Clé Publique

1 Chiffrement

1.1 Le chiffrement RSA

1.1.1 La fonction RSA

Soit $n = pq$ le produit de deux grands nombres premiers p et q. On note $\phi(n)$ la fonction d'Euler :

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

Soit e un nombre premier avec $\phi(n)$. D'après le théorème de Bézout il existe d tel que :

$$ed \equiv 1 \quad (\phi(n)).$$

- La **clé publique** est (n, e) où n est le **module**, et e **l'exposant de chiffrement**.
- La **clé privée** est **l'exposant de déchiffrement** d.

Soit $0 \leq x < n$. On définit la fonction RSA par :

$$RSA_{n,e}(x) = x^e \mod n.$$

Théorème 1.1 *La fonction $RSA_{n,e}$ est une* **permutation** *de l'ensemble $\{0, \cdots, n-1\}$ et son inverse est donné par :*

$$RSA_{n,e}^{-1}(y) = y^d \mod n.$$

Preuve. Soit $y = x^e \mod n$ un point de l'image de la fonction RSA. Calculons :

$$y^d \mod n = x^{de} \mod n = x^{1+k\phi(n)} \mod n = x \mod n = x.$$

Donc la fonction est injective et comme $\{0, \cdots, n-1\}$ est un ensemble fini, la fonction est bijective. De plus, le calcul fait montre que $y^d \mod n$ est la fonction inverse de la fonction $RSA_{n,e}$. □

Remarquons que dans la mise en place du système on peut remplacer dans tous les calculs $\phi(n) = (p-1)(q-1)$ par $\lambda(n) = ppcm(p-1, q-1)$.

On espère que la fonction RSA soit un **bon candidat pour être une fonction à sens unique**. La fonction RSA possède une **trappe** : toute personne connaissant la clé publique (n, e) et la factorisation de n peut calculer d.

En ce qui concerne la fonction RSA, la construction de n, e, d, le calcul de la fonction, la sécurité, de nombreux problèmes se posent :

- **Quelle taille doit-on prendre pour n ?** En pratique n doit avoir au moins 1024 bits.
- **Y-a-t-il d'autres conditions sur n ?** La sécurité de RSA repose pour une grande part sur la difficulté de factoriser. Ainsi certains n possédant des propriétés particulières sont interdits. Par exemple par la méthode $p-1$ de Pollard, on sait que si $p-1$ est le produit de petits facteurs, on peut factoriser n. Mais ces cas particuliers ne se produisent, lorsqu'on prend des nombres premiers au hasard, qu'avec une probabilité négligeable. Cependant on peut inclure des tests dans les choix de p et q pour être sûr d'écarter ces cas défavorables.
- **Comment construire des grands nombres premiers ?** La méthode la plus rapide est d'utiliser un test probabiliste de primalité (par exemple celui de Miller-Rabin). Avec un tel test on n'est pas absolument sûr que p soit premier, mais la probabilité de déclarer premier un nombre composé est très faible.
- **Comment déterminer e ?** Une façon de faire est de choisir e au hasard et de tester si $pgcd\,(e, \phi(n)) = 1$. Si ce n'est pas le cas on fait un nouveau choix. Mais ce n'est pas la seule façon, on peut par exemple construire un nombre premier strictement plus grand que le plus grand des deux nombres p et q et plus petit que $\phi(n)$. Le nombre

e construit est alors certainement premier avec $\phi(n)$. On peut aussi vouloir fixer un e à priori, par exemple $e = 3$; on construit alors p et q de telle sorte que e ne divise ni $p-1$ ni $q-1$. Toutes ces méthodes donnent en pratique un résultat en temps raisonnable.

- **Comment calculer** d**?** Connaissant le module n et l'exposant de chiffrement e, l'exposant de déchiffrement d peut être calculé à partir de l'algorithme d'Euclide étendu.
- **Comment calculer** $x^e \mod n$ **et** $y^d \mod n$**?** Ces calculs se font par un algorithme du type "square and multiply". Remarquons qu'il est possible d'accélérer le calcul de $y^d \mod n$, car le propriétaire de la clé privée d connaît aussi la factorisation de n. On peut donc dans ce cas appliquer l'algorithme de calcul de puissance conjugué avec le théorème des restes chinois. Cette méthode divise à peu près par un facteur 4 le temps de calcul de la puissance modulo n.
- **Que peut-on dire de la sécurité sémantique ?** Intuitivement la sécurité sémantique signifie qu'aucune information ne peut être calculée en pratique (c'est-à-dire en temps moyen polynomial) à partir du texte chiffré. Un chiffrement déterministe ne peut être sémantiquement sûr. Pour la fonction RSA nous pouvons citer facilement une information qui se calcule en temps polynomial à partir du chiffré. Par exemple si e est impair, le symbole de Jacobi de $y = x^e \mod n$ est égal au symbole de Jacobi de x :

$$\left(\frac{x^e}{n}\right) = \left(\frac{x^e}{p}\right)\left(\frac{x^e}{q}\right) = \left(\frac{x}{p}\right)\left(\frac{x}{q}\right) = \left(\frac{x}{n}\right).$$

Donc nous aurons quelques précautions à prendre dans l'utilisation de la fonction RSA.

D'un autre côté le texte chiffré ne laisse pas fuir certaines informations importantes. Par exemple la parité du texte clair est aussi difficile à déterminer que tout le texte clair lui-même.

Théorème 1.2 *Soit f la fonction définie par :*

$$f(y) = \begin{cases} 0 & si\ RSA_{n,e}^{-1}(y)\ est\ pair, \\ 1 & si\ RSA_{n,e}^{-1}(y)\ est\ impair. \end{cases}$$

Si f peut être calculée en pratique, il est possible de déchiffrer tout message.

Théorème 1.3 *Soit g la fonction définie par :*

$$g(y) = \begin{cases} 0 & si\ RSA_{n,e}^{-1}(y) \leq n/2, \\ 1 & si\ RSA_{n,e}^{-1}(y) > n/2. \end{cases}$$

Si g peut être calculée en pratique, il est possible de déchiffrer tout message.

Preuve. Dans un premier temps nous montrons que calculer f est équivalent à calculer g. Pour cela soit $0 \leq x < n$ et soit y le chiffré de x, c'est-à-dire $y = RSA_{n,e}(x) = x^e \mod n$. Posons $y_1 = 2^e y \mod n$ et $x_1 = 2x \mod n$. Avec ces notations on a la suite d'égalités :

$$RSA_{n,e}(x_1) = (2x)^e \mod n,$$

$$RSA_{n,e}(x_1) = 2^e y \mod n = y_1.$$

Donc :

$$y_1 = RSA_{n,e}(x_1) = RSA_{n,e}(2x \mod n).$$

Si $0 \leq x \leq n/2$ alors $x_1 = 2x$ donc x_1 est pair. Si $n/2 < x < n$ alors $x_1 = 2x - n$, donc x_1 est impair (car n est impair). Par exclusion on a bien entendu les réciproques. Donc :

$$f(2^e y \mod n) = f(y_1) = g(y).$$

Posons $y_2 = 2^{-e} y \mod n$ (2^e est inversible modulo n donc on peut faire intervenir 2^{-e}) et calculons $g(y_2)$ en utilisant la formule démontrée précédemment :

$$g(y_2) = f(2^e y_2 \mod n) = f(y)$$

si bien que :

$$g(2^{-e} y \mod n) = f(y).$$

En conclusion, si on a un algorithme polynomial pour calculer f on a un algorithme polynomial pour calculer g et réciproquement.

