RSA : Théorie et attaque

La cryptographie, c'est à dire l'étude des systèmes de chiffrement et de sécurisation, est une science qui évolue vite. Nous en voulons pour preuve la multitude de documents datant de cette année (parus essentiellement dans les conférences Crypto'98, EuroCrypt'98, AusCrypt'98). Si elle existe depuis la nuit des temps (César avait son propre algorithme de cryptage, toutes les diplomaties du monde en utilisent un), elle a connu un essor fulgurant ces dernières années avec l'expansion d'Internet. Ainsi, certaines disciplines mathématiques (comme l'arithmétique, et en particulier la factorisation des nombres (courbes elliptiques, crible algébrique), qui datent de moins de 20 ans), ont-elles connu un renouveau par leur utilité dans la cryptographie.

Le but de la présente étude est de montrer la corrélation très étroite qui doit exister entre cryptographie et cryptanalyse, c'est à dire la science de l'attaque des systèmes cryptographique.

Pour ce faire, on étudiera le RSA (du nom de ses créateurs, Rivet, Shamir, et Adleman) un système cryptographique à clef publique, rendu possible par l'existence d'une fonction à sens unique.

I. Codage RSA

A. Théorie des nombres

Définitions :

On note pgcd (a, b) le plus grand nombre qui divise a et b.

On note $\subseteq_n$ le sous anneau des restes modulo n. Ainsi on a la surjection canonique de $\subseteq$ dans $\subseteq_n$ à qui z associe z mod n.

On dispose de l'algorithme d'Euclide qui calcule le pgcd de deux nombres.

De plus, l'ensemble des nombres inversibles de $\subseteq_n$ est { z < n / z $\wedge$ n = 1} et on possède l'algorithme d'Euclide étendu qui calcule l'inverse d'un nombre inversible. On note le cardinal de cet ensemble $\Phi(n)$.

Le petit théorème de Fermat : pour tout b non nul, pour p premier, $b^{p-1} \bmod p = 1$.

On a également $\Phi(p) = p - 1$ pour p premier, $\Phi(n) = (p-1)(q-1)$ si p, q premier et $n = p * q$.

De plus, un théorème des restes chinois donne : pour p et q premiers entre eux,

$$\varphi : \begin{array}{ccc} Z_n & \to & Z_p \times Z_q \\ x \bmod n & \mapsto & (x \bmod p, x \bmod q) \end{array}$$ est un isomorphisme d'anneaux

Enfin, le théorème de Lavallée Poussin donne la raréfaction des nombres premiers : la probabilité pour que (p tiré aléatoirement soit premier) équivaut à 1/log(p).

B. Application : une fonction à trappe

Supposons donnés p et q premiers distincts. On pose $n = p\,q$, $\Phi(n) = (p-1)(q-1)$.

On prend alors $c < \Phi(n)$ tel que pgcd(c, $\Phi(n)$) = 1. Donc c inversible dans $\subseteq_{\Phi(n)}$, et on note d son inverse, calculé grâce à l'algorithme d'Euclide étendu : $c\,d = t\,\Phi(n) + 1$.

Pour x ♣ $\subseteq_n$, on a $(x^c)^d = x^{c\,d} = x * (x^t)^{\Phi(n)}$

On applique le théorème chinois aux quantités : $a_1 = (x \bmod p)$ et $a_2 = (x \bmod q)$. x convient, donc c'est l'unique solution modulo n.

Or $x(x^t)^{\Phi(n)} \equiv x(x^{t(q-1)})^{p-1} \equiv x \pmod p$ d'après Fermat

En effet, si $x \equiv 0 \pmod p$, *alors on a bien* $x(x^{t(q-1)})^{p-1} \equiv 0 \equiv x \pmod p$.

Si $x \neq 0 \pmod p$, *comme* $\subseteq_p$ *est un corps,* $(x^{t(q-1)}) \neq 0 \pmod p$.

Donc d'après Fermat, $(x^{t(q-1)})^{p-1} \equiv 1 \pmod p$, *ie* $x(x^{t(q-1)})^{p-1} \equiv x \pmod p$

De même, $x(x^t)^{\Phi(n)} \equiv x(x^{t(p-1)})^{q-1} \equiv x \pmod q$

Donc $x(x^t)^{\Phi(n)}$ convient aussi, d'où $x(x^t)^{\Phi(n)} \equiv x \pmod n$

On a donc $(x^c)^d \equiv x(x^t)^{\Phi(n)} \equiv x \pmod n$.

