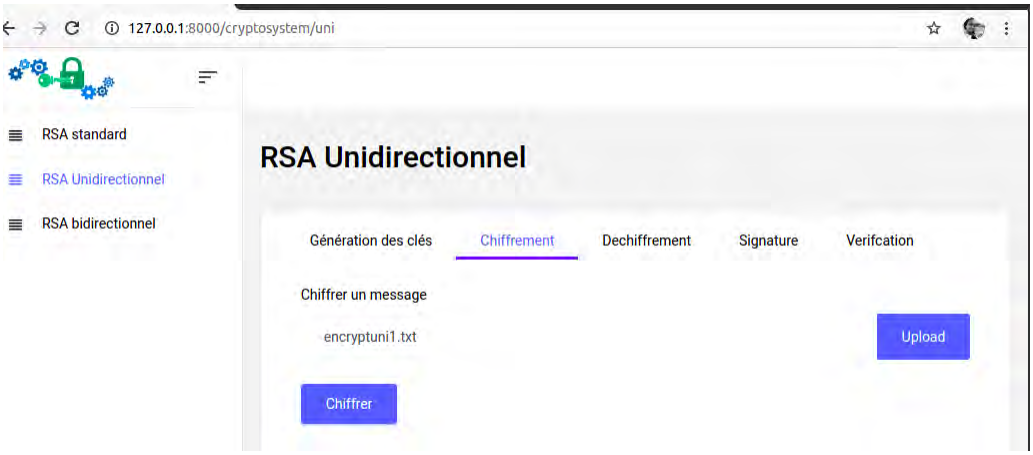


Figure 56 – Page de chiffrement du message stocké dans encryptuni1.txt

Figure 57 – Le fichier encryptuni2.txt

c. **Page de déchiffrement du message stocké dans "encryptuni2.txt"**

On clic sur le bouton "Upload", ensuite on choisit le fichier "encryptuni2.txt" et on clic sur le bouton "Déchiffrer" pour le premier déchiffrement. Le contenu du fichier est déchiffré avec la deuxième clé privée stockée dans "privateuni2.pem" et le chiffré est gardé dans le fichier "decrypt1.txt"

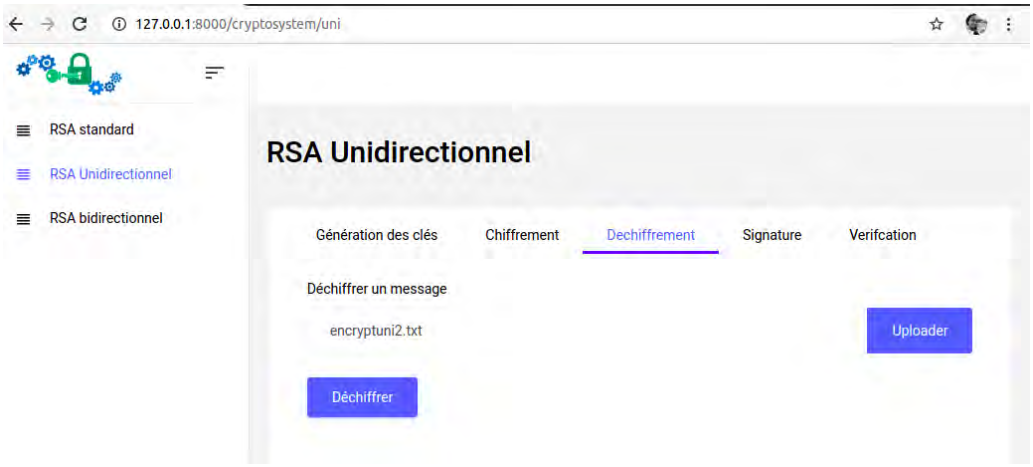


Figure 58 – Page de déchiffrement du message stocké dans encryptuni2.txt

Figure 59 – Le fichier decrypt1.txt

d. **Page de déchiffrement du message stocké dans decrypt1.txt**

on clic sur "Uplaod", on choisit le fichier "décrypt1.txt et on clic sur le bouton "Déchiffrer". Le contenu du fichier est déchiffré avec la premiere clé(privateuni1.pem) et est gardé dans le fichier **decrypt2.txt**