Montrons maintenant l'existence d'un algorithme polynomial en la taille de n qui utilise des appels à la fonction g, c'est-à-dire qui utilise g comme oracle, et qui permet de retrouver x connaissant y, e et n. On pose pour tout entier $k \geq 0$:

$$y_k = RSA_{n,e}(2^k x \mod n),$$

$$I_k = \bigcup_{q=0}^{2^{k-1}-1} \left[\frac{2qn}{2^k}, \frac{(2q+1)n}{2^k}\right[.$$

On a :

$$y_k = (2^k x)^e \mod n = 2^{ke} x^e \mod n = 2^{ke} y \mod n.$$

Si $x \in I_{k+1}$ alors il existe q tel que :

$$\frac{2qn}{2^{k+1}} \leq x < \frac{(2q+1)n}{2^{k+1}},$$

$$qn \leq 2^k x < qn + \frac{n}{2}.$$

Si bien que :

$$0 \leq 2^k x \mod n < \frac{n}{2},$$

$$0 \leq x_k < \frac{n}{2},$$

et donc $g(y_k) = 0$.
De même si $x \notin I_{k+1}$, alors il existe q tel que :

$$\frac{(2q+1)n}{2^{k+1}} < x < \frac{2(q+1)n}{2^{k+1}},$$

$$qn + \frac{n}{2} < 2^k x < (q+1)n.$$

Si bien que :

$$\frac{n}{2} < 2^k x \mod n < n,$$

$$\frac{n}{2} < x_k < n,$$

et donc $g(y_k) = 1$.

On vient donc de montrer que $x \in I_{k+1}$ si et seulement si $g(y_k) = 0$, et bien entendu aussi que $x \notin I_{k+1}$ si et seulement si $g(y_k) = 1$.
Si y est donné on peut calculer en temps polynomial pour chaque k la valeur y_k puis $g(y_k)$. On peut donc savoir pour chaque cas si $x \in I_{k+1}$ ou non. Posons alors :

$$A_N = \bigcap_{k=1}^{N} \chi_k$$

où

$$\chi_k = \begin{cases} I_k \\ ou \\ \complement I_k \end{cases}$$

suivant l'appartenance ou non de x_k à I_k. L'ensemble A_N est un intervalle de la forme :

$$A_N = \left[\frac{qn}{2^N}, \frac{(q+1)n}{2^N}\right[.$$

Pour $N > \log_2(n)$, la longueur de cet intervalle est < 1 et il contient au plus un entier (qui est le x cherché). □
Un bit de texte clair aussi difficile à calculer que le message tout entier est appelé un **hard-core bit**.

- **Est-il possible de casser RSA sans factoriser le module ?** La réponse à cette question n'est pas connue. Donnons quelques résultats reliés à ce problème.
Théorème 1.4 *Il est équivalent de connaître $\phi(n)$ ou la factorisation de n.*
Preuve. Si p et q sont connus il est facile de calculer en temps polynomial $\phi(n) = (p-1)(q-1)$.
Réciproquement on dispose des relations suivantes :

$$(p-1)\left(\frac{n}{p} - 1\right) = \phi(n),$$

$$p^2 - (n + 1 - \phi(n))\, p + n = 0.$$

On peut résoudre cette équation en temps polynomial. □

Théorème 1.5 *Il y a un* **algorithme probabiliste** *ayant pour entrées le module n et l'exposant de chiffrement e, et qui renvoie la factorisation pq de n, s'exécutant en* **temps probabiliste polynomial** *(algorithme de Las Vegas) et* **qui fait appel à un oracle** *qui fournit d.*

Preuve. Remarquons tout d'abord que :

$$x^2 \equiv 1 \quad (n)$$

si et seulement si :

$$\begin{cases} x^2 \equiv 1 \quad (p) \\ et \\ x^2 \equiv 1 \quad (q). \end{cases}$$

Ces équations sont vérifiées si et seulement si :

$$\begin{cases} x \equiv \pm 1 \quad (p) \\ et \\ x \equiv \pm 1 \quad (q) \end{cases}$$

L'équation initiale posséde quatre solutions, deux d'entre elles sont les solutions triviales vérifiant $x \equiv \pm 1 \quad (n)$.

Si x est une solution non triviale, n divise $(x+1)(x-1)$ mais ne divise ni $(x-1)$ ni $(x+1)$. Donc :

$$pgcd(x+1, n) = p \; ou \; q,$$

$$pgcd(x-1, n) = q \; ou \; p.$$

Donc, si nous connaissons une racine carrée non triviale 1 modulo n, nous pouvons calculer en temps polynomial la factorisation de n.

Maintenant le problème qui se pose est : Comment calculer une racine carrée non triviale de 1 modulo n ?

Choisissons au hasard w tel que $1 < w \leq n-1$. Si $pgcd(w, n) > 1$, nous avons un facteur premier de n.

Sinon, nous écrivons (n'oublions pas que d est connu) :

$$ed - 1 = 2^s r,$$

où $s \geq 1$ et r est impair. Nous savons que :

$$w^{2^s r} = w^{ed-1} = w^{k\phi(n)},$$

et par suite :

$$w^{2^s r} \equiv 1 \quad (n).$$

Donc, il existe un plus petit t tel que $t \leq s$ et tel que :

$$w^{2^t r} \equiv 1 \quad (n).$$

Notons t_0 cet entier. Si $t_0 > 0$ définissons :

$$v_0 = w^{2^{t_0-1} r},$$

de telle sorte que :

$$v_0 \not\equiv 1 \quad (n)$$

et que :

$$v_0^2 \equiv 1 \quad (n).$$

Maintenant si :

$$v_0 \not\equiv -1 \quad (n),$$

nous avons trouvé une racine carré non triviale de 1 modulo n. L'algorithme échoue, et nous devons le réappliquer à un autre w dans les deux cas suivants :

a) $t_0 = 0$, c'est-à-dire :

$$w^r \equiv 1 \quad (n);$$

b) il existe t tel que $0 \leq t \leq s-1$ et tel que :

$$w^{2^t r} \equiv -1 \quad (n).$$

Le nombre des w pour lesquels cet algorithme échoue est d'après le théorème de Rabin majoré par $\frac{\phi(n)}{4}$. Donc la probabilité d'obtenir le résultat avec un w donné est $\geq 3/4$. Si nous ne pouvons pas obtenir le résultat avec ce w on en tire un autre. La probabilité d'un nombre infini d'itérations est nulle. De plus l'espérance du nombre

d'itérations est $\leq 4/3$. Maintenant nous pouvons conclure sachant que chaque itération a un coût polynomial.□

D'un autre côté si e et la factorisation de n sont connus, nous pouvons calculer d. Ainsi quand nous avons une fonction RSA, calculer l'exposant de déchiffrement d est **en pratique** aussi dur que factoriser n.

Mais ceci n'exclut pas la possibilité de pouvoir calculer l'inverse $RSA_{n,e}^{-1}$ sans utiliser d.

1.1.2 Schéma de chiffrement et de déchiffrement RSA

Un texte clair étant donné sous forme d'une chaîne de caractères, comment le chiffrer en utilisant la fonction RSA ? La difficulté de l'inversion de RSA n'est pas suffisante, loin de là, pour garantir la sécurité du système global. Prenons l'exemple naïf suivant : supposons le texte clair codé en ASCII et chiffrons un par un tous les caractères ASCII en élevant leurs codes à la puissance e, modulo n. Bien entendu ce système peut se casser facilement par une attaque statistique simple. Dans ce cas, le **schéma de chiffrement est mauvais**.

Nous présentons ici, le schéma RSAES-OAEP, recommandé dans la directive PKCS♯1v2.1 de la société RSA : **RSA Cryptography Standard** (January 5, 2001). (RSAES : **RSA E**ncryption **S**cheme. OAEP : **O**ptimal **A**symmetric **E**ncryption **P**adding).