On a alors trouvé une fonction à trappe, c'est à dire une fonction bijective dont la fonction réciproque est presque impossible à trouver à partir de la seule connaissance de la fonction. On peut donc définir un système à clef publique

C. Mise en place du RSA

Un système cryptographique à clef publique requière une fonction à trappe : on fournit la fonction f, qui servira à crypter, tout en connaissant le secret (la trappe) qui permet de calculer f^{-1}, pour décrypter.

En fait, pour le RSA la fonction est $f : \begin{matrix} Z_n \to Z_n \\ x \mapsto x^d \pmod n \end{matrix}$,et sa réciproque est $f^{-1} : \begin{matrix} Z_n \to Z_n \\ x \mapsto x^e \pmod n \end{matrix}$

Ainsi, concrètement, Bob choisit « aléatoirement » deux nombres premiers « forts » p et q, puis il calcule $\Phi(n) = (p-1)(q-1)$.

Il choisit ensuite son **exposant public d**, qui doit satisfaire pgcd(d, Φ(n)) = 1 pour que d soit inversible, et il en calcule l'inverse par l'algorithme d'Euclide étendu, qu'il note $e = d^{-1} \bmod \Phi(n)$.

Il publie ensuite **(d, n), sa clef publique**, en gardant jalousement **e secret (c'est sa clef secrète**).

Pour qu'on lui envoie des informations décryptables par lui seul, Alice doit suivre le processus de la figure 1.

D. Génération de nombres premiers particuliers, implémentations

Exponentiation modulaire (programme 1)

Pour utiliser le RSA, il faut effectuer de nombreux calculs du type $x^d \bmod n$. On utilise une méthode "diviser pour régner " pour calculer ses quantités très rapidement afin que le RSA soit efficace.

Test probabiliste de Miller Rabin pour la primalité d'un chiffre. (programme 2)

Pour mettre en place le RSA, il faut trouver des grands nombres premiers, p et q, (de 100 à 150 chiffres de long). Comment les obtenir ?

On utilise ici un algorithme probabiliste de Monte Carlo négatif pour la primalité (donc positif pour la factorisabilité), c'est à dire que, s'il répond qu'un nombre est factorisable, ce nombre est factorisable, et s'il répond que le nombre est premier, la probabilité d'erreur est de ε. On n'a donc ici qu'une pseudo primalité (probabiliste), mais on réitère le test suffisamment de fois pour que la probabilité qu'un nombre annoncé premier le soit de plus de $1-2^{-100}$ *(50 itérations du test de Miller Rabin).*

Note : il existe des algorithmes démontrant de manière déterministe la primalité d'un nombre, mais ils demandent une théorie poussée ou une implémentation si compliquée qu'une erreur du programme a une probabilité d'erreur de plus de 2^{-100}.

Soit k et m impair tels que $n-1 = 2^k m$
Soit a entier tel que 1 7 a < n
On pose $b = a^m \bmod n$
Si (b mod n) = 1 **alors** renvoyer "n premier" et fin
Faire k fois
    Si (b mod n) = -1 **alors** renvoie "n premier" et fin
        sinon b := b²
Fin Faire
renvoyer "n décomposable"

Une étude montrerait que la probabilité d'erreur de l'algorithme est majorée par 1/4 . D'où les 50 itérations du test ($4^{-50} = 2^{-100}$).

Pourquoi faut-il des nombres premiers particuliers, lesquels, comment les obtenir ?

Pour prévenir de la méthode p-1 de pollard et p + 1 de ??, il faut que ces nombres admettent au moins un grand facteur premier :

c-1 admet un grand facteur premier p_1 si $c - 1 \bmod p_1 = 0$, c'est à dire $c=1 \bmod p_1$.

De même c+1 admet un grand facteur premier p_2 si $c = -1 \bmod p_2$.

On choisit donc aléatoirement deux nombres premiers p1 et p2 de la taille voulue pour les facteurs premiers de c+1 et c-1.

On pose $R = (p_2^{-1} \bmod p_1)\, p_2 - (p_1^{-1} \bmod p_2)\, p_1$.

On a $R \bmod p_1 = (p_2^{-1} \bmod p_1)\, p_2 \bmod p_1 = -1$ par définition de l'inverse. De même $R \bmod p_2 = 1$.