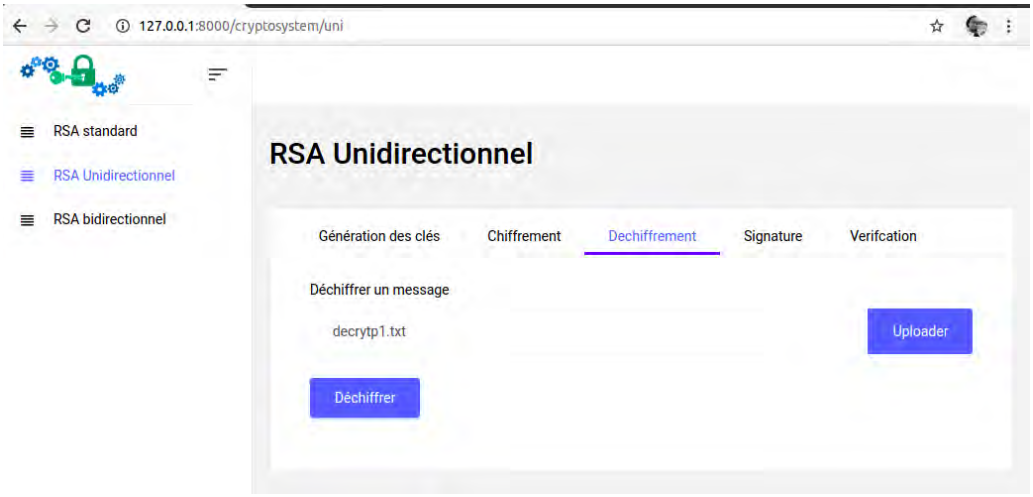


Figure 60 – Page de déchiffrement du message stocké dans "décrypt1.txt"

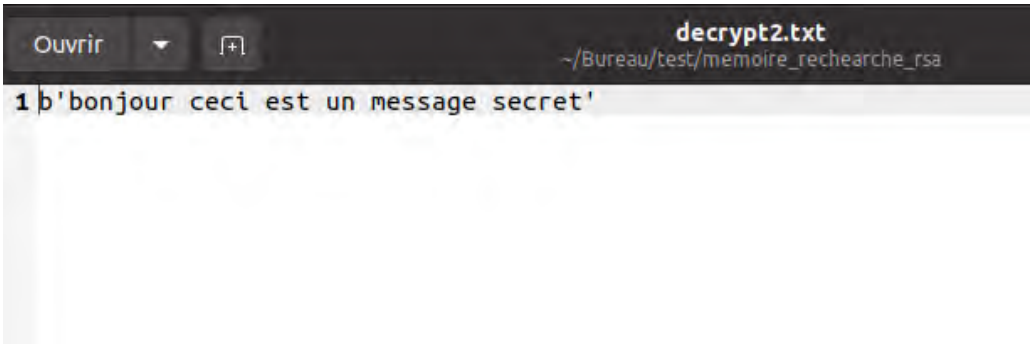


Figure 61 – Le fichier decrypt2.txt

Conclusion :

Dans ce chapitre nous avons utilisé le framework Django et le langage python avec l'algorithme RSA afin de développer une application web pour le RSA standard et RSA unidirectionnel qui nous permet de faire le chiffrement, le déchiffrement, la signature et la vérification des données.

.

Conclusion et Perspective

Ce mémoire de fin d'étude avait pour ambition de déléguer à une personne de déchiffer ou de signer à la place d'une autre personne sans connaitre la clé secrète, mandater une personne sans que cette derniére ne puisse abuser.

«Schéma uni(bi)directionnel : le cas de RSA» a fait l'objet de notre étude. En dépit de l'abondance de la matière qu'il déborde, nous avons orienté cette étude sur l'axe de cryptosystème RSA et la sécurité des données, car ces derniers constituent la matière première de la cryptographie.

Notre travail a porté sur trois chapitres dont le premier intitulé « Proxy Unidirectionnel et Bidirectionnel» nous donne une vue d'ensemble sur la généralité de la cryptographie et les enjeux de la sécurité informatique.

Le deuxième chapitre intitulé « RSA Unidirectionnel et Bidirectionnel » nous avons fait la présentation du chiffrement et la signature RSA standard et le chiffrement, Signature uni(bi)directionnel basé sur RSA. Ce chapitre introduit les notions de base de la sécurité de RSA.

Enfin, dans le troisième chapitre qui est le dernier, « Application de RSA standard et RSA Unidirectionnel », nous étalons nos capacités. Nous avons dans un premier temps présenté l'implémentation de RSA standard, ensuite une implémentation de RSA unidirectionnel avec une interface graphique permettant de crypter et de décrypter les informations.

La réalisation de ce travail nous a permis d'exploiter et améliorer nos connaissance, sur le framework Django de Python, découvrir la notion de chiffrement et de signature uni(bi)directionnel, vaste domaine de la conception et de la réalisation d'une application web en approfondissant nos connaissances. Ce projet a été très bénéfique pour nous dans la mesure où il nous a permis de parcourir un peu tous les domaines de notre formation (Transmission des Données et Sécurité de l'Information) en particulier.

En perspectives, nous pouvons élargir notre travail en finalisant notre application en faisant le chiffrement et la signature bidirectionnel basé sur RSA. Ainsi ce Projet pourrait faire l'objet d'une étude comparative entre les cryptosystèmes RSA et ElGamal.