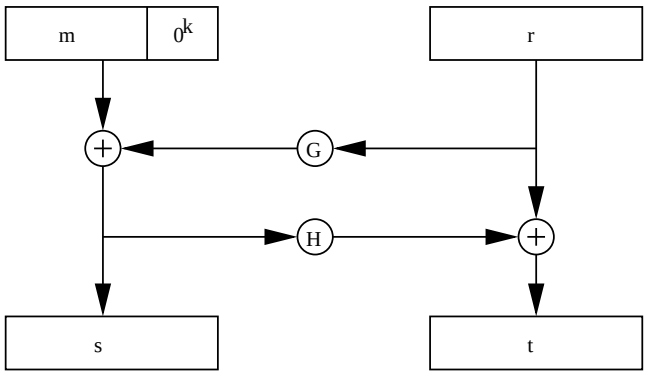

FIG. 1 – Fonctionnement de OAEP

Cette directive met en place le modèle OAEP introduit par Bellare et

Rogaway, dont nous donnons une description sur la figure 1.1.2. Dans ce modèle, le message est découpé en blocs de taille fixée complétés par des zéros (voir les détails plus loin, dans la description de l'implémentation). Notons maintenant M un tel bloc (M contient donc une partie message m complétée par des zéros). Par ailleurs on dispose d'un générateur aléatoire G et d'une fonction de hachage H. On tire au sort un nombre r de la taille commune du germe du générateur aléatoire G et de la sortie de la fonction de hachage h. On calcule alors :

$$s = M \oplus G(r),$$

$$t = H(s) \oplus r.$$

C'est le couple (t, s) qui est transformé en nombre et chiffré par la fonction RSA. Le système devient alors un système de chiffrement aléatoire. De plus il a été montré que OAEP combiné avec RSA est sémantiquement sûr contre une attaque CCA2. Mais attention, OAEP combiné avec une autre fonction à sens unique avec trappe est toujours sûr contre une attaque CCA1 , mais pourrait ne pas être sûr contre une attaque CCA2. On peut déchiffrer facilement ce système (si bien entendu on connaît la clé). En effet à partir de (s, t) on peut calculer successivement $H(s)$, $r = H(s) \oplus t$, $G(r)$, $M = s \oplus G(r)$.
Entrons maintenant dans les détails de l'implémentation. Supposons que nous partions d'un bloc de texte clair M (le message à chiffrer) qui est une suite d'octets de longueur $mlen$.

- **Encodage**. Cette suite d'octets M est transformée en une suite d'octets EM de longueur $emlen$. Le bloc EM est le **message encodé**. La transformation utilisée sera décrite plus tard. C'est **l'encodage** EME-OAEP. (EME : **E**ncoding **M**ethod for **E**ncryption. OAEP : **O**ptimal **A**symmetric **E**ncryption **P**adding).
- **Transformation en entier**. Puis, EM est transformé en un entier m par la procédure OS2IP (Octet String To Integer Primitive).
- **Chiffrement**. L'entier m est chiffré en c par la fonction RSAEP (RSA Encryption Primitive). Cette transformation est simplement l'application de la fonction RSA à l'entier m.

- **Transformation en suite d'octets** . L'entier c est transformé en une suite d'octets C par la procédure I2OSP (Integer To Octet String Primitive).

Quand le destinataire reçoit C il applique le schéma de déchiffrement suivant :

- **Transformation en entier**. C est transformé en un entier c par la procédure OS2IP (Octet String To Integer Primitive).
- **Déchiffrement** L'entier c est transformé en m par RSADP (RSA Decryption Primitive). Cette transformation n'est rien d'autre que la fonction RSA^{-1}.
- **Transformation en suite d'octets**. L'entier m est transformé en une suite d'octets EM par la procédure I2OSP (Integer To Octet String Primitive).
- **Décodage**. La suite d'octets EM est transformée en une suite d'octets M par la procédure de décodage de EME-OAEP.

La procédure OS2IP est très simple. Si $X_1, X_2, \cdots, X_l$ sont les octets de la chaîne, soit x_{l-i} l'entier représenté par l'octet X_i. Posons :

$$x = \sum_{j=0}^{l-1} x_j (256)^j;$$

c'est la sortie de la procédure OS2IP.

La procédure I2OSP, qui est l'inverse de la procédure OS2IP, correspond à la décomposition en base 256 de l'entier auquel s'applique la procédure.

Les procédures EME-OAEP pour l'encodage et pour le décodage utilisent une **fonction de hachage** h et une **fonction de génération de masque** g (c'est en fait un générateur aléatoire). L'entrée de la fonction h est une suite d'octets et sa sortie est une suite d'octets de longueur fixée $hlen$. La fonction g a pour entrée une suite d'octets et un nombre l, et pour sortie une suite d'octets de longueur l.

Pour la partie encodage, on part d'une suite M (message à encoder) et P un paramètre d'encodage public (qui est aussi une suite d'octets).

On construit une suite PS ayant $emlen-mlen-2hlen-1$ octets nuls. On calcule $pHash = h(P)$ une chaîne de longueur $hlen$. On concatène alors $pHash$, PS, l'octet $01H = 00000001$ et M pour obtenir DB :

$$DB = pHash||PS||01H||M,$$

où $||$ désigne l'opération de concaténation. Remarquons que la longueur de DB est $emlen - hlen$.
Maintenant on tire au hasard une chaîne s de longueur $hlen$ et on calcule $dbmask = g(s, emlen - hlen)$. On détermine successivement :

$$maskedDB = DB \oplus dbmask,$$

$$smask = g(maskedDB, hlen),$$

$$maskeds = s \oplus smask,$$

$$EM = maskeds||maskedDB.$$

La sortie est EM.
Pour la partie décodage on part de la chaîne EM (le message encodé) et du paramètre d'encodage P qui est public. Les $mlen$ premiers octets de EM nous donnent $maskeds$, les autres constituant $maskedDB$. Nous calculons alors successivement :

$$smask = g(maskedDB, hlen),$$

$$s = smask \oplus maskeds,$$

$$dbmask = g(s, emlen - hlen),$$

$$DB = dbmask \oplus maskedDB.$$

Il est maintenant facile de reconstituer M.

1.1.3 Attaques de RSA

Nous ne donnerons pas ici une liste complète des attaques connues contre RSA, ni les détails, avec les différentes variantes de ces attaques. Nous donnerons seulement quelques résultats généraux et quelques

exemples. Construire des mécanismes sûrs utilisant RSA n'est pas chose facile. En particulier un chiffrement naïf du texte clair est loin d'être suffisant. On doit faire un pré-traitement des données, par exemple en utilisant OAEP, avant d'utiliser la fonction RSA.
Remarquons que **tout algorithme efficace de factorisation** donne un moyen de casser la fonction RSA. Nous renvoyons au livre de H. Cohen [**?**] pour une étude complète des méthodes de factorisation.

- **Module commun.** C'est une attaque très simple et très connue, qui s'applique si deux utilisateurs ont le même module. Dans ce cas notons (n, e_A) la clé publique de A et (n, e_B) la clé publique de B. Le premier point est que chacun des utilisateurs A et B peut trouver la clé privée de l'autre. En effet A par exemple, connaissant la paire (e_A, d_A), peut factoriser n, puis connaissant e_B peut calculer d_B. Supposons maintenant que e_A et e_B soient premiers entre eux. On peut trouver x et y tels que :

  $$xe_A + ye_B = 1.$$

  Si un expéditeur C envoie un message m à A et B, il envoie :

  $$c_A = m^{e_A} \mod n,$$

  $$c_B = m^{e_B} \mod n.$$

  Alors tout le monde est capable de retrouver m par la formule :

  $$m = c_A^x c_B^y \mod n.$$

  En conclusion, le chiffrement RSA ne supporte pas que deux utilisateurs aient le même module.
- **Petit exposant privé.** On peut penser, afin d'accélérer les calculs de déchiffrement, surtout lorsque ceux-ci sont effectués par de petits systèmes (cartes à puce par exemple), à choisir un petit exposant de déchiffrement d. Mais ceci n'est pas un bon choix. Il existe, dans ce cas, une attaque dûe à Wiener, basée sur les développements en fraction continue. Nous partons de la relation :