Ainsi, on prend $R + n\, p_1\, p_2$, où n est tel que ce nombre soit premier de la grandeur voulue.

Il est à noter que R est unique modulo $p_1\, p_2$ d'après le théorème des restes chinois.

II. Problème de sécurité

A. Un Protocole de transmission : Le PKCS #1 v2

En pratique, on rajoute devant le message à coder (ou les parcelles de message), quelques octets :

$$EM = 02 \,||\, PS \,||\, 00 \,||\, M$$

Où PS est une suite d'octets non nuls pris au hasard, de longueur k – 3 – |M| et doit être au moins de huit. Au total, EM est de longueur k-1, où k est la longueur de n en octets.

Utilité de rajouter le premier octet : Pour tester l'intégrité du message : si au décryptage on ne trouve pas 02 au départ, c'est que le message n'a pas été transmis correctement.

Utilité de PS . (Padding String) : Si on envoie deux fois le même message, un observateur ne pourra pas le savoir car EM sera différent.

B. Signature

Fonction de Hachage : on introduit une suite de caractères d'une longueur quelconque et la fonction de hachage renvoie une suite de longueur prédéfinie. De plus, il faut que cette fonction satisfasse à l'axiome de "sens unique" (one way) et celui de "non collision" (collision free), c'est à dire que connaissant le résultat, il est impossible de trouver un texte convenant, et qu'il est impossible de trouver deux textes donnant le même résultat par cette fonction de hachage. Elle donne la "digital fingerprint" d'un texte, c'est à dire son empreinte électronique.

Si Bob veut signer un message, il doit suivre le procédé décrit figure 2. On peut évidemment cumuler signature et cryptage (on réalise d'abord la signature, puis on crypte le message). On peut, en outre, pour éviter qu'un vieux message ne soit envoyé en faisant croire à une nouvelle, recourir au « *Timestamping* », c'est à dire à la datation du message, assurée par un organisme extérieur en qui tout le monde à confiance.

C. Factorisation . Preuve de la sûreté de RSA ?

En fait, on n'a pas de sûreté théorique puisque pour un texte y chiffré, on peut tester $a^d \bmod n$ pour $a < n$ jusqu'à ce que l'on trouve y, où a est unique car la fonction de chiffrement est bijective. La seule sûreté dont on doit se préoccuper est donc la sûreté calculatoire, c'est à dire vérifier qu'il est impossible de trouver la clef secrète de RSA en un temps convenable (l'âge de l'univers par exemple). En fait, c'est encore un problème ouvert. La seule chose que l'on sait est que casser le RSA est au moins aussi facile que de factoriser n. En effet, si on connaît la factorisation de n, on en déduit $\phi(n)$ et donc on peut calculer facilement la clef secrète, inverse de la clef publique modulo $\phi(n)$ (de la même manière que Bob a calculé sa clef secrète).

Et même en supposant que le cassage de RSA est polynomialement équivalent à celui de factoriser n, on ne sait pas s'il est impossible de factoriser n de façon rapide. Le meilleur algorithme à ce jour peut factoriser des nombres de 150 bits, et est en $O(\exp(1.92+o(1))(\log n)^{1/3} (\log \log n)^{2/3}$ (crible algébrique), mais sa complexité est telle qu'on ne l'exposera pas ici (cet algorithme est fondé sur la recherche de x, y tel que x ne soit pas congru à y ou - y modulo n, mais que x^2 soit congru à y^2 modulo n). Et rien ne dit qu'il n'existe pas de meilleur algorithme.

On expose ici une méthode capable de factoriser un nombre pourvu qu'un de ses facteurs premiers p soit tel que p-1 n'admette que des "petits" facteurs puissance de nombres premiers, c'est à dire soit fortement B-friable, pour B petit. (Pour tout n^k divisant (p-1) où n premier, n^k 7 B). C'est la méthode p-1 de pollard. (programme 3).

```
Soit B et g < n
On pose a = (g^B! mod n)
on pose d = pgcd(a-1,n)
renvoyer "d facteur de n "
```

En effet, d = pgcd(a-1,n), donc d divise n. Intéressons nous au cas où il existe p diviseur premier de n tel que p-1 soit fortement B-friable.

Si p ne divise pas g, comme (p-1) divise B!, $g^{B!} \bmod p = 1$ par Fermat.

Or $a \equiv g^{B!} \bmod n$, d'où $a \equiv g^{B!} \bmod p$ car p divise n, i.e. $a \equiv 1 \bmod p$: **p divise (a-1).**

Or p divise n, donc p divise pgcd (n, a-1) = d : d= k p, k entier. k non nul si et seulement si a-1 non nul.

Ainsi, si p - 1 B-friable et $g^{B!} \notin p\,\mathfrak{I} \cup \{1 + n\,\mathfrak{I}\}$, on obtient un facteur non trivial.

Complexité : Pour calculer $g^{B!} \bmod n$, on réalise g X ($g^{i} \bmod n$) pour i de 1 à B. Or ces exponentiations modulaires sont en O(log i), donc au total O(B log B). Le calcul du pgcd est en $O(\log n)^3$.

Si l'on prend $B = n^{1/2}$, l'algorithme réussira forcément, mais on n'aura pas un algorithme polynomialement équivalent à la réalisation du code: si l'on prend un grand nombre n, casser le RSA avec cette méthode est exponentiellement plus compliqué que réaliser la clef et le codage, donc irréalisable en pratique.

Si l'on prend $B = (\log n)^i$, on aura un algorithme polynomialement équivalent à la réalisation du code (qui requière le calcul d'un pgcd, donc en $O(\log n)^3$), donc utilisable en pratique, mais qui ne fonctionnera que si n admet un diviseur p premier tel que p -1 soit B-friable.

III. L'attaque de bleichenbacher de PKCS #1

On explique ici une attaque fondée sur une faille dans le protocole de communication PKCS #1, ce qui en fait l'originalité. Il s'agit d'une attaque informatique, réalisable dans la pratique, ce qui la différentie des attaques plus mathématiques, s'appuyant sur des présupposés difficilement réalisables dans la pratique. C'est une attaque adaptative à texte chiffré choisi.

A. Description de l'attaque

On note k la longueur de n en octet. D'après le protocole PKCS, le nombre à transmettre est, avant d'être crypté, compris dans $[2\ 2^{8(k-2)}; 3\ 2^{8(k-2)}[$. Nous supposons tout d'abord en notre possession un « oracle » qui nous dit si un texte chiffré, une fois décrypté, est PKCS – conforme, donc si en particulier son résidu modulo n est dans [2 B; 3 B –1], où $B = 2^{8(k-2)}$. En pratique, on envoie le message au serveur et on regarde s'il renvoie un message d'erreur ou non.

Supposons que l'on veuille connaître la signification du message crypté c, c'est à dire c^e où e est l'exposant privé. On notera M_i un ensemble d'intervalles de $\Im$ auquel on sait que ($c^e \bmod n$) appartient à l'étape i, et on notera $c_i = (\Delta_i^d\ c \bmod n)$ le « message crypté » que l'on va tester à cette étape.

L'idée de l'algorithme est que chaque nouvelle acceptation d'un message va nous fournir des informations, donc va permettre de diminuer les intervalles de plus en plus. On réitère le test jusqu'à ce qu'il ne reste plus qu'une unique possibilité.

Initialisation : .On prend $c_0 = c$, qui satisfait c_0^e est PKCS-conforme. Si l'on veut que le destinataire ne sache pas quel message on a décodé (« blinding »), on peut prendre $c_0 = \Delta_0^d c$, où Δ_0 est choisit pour que $\Delta_0 c^e$ soit PKCS-conforme. On se placera dans le premier cas, donc $M_0 = [2\ B\ ;\ 3\ B - 1]$.

Recherche de message PKCS-conforme : ($c^e \Delta_i \bmod n$) doit être supérieur à 2 B pour avoir des chances d'être PKCS-conforme. Or ($c^e \bmod n$) < 3B, donc Δ_i doit être supérieur à n/3B.

Tant que Mi diffèrent de {[a,a]}, faire

α **Si** M_{i-1} contient au moins deux intervalles ou i = 1, chercher le plus petit entier Δ_i à partir de $\max(\Delta_{i-1}, n/3B)$ tel que $c^e\Delta_i$ soit PKCS-conforme (on prend $c_i = c\ \Delta_i^d \bmod n$). On a alors $c^e\Delta_i \bmod n\ 0\ [2\ B; 3\ B - 1]$.