$$ed \equiv 1 \quad (\phi(n)),$$

où $\phi(n) = (p-1)(q-1) = n - (p+q) + 1$. Cette relation peut être mise sous la forme :

$$ed = 1 + k\left(\frac{n+1}{2} - \frac{p+q}{2}\right),$$

où k est un entier. De cette relation nous tirons :

$$\left|\frac{2e}{n} - \frac{k}{d}\right| = \frac{|2 + k(1-(p+q))|}{nd},$$

de telle sorte que si $\frac{|k(p+q-1)-2|}{n} < \frac{1}{2d}$ nous obtenons :

$$\left|\frac{2e}{n} - \frac{k}{d}\right| < \frac{1}{2d^2}.$$

Cette inégalité montre que k/d est une réduite du développement en fraction continue de $2e/n$. Si p et q ont la même taille, $p + q = \mathcal{O}(\sqrt{n})$. De plus comme $k = \mathcal{O}(d)$, nous concluons que si $d = \mathcal{O}(n^{0,25})$ cette attaque fonctionne. Nous faisons alors un développement en fraction continue de $2e/n$, et pour chaque réduite nous regardons si le dénominateur est le d cherché en essayant avec ce dénominateur de factoriser n.

Il existe des améliorations de l'attaque de Wiener, en particulier l'attaque de Boneh-Durfee qui permet de récupérer d, dès que $d < n^{0,292}$.

- **Petit exposant public, attaque par "broadcast".** Regardons tout d'abord une situation très simple. Considérons trois utilisateurs du système P_1, P_2, P_3, ayant pour clés publiques $(n_1, 3)$, $(n_2, 3)$, $(n_3, 3)$. Supposons que (n_1, n_2, n_3) soient premiers deux à deux (sinon on peut factoriser certains n_i). Si un expéditeur envoie un message m à P_1, P_2, P_3, et si $m < \inf(n_1, n_2, n_3)$ on peut obtenir par simple observation des communications :

$$c_1 = m^3 \quad (n_1),$$

$$c_2 = m^3 \quad (n_2),$$
$$c_3 = m^3 \quad (n_3),$$

et en utilisant l'algorithme des restes chinois :

$$c = m^3 \quad (n_1 n_2 n_3).$$

Mais $m^3 < n_1 n_2 n_3$, de telle sorte que $c = m^3$ dans $\mathbb{N}$. Pour retrouver m on a juste à extraire une racine cubique de c dans $\mathbb{N}$, ce qui est facile.

Considérons maintenant k utilisateurs $P_1, P_2, \cdots, P_k$, ayant des clés publiques $(n_1, e_1), (n_2, e_2), \cdots, (n_k, e_k)$ où les n_i sont premiers entre eux deux à deux. Supposons qu'un expéditeur envoie un message $m < n_{min} = \inf n_i$ aux k utilisateurs P_i. Pour éviter l'attaque précédente il parasite m avant de l'envoyer. Plus précisément il envoie à chaque participant P_i :

$$c_i = (f_i(m))^{e_i} \mod n_i,$$

où pour chaque i, f_i est un polynôme public normalisé de degré $deg(f_i)$, tel que l'application donnant c_i à partir de m soit injective. Posons :

$$g_i = f_i^{e_i} - c_i.$$

Si $k \geq \max e_i deg(f_i)$, un attaquant E qui a observé les échanges est capable de retrouver m. Pour ce faire, E utilise le théorème des restes chinois, et construit la fonction polynomiale :

$$g(x) = \sum_{i=1}^{k} u_i g_i(x),$$

où

$$u_i \equiv \begin{cases} 1 & (n_i), \\ 0 & (n_j) \quad si\ j \neq i. \end{cases}$$

Remarquons que le polynôme g est normalisé.

Soit $n = n_1 n_2 \cdots n_k$. Le message m est l'unique solution de :

$$g(x) \equiv 0 \quad (n)$$

telle que :

$$m < n_{min} < n^{\frac{1}{k}} \leq n^{\frac{1}{deg(g)}}.$$

L'attaquant E utilise alors le théorème suivant de Coppersmith :

Théorème 1.6 *Soit n un entier, et $g \in \mathbb{Z}[x]$ un polynôme normalisé de degré $deg(g)$. Posons $X = n^{\frac{1}{deg(g)} - \epsilon}$ pour un certain $\epsilon \geq 0$. Il est alors possible de calculer de manière efficace tous les entiers x tels que $g(x) = 0$ et $|x| < X$. Le temps d'exécution est majoré par le temps nécessaire à l'exécution de l'algorithme LLL sur un réseau de dimension $\mathcal{O}\left(\min\left(\frac{1}{\epsilon}, \log_2(n)\right)\right)$.*

- **Attaques par analyse de temps et de courant.** Pour une implémentation de RSA en carte à puce, une attaque dûe à Kocher qui consiste à mesurer précisément le temps nécessaire au déchiffrement RSA, permet de reconstituer la clé de déchiffrement. L'attaque prend en compte les algorithmes habituellement utilisés pour faire des multiplications, des élévations à la puissance d modulo n. Par exemple l'algorithme classique "élévation au carré et multiplication" pour élever à la puissance d modulo n, suivant la valeur du bit i de d fait une multiplication ou deux lors de la $i^{ème}$ boucle. Ceci peut être mis à profit, par mesure des temps d'exécution, pour estimer successivement les bits de d. À vrai dire, l'attaque n'est pas si commode que cela à mettre en place effectivement.
  Kocher a aussi monté une attaque basée sur des mesures de la consommation de courant appelée "**power cryptanalysis**".
  Ces attaques, basées sur des mesures de temps d'exécution, de consommation de courant, de rayonnement électromagnétique, sont appelées des **side channel attacks**.

1.2 Le chiffrement d'ElGamal

1.2.1 La fonction exponentielle et la fonction logarithme

Soit p un nombre premier et $(\mathbb{Z}/p\mathbb{Z})^*$ le groupe multiplicatif des entiers non nuls modulo p. Nous savons que ce groupe est cyclique. Soit α un élément primitif. Soit E_α une fonction de $(\mathbb{Z}/p\mathbb{Z})^*$ sur $(\mathbb{Z}/p\mathbb{Z})^*$ définie par :

$$E_\alpha(x) = \alpha^x \mod p.$$

L'inverse de E_α est l' **index** ou le **logarithme discret** i_α.
Calculer le logarithme discret d'un élément donné de $(\mathbb{Z}/p\mathbb{Z})^*$, c'est-à-dire le **problème du logarithme discret**, est connu pour être difficile. Plus précisément ce problème est NP. La fonction E_α est donc à sens unique, mais on ne lui connaît actuellement aucune trappe. Son utilisation pour un système cryptographique à clé publique ne sera pas tout à fait directe.

1.2.2 Le problème de Diffie-Hellman

Étant donnés :

$$\alpha^a \mod p,$$
$$\alpha^b \mod p,$$

est-il possible de calculer :

$$\alpha^{ab} \mod p.$$

Ce problème est le **problème de Diffie-Hellman**. Si nous pouvons résoudre en temps raisonnable le problème du logarithme discret, clairement nous pouvons résoudre le problème de Diffie-Hellman. La réciproque en général n'est pas connue, mais **pour certains nombres premiers particuliers** il a été démontré que si on sait résoudre le problème de Diffie-Hellman on peut aussi résoudre le problème du logarithme discret.

Définissons aussi le **problème de décision de Diffie-Hellman** : Supposons que $\alpha^a \mod p$, $\alpha^b \mod p$, et y sont donnés. A-t-on $y = \alpha^{ab}$? Là aussi on se rend tout de suite compte que si on sait résoudre le problème de Diffie-Hellman, alors on sait résoudre le problème décisionnel de Diffie-Hellman. Dans l'autre sens on ne sait en général rien dire, mais on dispose d'exemples de groupes où le problème de décision de Diffie-Hellman est facile et où on ne dispose pas d'algorithme simple pour le problème de Diffie-Hellman.