α **Sinon**, $M_{i-1} = \{[a, b]\}$. Chercher (ρ, Δ_i) tels que : $\begin{cases} \rho \geq 2\dfrac{b\varepsilon_{i-1} - 2B}{n} \\ \dfrac{2B + \rho n}{b} \leq \varepsilon_i < \dfrac{3B + \rho n}{a} \end{cases}$ et $c^e \Delta_i$ soit conforme.

Réduction : $M_i = \bigcup_{[a,b] \in M_{i-1}} \bigcup_r [a,b] \cap \left[\left\lceil \frac{2B + rn}{\varepsilon_i} \right\rceil, \left\lfloor \frac{3B - 1 + rn}{\varepsilon_i} \right\rfloor\right], avec \frac{a\varepsilon_i - 3B + 1}{n} \leq r \leq \frac{b\varepsilon_i - 2B}{n}$

Fin du Tant que

Solution : ($c^e \bmod n$) appartient à un intervalle de $M_i = \{[a,a]\}$. Donc $c^e \equiv a \pmod{n}$.

B. Idée de la preuve de l'attaque

Montrons que $\forall i, c^e\, 0\, M_i$. Il existe [a, b] 0 M_{i-1} tel que c^e 0 [a, b]. Or il existe r tel que $2B \le (c^e \Delta_i - r n) < 3B$.

c^e 0 $\bigcup_{[a,b] \in M_{i-1}} \bigcup_{r} [a,b] \cap \left[\left\lceil \frac{2B + rn}{\varepsilon_i} \right\rceil, \left\lfloor \frac{3B - 1 + rn}{\varepsilon_i} \right\rfloor\right]$. De plus, si $r > (b \Delta_i - 2B)/n$ ou

$r < (a \Delta_i - 3B + 1)/n$, l'intersection avec [a,b] est vide. Donc $\frac{a\varepsilon_i - 3B + 1}{n} \le r \le \frac{b\varepsilon_i - 2B}{n}$. $\mathbf{c^e\, 0\, M_i}$.

Explication des conditions pour le cas $M_{i-1} = \{[a, b]\}$. Si on trouve Δ_i pour la conformité, il existe ρ tel que

$2B \le (c^e \Delta_i - \rho n) < 3B$. Donc nécessairement $\frac{2B + \rho n}{b} \le \varepsilon_i < \frac{3B + \rho n}{a}$.

De plus on impose $\rho \ge 2\frac{b\varepsilon_{i-1} - 2B}{n}$ pour réduire M_i : Pour tout $r \le (b \Delta_i - 2B)/n$, $r \le 2\rho$.

Donc $(3B-1+r n)/\Delta_i \le b\,(3B-1+r n)/(2B + \rho n)$. Or $B < n/256$ (B très inférieur à n), donc à peu près $(3B-1+r n) = r n$ et $(2B + \rho n) = \rho n$. Donc $(3B-1+r n)/\Delta_i \le b/2$. C'est à dire $Mi \subset [a,b/2]$: on réduit l'intervalle de moitié au moins. Dans le cas où M_{i-1} est quelconque, on pourrait au pire l'inclure dans un intervalle [α,β] et le réduire de moitié, d'où la terminaison de l'algorithme.

C. Attaque effectuable en pratique, complexité ?

La probabilité Pr(A) d'avoir un texte m de longueur < n dont le premier octet est 2 est dans $[2^{-16} ; 2^{-18}]$ (si n est un multiple de 8, la probabilité est la plus basse). La probabilité Pr(P/A) qu'un texte soit PKCS-conforme sachant A (c'est à dire que les 8 premiers octets après 2 soient non nuls et qu'il y en ait au moins un nul après) est supérieur à 0.18 pour k>64 (n de 512 bits). Donc Pr(P), la probabilité qu'un texte soit PKCS conforme, est dans $[0.18\, 2^{-16} ; 0.97\, 2^{-18}]$. On a donc approximativement $\Delta_1 = 1/Pr(P)$ et $\Delta_2 = 2/Pr(P)$

Heuristiquement, on arrive à : le nombre d'intervalles dans M_i est $\le 1 + \Delta_i/2\, (2\, B/n)^i$, majoration décroissante avec i. Donc en moyenne, on a 1 + 1/20 intervalles dans M_2, donc sans doute un seul intervalle. De même, il y a encore plus de chance que les autres M_i soit constitués d'un seul intervalle.