1.2.3 La fonction d'ElGamal

Soit p un grand nombre premier tel que $p-1$ ait un grand facteur premier q. Soit α un élément d'ordre q dans le groupe $G = (\mathbb{Z}/p\mathbb{Z})^*$. On appelle H le sous-groupe multiplicatif de $(\mathbb{Z}/p\mathbb{Z})^*$ engendré par α. Soit a un élément de $\mathbb{Z}/q\mathbb{Z}$ et $\beta = \alpha^a \mod p$ (qui est un élément de H). On définit la fonction d'ElGamal de $G \times (\mathbb{Z}/q\mathbb{Z})$ dans $H \times G$ par :

$$ELG_{p,q,\alpha,\beta}(x,k) = (y_1, y_2),$$

avec

$$y_1 = \alpha^k \mod p,$$

et

$$y_2 = x\beta^k \mod p.$$

Nous pouvons espérer que cette fonction soit à sens unique. En effet, si on peut calculer x connaissant y_1 and y_2, on peut calculer aussi $\beta^k = \alpha^{ak}$ connaissant α^a et α^k. Donc récupérer x revient à résoudre une instance du problème de Diffie-Hellman. Remarquons que ce n'est pas une instance générale dans G du problème de Diffie-Hellman car en fait tous les éléments α^a, α^k et α^{ak} sont dans le sous-groupe H de $(\mathbb{Z}/p\mathbb{Z})^*$. Mais en pratique, ceci ne semble pas plus facile que résoudre le problème pour une instance aléatoire dans $(\mathbb{Z}/p\mathbb{Z})^*$. C'est-à-dire que si par exemple p a 1024 bits et $q = 160$ bits, il ne semble pas plus facile

d'appliquer un algorithme générique de calcul du logarithme discret au groupe H (pour lequel il semble qu'on ne sache pas faire mieux) plutôt qu'appliquer un algorithme spécifique sous-exponentiel au groupe G. On peut dire que la fonction d'ElGamal possède en quelque sorte une trappe, car si quelqu'un connait le logarithme a de β, il peut aisément calculer x à partir de la relation :

$$y_2 = xy_1^a.$$

Il ne peut toutefois pas retrouver l'aléa k.
Remarquons qu'on peut penser à travailler directement sur le groupe G tout entier, en prenant pour α un élément primitif du groupe $G = (\mathbb{Z}/p\mathbb{Z})^*$ et sans introduire le sous-groupe H. Ceci oblige alors à utiliser des exposants plus gros sans gain apparent sur la sécurité. De plus, la détermination d'un élément primitif de G demande aussi la construction d'un nombre premier q grand qui divise $p - 1$.

1.2.4 Le chiffrement d'ElGamal

La fonction D'ElGamal nous permet de définir un système de chiffrement. En fait nous allons voir qu'on peut mettre en place diverses variantes de ce système.

a) Le chiffrement d'ElGamal simple qui utilise la fonction d'ElGamal telle que nous l'avons définie précédemment.

- **L'ensemble des textes clairs** est G.
- **L'ensemble des textes chiffrés** est $H \times G$.
- **La clé publique** de A est (p, q, α, β).
- **La clé privée** de A est a.
- **Le nombre aléatoire** k est tiré par l'expéditeur pour chaque message. Le nombre k ne peut pas être utilisé plus d'une fois.
- **Le chiffré** est calculé à partir du texte clair par la formule $y = ELG_{p,q,\alpha,\beta}(x, k)$.

Donc le chiffrement n'est pas déterministe. Le texte chiffré dépend non seulement du texte clair mais aussi du nombre aléatoire k.

On peut utiliser ce système sans introduire le sous-groupe H, en prenant pour α un élément primitif de G. Dans ce cas α est d'ordre $p-1$. On sait alors que

$$\alpha^{\frac{p-1}{2}} \equiv -1 \quad (p),$$

c'est-à-dire que le symbole de Legendre de α vérifie :

$$\left(\frac{\alpha}{p}\right) = -1.$$

Connaissant y_1 on peut, en calculant son symbole de Legendre, savoir si k est pair ou impair. Connaissant β, on peut alors calculer son symbole de Legendre, puis celui de β^k. Connaissant le symbole de Legendre de y_2 et celui de β^k, on trouve celui de x. De plus les symboles de Legendre des messages x se répartissent équitablement sur les deux valeurs 1 et -1. Donc l'information qu'on retrouve sur x à partir de son chiffré est significative. Le système, tout en étant aléatoire, n'est pas sémantiquement sûr.
Si on utilise le sous-groupe H d'ordre premier q et si on suppose que l'élément α est d'ordre q, alors le symbole de Legendre de α est 1. En effet, q est grand, donc ce n'est pas 2 ; par suite, le nombre premier q divisant $p-1$, divise aussi $\frac{p-1}{2}$; comme $\alpha^q = 1$ on conclut que $\alpha^{\frac{p-1}{2}} = 1$. Donc le symbole de Legendre de x est donné par le symbole de Legendre de y_2. Le système n'est pas non plus sémantiquement sûr. L'intérêt d'avoir introduit le sous-groupe H réside dans **l'utilisation d'exposants plus courts**.

b) Le chiffrement d'ElGamal modifié utilise la fonction d'ElGamal dont on **restreint l'espace des messages** au sous-groupe H. Dans ce cas tous les messages ont pour symbole de Legendre 1, ce qui ne permet plus de les distinguer. Notons toutefois au passage que se pose alors le problème de faire correspondre à un texte clair un élément de H.
Comme pour la fonction RSA nous pouvons nous poser plusieurs questions à propos de la fonction d'ElGamal.

- **Quelles tailles doivent avoir p et q ?** Pour la taille de p, c'est à peu près la même taille que celle du module utilisé par RSA. C'est-à-dire que p doit avoir au moins 1024 bits. Pour q, on prend une valeur ayant au moins 160 bits. Cette dernière valeur est celle utilisée pour la signature DSA.
- **Quelles sont les autres conditions sur p ?** La sécurité de la fonction ElGamal dépend pour une large part de la résistance du problème du logarithme discret. Donc certains p ayant des propriétés particulières sont interdits. Ici, par construction, $p-1$ possède un grand facteur premier, ce qui évite l'attaque de Pohlig-Hellman du problème du logarithme discret.
- **Comment construire explicitement p, q, α ?** La construction de p et q se fait en deux temps :
  - ▷ **Générer q.** On construit tout d'abord au hasard un nombre premier q de la taille voulue (en utilisant un test de primalité).
  - ▷ **Construire p.** On essaie de construire au hasard un multiple $p-1$ de q tel que p soit un nombre premier de la taille cherchée (on fait éventuellement plusieurs essais). Si cette partie échoue, on recommence avec une autre valeur de q.

  Cet algorithme aboutit en temps moyen de l'ordre de la taille désignée pour p.

  Pour ensuite trouver α d'ordre q nous pouvons procéder de la façon suivante : rappelons que $p-1$ est un multiple de q, c'est-à-dire que $p-1=sq$. Tirons au sort des éléments $0<\beta<p$ jusqu'à ce qu'on obtienne $\beta^s \not\equiv 1 \pmod{p}$. Posons alors $\alpha = \beta^s \mod p$. L'élément α ainsi construit est bien d'ordre q.
- **Que peut-on dire de la sécurité sémantique ?** Nous avons vu précédemment que le système n'est pas sémantiquement sûr. Si toutefois on modifie le système d'ElGamal en restreignant l'espace des messages au sous-groupe H alors on obtient la sécurité sémantique contre les attaques à texte clair choisi. Plus précisément si on suppose que l'adversaire ne peut disposer que des chiffrés des textes

clairs de son choix, c'est-à-dire est attaquant de type CPA , alors la sécurité sémantique du chiffrement d'ElGamal **modifié** est équivalente au problème de décision de Diffie-Hellman. Mais nous verrons que si l'attaquant est plus puissant (CCA1 ou CCA2) alors ce système n'est plus sémantiquement sûr.

- **Dans le cas du système d'ElGamal modifié comment faire correspondre à un texte clair un élément de H ?** Dans le cas général cela semble difficile. Mais si on prend un couple (p, q) (p et q premiers) tel que $p = 2q + 1$ alors H est le groupe des éléments qui sont des résidus quadratiques modulo p. Le nombre q est un nombre premier de Sophie Germain. Dans ce cas -1 n'est pas un résidu quadratique modulo p car $\frac{p-1}{2}$ est impair. On se restreint alors aux messages m tels que $0 \leq m < q = \frac{p-1}{2}$. Si m est un résidu quadratique modulo p on le laisse tel quel. Sinon on le transforme en $p-m$, ce qui donne de nouveau un résidu quadratique puisque -1 n'est pas un résidu quadratique.
- **Que faire sur des messages courts ?** Il est clair que si on diminue la taille des messages, on affaiblit le système. Dans le cas où on est amené à ne chiffrer que des messages courts, il faut utiliser conjointement un système de "padding", c'est-à-dire un système qui rajoute au message un certain nombre de bits.

On sait que pour le groupe multiplicatif $(\mathbb{Z}/pZ)^*$ il existe **un algorithme sous-exponentiel** qui résout le problème du logarithme discret. Travailler dans le groupe multiplicatif d'un corps fini général n'accroît pas cette complexité. D'un autre côté actuellement nous ne connaissons aucun algorithme sous-exponentiel pour résoudre le problème du logarithme discret sur le groupe d'une courbe elliptique bien choisie. Avec la cryptographie elliptique la taille de la clé peut être plus courte (de l'ordre de 200 bits). Mais attention pour bénéficier de cette sécurité on doit être prudent dans le choix du corps sur lequel est défini la courbe elliptique et de la courbe elle même. Des courbes utilisables ont été publiées dans divers documents de normalisation et en particulier

dans la note **FIPS 186-2** (**F**ederal **I**nformation **P**rocessing **S**tandard) du **NIST** (**N**ational **I**nstitute of **S**tandards and **T**echnology).

1.2.5 Amélioration théorique : le chiffrement de Cramer-Shoup

On reprend le groupe $G = (\mathbb{Z}/p\mathbb{Z})^*$ où p est un grand nombre premier. Un utilisateur tire au sort deux éléments α_1 et α_2 de G de telle sorte que tous les éléments de G soient équiprobables dans ce tirage. Il prend aussi au hasard cinq entiers a_1, a_2, b_1, b_2, z de l'intervalle $[0, p-1[$ et on calcule

$$c = \alpha_1^{a_1}\alpha_2^{a_2} \quad d = \alpha_1^{b_1}\alpha_2^{b_2} \quad h = \alpha_1^z.$$

Il dispose par ailleurs d'une fonction de hachage H de G^3 dans l'intervalle $[0, p-1[$. La clé publique de l'utilisateur est $(\alpha_1, \alpha_2, c, d, h, H)$. Sa clé privée est (a_1, a_2, b_1, b_2, z).

- **Chiffrement :** pour chiffrer un message m à destination de l'utilisateur en question dont on connaît la clé publique, on tire au sort un entier $k \in [0, p-1[$ et on calcule successivement $u_1 = \alpha_1^k, u_2 = \alpha_2^k, e = h^k m, u = H(u_1, u_2, e), v = c^k d^{ku}$. Le texte chiffré est (u_1, u_2, e, v).
- **Déchiffrement :** supposons que le message chiffré soit (u_1, u_2, e, v). Le propriétaire de la clé privée calcule alors $u = H(u_1, u_2, e)$, puis si $v = u_1^{a1+b_1u}u_2^{a2+b_2u}$ alors il récupère le message $m = e/u_1^z$, sinon le texte chiffré est invalide (ce n'est le chiffré d'aucun texte clair).

Ce chiffrement a l'avantage d'être sûr contre une attaque adaptative à texte chiffré choisi (on verra au chapitre que ce chiffrement est sûr au sens de IND-CCA2 pourvu que le problème décisionnel de Diffie-Hellman soit difficile). Cependant les temps de calcul sont plus longs.

1.3 Le chiffrement de Rabin

1.3.1 La fonction carrée

Si p est un nombre premier, il est facile de savoir, d'une part si un nombre x est un résidu quadratique, c'est-à-dire un carré modulo p, d'autre part de trouver les racines carrées de x modulo p dans le cas

où x est un carré. Il n'en est pas de même si on travaille modulo un entier n produit de deux grands nombres premiers (gardés secrets) p et q. Le problème du calcul des racines carrées d'un carré x modulo $n = pq$ est alors difficile. Il est équivalent **en pratique** au calcul de la factorisation de n. Plus précisément :

Théorème 1.7 *Pour tout entier n produit de deux grand nombres premiers p et q, et tout nombre x qui est un carré modulo n, il existe un algorithme de Las Vegas qui utilise un oracle fournissant la factorisation de n, et qui calcule les racines carrées de x.*

Preuve. Le résultat indiqué provient de la démarche suivante : on calcule les racines carrées de x modulo p et modulo q en utilisant si nécessaire l'algorithme de Shank. On reconstitue les racines carrées de x modulo n à partir de l'algorithme des restes chinois. □

Théorème 1.8 *Pour tout entier n produit de deux grand nombres premiers p et q, il existe un algorithme de Las Vegas, qui utilise un oracle fournissant une racine carrée modulo n de tout carré donné, et qui calcule les facteurs p et q de n.*

Preuve. On choisit un nombre u au hasard et on pose $a = u^2 \mod n$. L'oracle nous fournit une racine carrée v de a. Il y a une chance sur deux pour que $u \not\equiv \pm v \quad (n)$. Dans ce cas on sait que $u^2 - v^2 = kn$, c'est-à-dire que n divise $(u+v)(u-v)$. Mais n ne divise ni $u+v$ ni $u-v$. Donc p divise l'un de ces deux facteurs et q divise l'autre. Si on calcule $pgcd(n, u+v)$ on obtient donc l'un des deux nombres p ou q. Si on tombe sur un v tel que $v \equiv \pm u \quad (n)$ on tire un autre u. □
Remark. En revanche on ne sait pas si la détermination du caractère quadratique d'un nombre x modulo $n = pq$ (où p et q sont premiers), c'est-à-dire savoir si x est un carré ou non modulo n, est équivalente à la factorisation de n. Bien sûr, si on sait factoriser n, il est facile de déterminer le caractère quadratique de tout x. Par contre, on ne dispose d'aucun algorithme qui permette d'obtenir la factorisation de n à partir

de la détermination du caractère quadratique de tout x. Remarquons encore que le symbole de Jacobi se calcule en temps polynomial sans connaître la factorisation de n. Malheureusement lorsque n n'est pas premier, et que le symbole de Jacobi de x modulo n vaut 1, on ne peut rien conclure sur le caractère quadratique de x.
Remark. En pratique on est toujours hors des cas particuliers $x \equiv 0 \quad (n)$, x multiple de p sans être multiple de q, x multiple de q sans être multiple de p ; on remarquera que ces deux derniers cas permettent de factoriser n ; en conséquence on est en pratique toujours dans le cas où il y a quatre racines carrées pour un carré modulo n.