La taille de $[2\, B + \rho n)/\, b\, ;\, (3\, B - 1 + \rho n)/\, a]$ est supérieur à 1/3 (B-1)/B, donc pour à peu près 3 valeurs de ρ, l'intervalle sera non vide (donc il y a suffisamment de Δ_i pour en trouver un qui convienne). Or on a choisit Δ_i pour que A soit réalisé assez facilement : on a à peu près une chance sur deux pour que Δ_i pris dans $[2\, B + \rho n)/\, b\, ;\, (3\, B - 1 + \rho n)/\, a]$ soit dans $[2\, B + \rho n)/\, c^e\, ;\, (3\, B - 1 + \rho n)/\, c^e\,]$. On a donc Pr(P) = Pr(P/A)/2 pour de tels Δ_i.

Au total, il faut diminuer 8(k-2) fois l'intervalle de M_2 (de taille $2^{8\,(k-2)}$) en deux pour avoir un intervalle de taille un, et donc obtenir le résultat, auquel il faut rajouter le premier découpage.

Nombre d'essais = 3/Pr(P) + 8*(k-2) * 2/ Pr(P/A). Pour n de 1024 bits, il faudra 2^{20} essais.

Enfin, les expérimentations réalisées par l'équipe de Bleichenbacher ont montré que sur trois serveurs testés, deux étaient attaquables, et ont été attaqués avec succès. Dans la pratique, le nombre de textes utilisés était inférieur aux 2^{20} annoncés.

Conclusion

L'attaque que l'on vient d'exposer permet donc de tirer des enseignements sur le code RSA : il faut éviter le plus possible de donner des informations à un observateur extérieur (ne pas lui dire que son message n'est pas accepté, ou au moins ne pas lui dire que c'est parce qu'il n'est pas PKCS-conforme). De plus, il vaut mieux utiliser un modulo RSA qui ait un nombre d'octets multiple de 8 (de longueurs 1024 bits (300 chiffres) par exemple, qui est souvent utilisé dans la pratique)… Cette attaque illustre donc le fait que la réalisation d'un code cryptographique doive être réalisée en collaboration avec des personnes qui essayent de casser ces codes (cryptanalystes), et qu'il doit évoluer selon les nouvelles attaques inventées. Ainsi tout nouveau système cryptographique est exposé sur Internet sous forme de défi : « arrivez-vous à casser ce code ? ». On peut aussi observer l'utilisation de résultats heuristiques pour les preuves et les complexités en informatique.

Bibliographie :

Cryptographie : Théorie et pratique de Robert Stinson chez Thomson Research (Traduction Serge Vaudenay)
Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1 de Daniel Bleichenbacher (présenté à Crypto'98).
Presentation des Tipes 98/99 de l'ENSTA.
Contacts avec M. Guy Chassé de l'Ecole des Mines de Nantes
Contacts par E-mail avec M. Serge Vaudenay du GRECC
Utilisations intensives des sites Internet : de RSA laboratories (www.rsa.com)
Du GRECC (www.ens.fr)
De polytechnique (Page personnelle de François.Morin@lix.fr)
Utilisations d'une preversion de *X9.80 PRIME NUMBER GENERATION AND VALIDATION METHODS de l'ANSI (Commitée de standardisaton des Etats-Unis)*

PROGRAMME 1 : Exponentiation modulaire

On écrit d = $d_n..d_0$ en base 2, c'est à dire $d = \sum_{i=0}^{l-1} d_i 2^i$.

On pose z := 1
Faire pour i = l -1 jusqu'à 0
z := (z^2 mod n)
Si c_i = 1 **alors** z := (z x mod n)
Fin Faire
renvoie z.

PROGRAMME 2 : Test probabiliste de Miller

Soit k et m impair tels que n-1 = 2^k m
Soit a entier tel que 1 7 a < n
On pose b = a^m mod n
Si (b mod n) = 1 **alors** renvoyer "n premier" et fin
Faire k fois
Si (b mod n) = -1 **alors** renvoie "n premier" et fin
sinon b := b²
Fin Faire
renvoie "n décomposable"

Preuve : Le test renvoie n décomposable si (a^m mod n) ← 1 et ($a^{2^i m}$ mod n) ←-1 pour tout i< k.

Raisonnons par l'absurde : supposons n premier et que le test renvoie n décomposable.

$a^{n-1} \equiv a^{2^k m} \equiv 1 \pmod{n}$ d'après Fermat.