Théorème 1.9 *Considérons le cas où p et q sont tous les deux des grands nombres premiers congrus à 3 modulo 4. Pour tout x résidu quadratique modulo n, il existe une racine carrée et une seule qui soit elle même un résidu quadratique.*

Preuve. Faisons tout d'abord les deux remarques suivantes : Tout d'abord x est un carré modulo n, si et seulement si c'est un carré modulo p et modulo q, ensuite, si un nombre premier est de la forme $4k+3$, pour tout $u \neq 0$, un et un seul des deux nombres $u, -u$ est un résidu quadratique (il suffit pour le voir de calculer leurs symboles de Legendre). Notons u la racine carrée de x modulo p qui est un résidu quadratique modulo p et v la racine carrée de x modulo q qui est un résidu quadratique modulo q. Alors la racine carrée x_1 de x modulo $n = pq$, solution du système de congruences

$$\begin{cases} x_1 \equiv u \quad (p), \\ x_1 \equiv v \quad (q), \end{cases}$$

est un résidu quadratique. Les trois autres, dont les deux résidus modulo p et q ne sont pas simultanément des résidus quadratiques, ne sont pas des résidus quadratiques. □

1.3.2 Application de la fonction carrée au chiffrement de Rabin

La clé publique d'un utilisateur A est un module n produit de deux grands nombres premiers p et q congrus à 3 modulo 4. La clé privée de A est donnée par les facteurs p et q de n. Si on veut chiffrer un message m (où $0 < m < n - 1$) on calcule $x = m^2 \mod n$, le symbole de Jacobi de m et le bit b auquel on attribue la valeur 1 si $m \geq n/2$, et la valeur 0 sinon. On transmet ces données à A, qui dans un premier temps peut retrouver les 4 racines carrées $m_1, -m_1, m_2, -m_2$ de x. Le symbole de Jacobi de m lui permet de savoir si le message est l'un des messages $m_1, -m_1$ ou l'un des messages $m_2, -m_2$. Une fois que ce couple $m_i, -m_i$ est déterminé, la valeur du bit b lui permet alors de savoir si $m = m_i$ ou si $m = -m_i$.

2 Signature

Nous décrivons ici **les schémas de signature avec appendice**, c'est à dire ceux qui consistent à rajouter au message un résumé signé du message. Nous ne parlerons pas des **schémas de signature avec récupération du message** pour lesquels le message clair est reconstitué à partir de la seule signature.
Rappelons les principes d'une telle signature. Le système définit une fonction de hachage publique h qui transforme tout message à signer M en un message condensé $m = h(M)$. Chaque utilisateur X dispose d'une clé publique e_X et d'une clé privée d_X. Le système définit une fonction de signature $\mathcal{S}$ qui à une clé privée d_X et à un message condensé m fait correspondre $s = \mathcal{S}(d_X, m)$, appelé **appendice** de la signature de M par l'utilisateur X. Le système définit également une fonction de vérification $\mathcal{V}$ qui à une clé publique e_X, et à un message signé (M, s) fait correspondre $\mathcal{V}(e_X, M, s)$, valant "vrai" si (M, s) correspond bien à un message signé par X et "faux" sinon.
La clé privée définit **une fonction de signature** S_X définie par $S_X(M) = (M, \mathcal{S}(d_X, h(M)))$. La clé publique définit une **fonction de vérifi-**

cation V_X, d'ue signature provenant de X, définie par $V_X(M,s) = \mathcal{V}(e_X, M, s)$, de telle sorte que :

$$V_X(M,s) = \begin{cases} 0 & si \quad (M,s) \neq S_X(M), \\ 1 & si \quad (M,s) = S_X(M). \end{cases}$$

Supposons que B souhaite envoyer un message signé M à A.

- Tout d'abord B utilise la fonction de hachage publique h pour calculer le résumé $m = h(M)$.
- B utilise alors sa clé privée d_B pour calculer la partie signature proprement dite $s = S(d_B, m)$.
- Le message signé (m, s) est transmis à A.

Remarquons que le message signé est **le message M lui-même, auquel on adjoint l'appendice s.**

Le destinataire A connaît alors M et s.

- Il calcule tout d'abord $m = h(M)$.
- Alors A utilise la clé publique e_B de B pour vérifier la signature en calculant $V(e_B, M, s)$.

2.1 La signature d'ElGamal

On fixe un grand nombre premier p. On utilise le groupe multiplicatif cyclique $G = (\mathbb{Z}/p\mathbb{Z})^*$. Soit g un générateur de ce groupe (ainsi tout élément de $(\mathbb{Z}/p\mathbb{Z})^*$ est une puissance de g). L'utilisateur A choisit un x tel que $0 \leq x \leq p-2$ et calcule

$$y = g^x \; mod \, p.$$

La clé publique de A est (p, g, y), sa clé privée est x.

Lorsque A veut signer un message M, grâce à une fonction de hachage publique h bien choisie il calcule $m = h(M)$ avec $0 \leq m \leq p-1$. Puis il tire au hasard k tel que $1 \leq k \leq p-2$ et k premier avec $p-1$. il calcule $r = g^k \; mod \, p$ et $s = k^{-1}(m - rx) \; mod \, (p-1)$. Le message signé est (M, r, s).

Pour vérifier la signature on procède alors de la façon suivante : on calcule $m = h(M)$ puis $v = g^m \ mod \, p$ et $w = y^r r^s \ mod \, p$.
Si c'est bien A qui a signé alors $r < p$ et $v = w$. Dans ce cas la signature sera acceptée.
Cette signature peut être attaquée si le générateur g de G est un "petit diviseur" de $p - 1$. Plus précisément, on suppose maintenant que $p = 4u + 1$ et que $p - 1 = gt$. Puisque g divise $p - 1$ on sait qu'il existe un sous-groupe cyclique H d'ordre g du groupe cyclique G. On suppose en outre g suffisamment petit pour qu'on sache calculer le logarithme discret dans ce sous-groupe H. Remarquons que g^t est un élément primitif de H. De plus, y^t appartient au sous-groupe H puisque $(y^t)^g = 1$. Comme la résolution du problème du logarithme discret est possible dans H, on peut calculer z tel que $g^{tz} = y^t$. Comme g est un élément du groupe G on sait que $g^{p-1} \equiv 1 \quad (p)$ et donc que $g^{\frac{p-1}{2}} \equiv \pm 1 \quad (p)$. Mais comme g est primitif cette puissance de g n'est sûrement pas 1, et on peut conclure que

$$g^{\frac{p-1}{2}} \equiv -1 \quad (p).$$

On sait que $gt \equiv -1 \quad (p)$, donc

$$(gt)^{\frac{p-3}{2}} \equiv (-1)^{\frac{p-3}{2}} \quad (p),$$

ce qui donne compte tenu de l'hypothèse faite sur p

$$(gt)^{\frac{p-3}{2}} \equiv -1 \quad (p),$$

puis

$$g^{-1} g^{\frac{p-1}{2}} t^{\frac{p-3}{2}} \equiv -1 \quad (p),$$

et donc

$$t^{\frac{p-3}{2}} \equiv g \quad (p).$$

Un adversaire, voulant imiter la signature de A calcule $m = h(M)$ puis prend $r = t$ et $s = 1/2(p-3)(m - tz) \ mod \ (p-1)$. Dans ces conditions

$$w = y^t \left(t^{\frac{p-3}{2}} \right)^{(m-tz)} \mod p,$$

et donc

$$w = y^t g^{(m-tz)} \mod p,$$

c'est-à-dire

$$w = y^t g^m y^{-t} \mod p,$$

ce qui permet de conclure que $w = v$. Cette signature forgée est donc acceptée comme signature de A.

2.2 Le standard DSS de signature

Le **NIST** définit dans la note **FIPS 186-2** le standard de signature **DSS** (**D**igital **S**ignature **S**tandard).
Ce standard autorise trois algorithmes de signature : **DSA** (**D**igital **S**ignature **A**lgorithm), **ECDSA** (**E**lliptic **C**urve **D**igital **S**ignature **A**lgorithm) et **RSA**.

2.2.1 La signature DSA

La signature DSA est basée sur le problème du logarithme discret. Elle est reliée à la fonction d'ElGamal et ressemble à la signature d'ElGamal. Décrivons plus précisément la fonction de signature et le vérificateur. Les valeurs numériques que nous imposons ici sont celles du standard.