Donc $(a^{2^{k-1}m})^2 \equiv 1 \pmod{n}$, d'où $(a^{2^{k-1}m} - 1)(a^{2^{k-1}m} + 1) \equiv 0 \pmod{n}$. Comme n premier, $\subseteq_n$ est intègre : $a^{2^{k-1}m} \equiv \pm 1 \pmod{n}$. Or on sait que cette valeur est différente de -1, donc c'est 1. On peut donc appliquer la même propriété à $a^{2^{k-2}m}$... ainsi, par une récurrence triviale, il vient $a^m \equiv 1 \pmod{n}$, qui contredit la quatrième ligne de l'algorithme.

Nous avons donc à faire à un algorithme de Monte-Carlo négatif pour la primalité.

PROGRAMME 3 : Méthode p-1 de Pollard

Soit B et g < n
On pose a = ($g^{B!}$ mod n)
on pose d = pgcd(a-1,n)
renvoyer "d facteur de n "

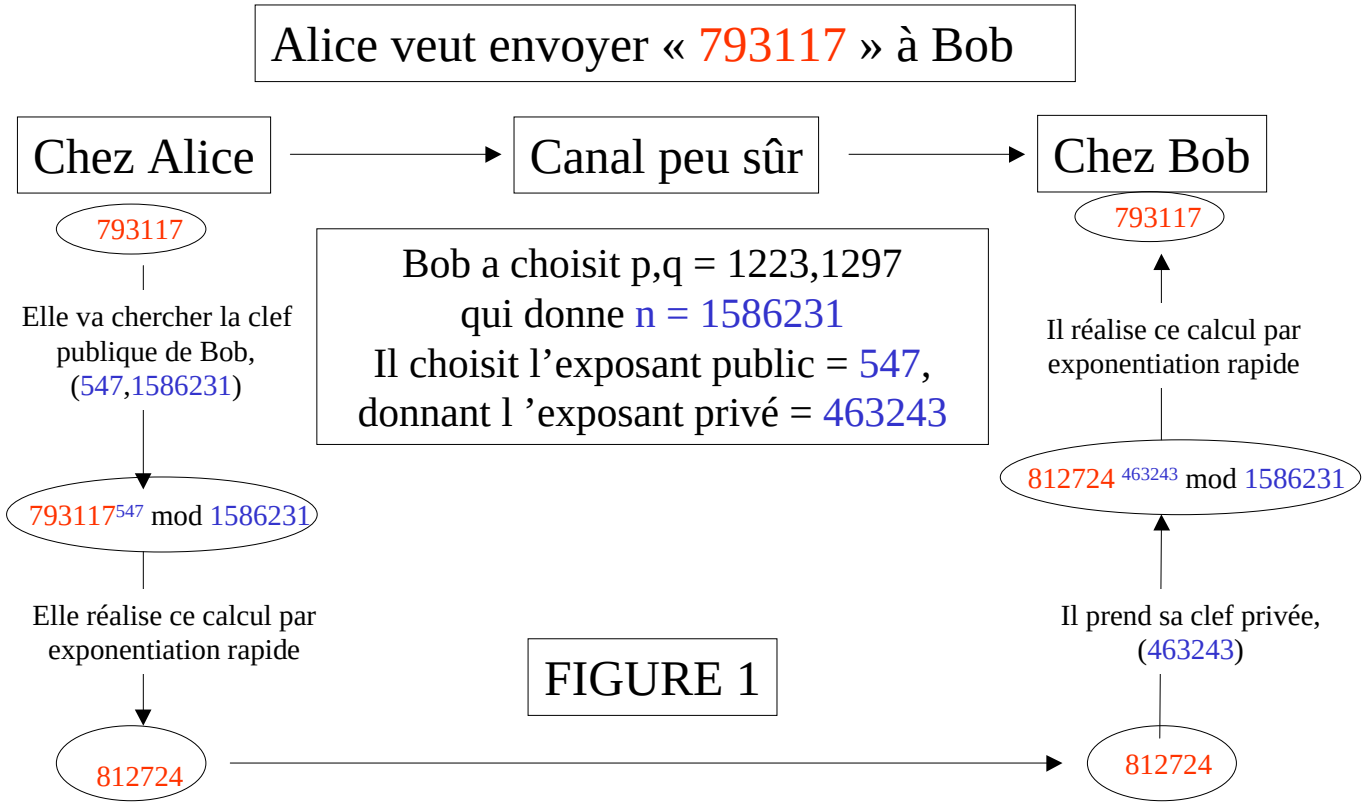

FIGURE 1

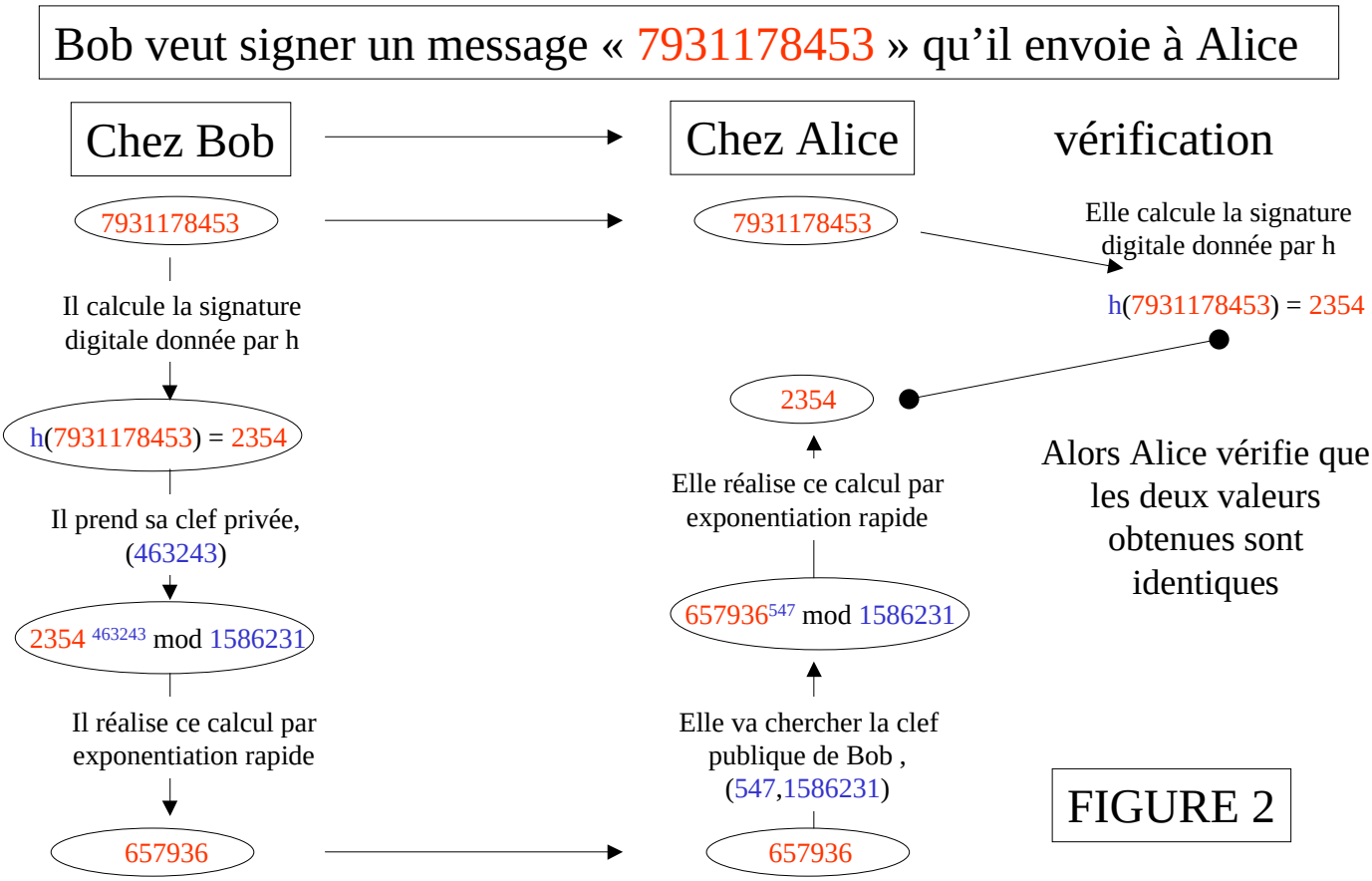

FIGURE 2

Remarque : on pourrait se passer de la fonction de hachage, mais alors il faudrait transmettre le texte **entier** une deuxième fois, ce qui serait plus coûteux en temps.