Soit p un grand nombre premier vérifiant :

$$2^{t-1} < p < 2^t,$$

où t est un multiple de 64 tel que :

$$512 \leq t \leq 1024.$$

Remarquons que p a t bits.
Soit q un diviseur premier de $p - 1$ où :

$$2^{159} < q < 2^{160}.$$

Le nombre premier q a 160 bits.
Soit α un élément d'ordre q de $(\mathbb{Z}/p\mathbb{Z})^*$. Soit a un élément de $(\mathbb{Z}/q\mathbb{Z})^*$ et $\beta = \alpha^a \mod p$.
La clé publique de A est (p, q, α, β), sa clé privée est a. Si A veut envoyer un message signé à B, il calcule le résumé m du message M et tire un entier k aléatoire (un pour chaque signature) dans $(\mathbb{Z}/q\mathbb{Z})^*$. Ensuite il calcule :

$$r = (\alpha^k \mod p) \mod q$$

et

$$s = (k^{-1}(m + ar)) \mod q.$$

Enfin, A transmet $(M, (r, s))$.
Pour vérifier le message signé reçu $(M, (r, s))$, B calcule le résumé m de M, puis il s'assure que r et s sont bien compris entre 0 et $q - 1$, enfin il calcule :

$$\begin{aligned} w &= s^{-1} \mod q, \\ u_1 &= mw \mod q, \\ u_2 &= rw \mod q, \\ v &= (\alpha^{u_1}\beta^{u_2} \mod p) \mod q. \end{aligned}$$

Si $v = r$ la signature est vérifiée.
Preuve. Si la signature (r, s) est correcte, alors :

$$\begin{aligned} w &= k(m + ar)^{-1} \mod q, \\ u_1 &= km(m + ar)^{-1} \mod q, \\ u_2 &= kr(m + ar)^{-1} \mod q, \\ v &= \left(\alpha^{km(m+ar)^{-1}}\alpha^{akr(m+ar)^{-1}} \mod p\right) \mod q, \\ v &= \left(\alpha^{km(m+ar)^{-1}+akr(m+ar)^{-1}} \mod p\right) \mod q, \\ v &= \left(\alpha^k \mod p\right) \mod q, \end{aligned}$$

$$v = r. \quad \square$$

Remark. Les nombres p, q, α peuvent être communs à un groupe d'utilisateurs.

L'algorithme DSA a besoin de deux nombres premiers p et q tels que :

$$2^{159} < q < 2^{160},$$

$$2^{k-1} < p < 2^k,$$

$$q|(p-1),$$

$$512 \leq k = 512 + 64j \leq 1024,$$

et d'un élément α d'ordre q.

Une méthode pour obtenir ces valeur a été décrite dans la section 1.2.3. Nous renvoyons aussi à la note **FIPS 186-2** du NIST pour les détails d'implémentation.

Remark. Il existe un moyen de détournement de la signature DSA, qu'on appelle plus précisément un canal subliminal, et qui consiste à utiliser le système de la façon suivante. Supposons que A et B se mettent d'accord et que B connaisse la clé privée a de A. Alors A ne tire pas k au hasard, mais choisit pour k un message à transmettre secrètement à B. A choisit alors un message clair quelconque M, de haché m, le signe et transmet à B :

$$r = (\alpha^k \mod p) \mod q$$

et

$$s = (k^{-1}(m + ar)) \mod q.$$

B qui connaît a, s, m, r, peut reconstituer k. Ainsi A et B font semblant d'utiliser un système de signature, alors qu'en réalité ils utilisent un système de chiffrement.

2.2.2 La signature RSA

La fonction RSA peut être utilisée pour signer. Le système à mettre en place est le même que pour le chiffrement. Mais un utilisateur qui

veut signer un message, **signe le résumé** en chiffrant celui-ci **avec sa propre clé privée**. Il obtient ainsi l'appendice. La **vérification** est faite en déchiffrant l'appendice avec la **clé publique de l'expéditeur**.
Nous n'insistons pas sur la nécessité d'utiliser un protocole bien étudié pour générer et vérifier des signatures.
Nous renvoyons au document élaboré par la société RSA : **PKCS# 1 v2.1 : RSA Cryptography Standard** pour une description complète de **RSASSA-PSS** (**RSA S**ignature **S**cheme with **A**ppendix-**P**robabilistic **S**ignature **S**cheme). Nous donnons juste la suite des opérations à faire. Ces opérations sont analogues à celles utilisées pour le chiffrement RSA.

- **Procédure d'encodage**. Le message (suite d'octets) M est transformé en une chaîne d'octets EM de longueur $emlen$. Ce bloc EM est le **message encodé**. La transformation utilise la procédure d'encodage EMSA-PSS. (EMSA : **E**ncoding **M**ethod for **S**ignature with **A**ppendix. PSS : **P**robabilistic **S**ignature **S**cheme).
- **Transformation en un entier**. Maintenant EM est transformé en un entier m par OS2IP (Octet String To Integer Primitive).
- **Procédure de signature**. L'entier m est transformé en s par RSASP1 (RSA Signature Primitive).
- **Transformation en suite d'octets**. L'entier s est transformé en une suite S par I2OSP (Integer To Octet String Primitive).

Quand le destinataire dispose de (M, S) il applique la procédure de vérification suivante :

- **Transformation en un entier**. S est transformé en un entier s par OS2IP (Octet String To Integer Primitive).
- **Inversion de la signature**. L'entier s est transformé en m par RSAVP1 (RSA Verification Primitive).
- **Transformation en suite d'octets**. L'entier m est transformé en une suite d'octets EM par I2OSP (Integer To Octet String Primitive).
- **Vérification**. La fonction de **vérification** EMSA-PSS est appliquée à M et à EM pour déterminer si ces valeurs sont cohérentes.

Les attaques du système de chiffrement RSA, décrites précédemment, s'appliquent aussi à la signature. Voici en outre un type d'attaque (appelé attaque par faute) qui s'applique à une implémentation bien particulière de RSA. Nous utilisons un système RSA pour signer des messages. Nous nous préoccupons seulement de la partie qui applique la fonction RSA à un message haché m. Le système est donc composé d'un module $n = pq$, d'une clé publique e et d'une clé privée d. On supposera que le message haché m ($0 < m < n$) à signer est premier avec n (sinon c'est catastrophique car en prenant le pgcd de m et n on obtient un facteur de n). L'utilisateur doit alors calculer la signature s à joindre au message

$$s = m^d \mod n.$$

Afin d'accélérer un peu les calculs il utilise la méthode qui suit.
Soient d_p et d_q définis respectivement par :

$$d_p = d \, mod \, (p-1),$$

$$d_q = d \, mod \, (q-1).$$

On a alors les congruences suivantes :

$$d_p e \equiv 1 \quad (p-1),$$

$$d_q e \equiv 1 \quad (q-1),$$

puis :

$$s \equiv m^{d_p} \quad (p),$$
$$s \equiv m^{d_q} \quad (q).$$

On calcule donc :

$$s_p = m^{d_p} \mod p$$

et

$$s_q = m^{d_q} \mod q.$$

et on établit s par la formule des restes chinois.

On suppose que pour une raison ou une autre, lors du calcul de

$$s_q = m^{d_q} \mod q$$

une erreur se produise, si bien qu'au lieu de calculer s on calcule s' tel que

$$s' \equiv m^{d_p} \quad (p),$$
$$s' \not\equiv m^{d_q} \quad (q).$$

Dans ces conditions on peut dire que :

$$(s')^e \equiv m \quad (p),$$

$$(s')^e \not\equiv m \quad (q).$$

Si bien que :

$$pgcd((s')^e - m, n) = p.$$

Ainsi grâce au message m (dont le destinataire dispose), à la signature erronée s' et aux données publiques n et e, le destinataire peut calculer la factorisation de n.