Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE
PREMIER MINISTRE

Secrétariat général
de la défense
nationale

*Direction centrale de la sécurité
des systèmes d'information*

Paris, le 6 novembre 2006

N° 2336/SGDN/DCSSI/SDS

Fournitures nécessaires à l'analyse de mécanismes cryptographiques

Version 1.2

Résumé

L'objet de ce document est de préciser les fournitures attendues par la DCSSI dans le cadre de l'analyse de mécanismes cryptographiques. Il comporte des recommandations sur le contenu et la forme de tels documents ainsi qu'un exemple de description pour chacune des primitives cryptographiques les plus classiques.

Abstract

The aim of this document is to state precisely which documents and information are required by DCSSI in order to analyze the security of cryptographic mechanisms. Recommendations for the redaction of such documents and examples of description of well known mechanisms are provided.

51 boulevard de Latour-Maubourg - 75700 PARIS 07 SP - Tél 01 71 75 84 65 - Fax 01 71 75 84 90

Table des matières

Annexes

Version 1.0	4 octobre 2002	Document initial
Version 1.1	30 octobre 2002	Traduction du document *en anglais* (hors annexes) et prise en compte des remarques mineures
Version 1.2	20 octobre 2006	Prise en compte des évolutions du référentiel relatives à la gestion des clés

Informations nécessaires à l'analyse de mécanismes cryptographiques par la DCSSI

Afin d'analyser la résistance et l'adéquation des mécanismes cryptographiques employés dans un produit de sécurité, la DCSSI exige un document de synthèse rédigé selon les règles suivantes :

1. Ce document doit décrire dans un premier temps le fonctionnement général du produit, dans ses aspects non cryptographiques, de la manière la plus **concise**, **synthétique** et **didactique** possible. La description doit être complète à l'échelle du produit, indépendamment d'une éventuelle cible de sécurité. Le but est de permettre une compréhension globale des objectifs et du fonctionnement général du produit. L'annexe A rappelle les informations indispensables qui doivent être indiquées. Pour certains produits complexes, une présentation orale peut être justifiée.

2. Dans un second temps, on précisera les fonctions de sécurité et le niveau de résistance visé. On précisera notamment, pour chaque transmission, les propriétés attendues, qu'elles soient réalisées par des mécanismes cryptographiques (confidentialité, intégrité) ou non (détection et correction d'erreur, protection contre le rejeu ou le déni de service,...).

3. On décrira ensuite **les mécanismes cryptographiques réalisant les fonctions de sécurité**. Nous ne donnons volontairement aucune définition formelle de ce qu'est un mécanisme cryptographique mais ce terme doit s'entendre au sens large. On attend donc la description précise des algorithmes et protocoles utilisés pour assurer la confidentialité, l'intégrité et l'authentification des données ou d'entités mais également les procédures de génération et de gestion des clés, les modes opératoires, les procédures de révocation,... **Nous ajoutons également à cette liste classique les primitives de générations et de test de données aléatoires**, que l'on ne peut certes pas rigoureusement qualifier de cryptographiques mais dont l'importance est cruciale lors de l'analyse des mécanismes de sécurité.

4. Les mécanismes cryptographiques doivent être décrits de manière suffisamment précise pour permettre leur implantation. Des exemples de description sont fournis en annexe de ce document (voir annexes D et E). Il est cependant inutile de décrire des primitives lorsque l'on peut faire référence à un document public contenant une description précise (voir annexe C). On veillera cependant dans ce cas à être très précis dans la référence.

5. La nature et le format des données traitées doivent être clairement spécifiés. On décrira donc les formats de codage utilisés, les éventuels entêtes et paddings ainsi que leur espace de définition (messages de taille fixe ou quelconque, messages formatés, commandes de carte,...). L'annexe B fournit des exemples de conventions et de notations classiques.

6. Si les mécanismes emploient des constantes, on justifiera leur génération en expliquant

par exemple comment elles ont été dérivées de constantes mathématiques, afin de justifier qu'elles n'ont pas été choisies afin d'introduire volontairement des faiblesses, ou bien obtenues par un processus d'optimisation. On fournira de plus ces données sur support informatique.

7. On précisera également l'architecture de gestion des clés (usage, cycle de vie, étude d'impact de chaque clé, ...). On fournira en particulier une idée précise du dimensionnement du système (nombre de paquets chiffrés avec une même clé de session par exemple).

8. Si l'on attend de l'implantation des mécanismes cryptographiques une résistance aux attaques par canaux auxiliaires (Timing attacks, SPA, DPA, DFA,...), une description complète et détaillée des contre-mesures mises en œuvre doit être fournie.

9. Les fournitures cryptographiques demandées sont purement descriptives. Il n'est demandé aucune justification des choix effectués, par exemple en terme d'algorithmes ou de taille de clés. De telles informations de conception sont cependant les bienvenues. L'annexe G résume les arguments permettant de justifier les choix cryptographiques.

10. La forme des fournitures est volontairement libre afin de permettre au rédacteur de choisir le moyen le plus pertinent de fournir l'information nécessaire à l'analyse des mécanismes cryptographiques. De nombreuses recommandations sont cependant annexées à ce document afin de l'y aider.

Il va de soi que la qualité des fournitures transmises conditionne directement les délais d'analyse des mécanismes cryptographiques par la DCSSI.

La suite de ce document est constituée d'**annexes**, purement informatives, dont l'objet est de faciliter la rédaction des fournitures demandées pour analyser des mécanismes cryptographiques. Les exemples proposés sont le plus souvent incomplets ou simplifiés ; ils ne pourront donc en aucun cas servir de description de référence. De plus, certains mécanismes, choisis pour leur simplicité, sont fournis exclusivement à titre d'exemple de présentation. La DCSSI ne recommande en aucun cas leur utilisation.

Information required for the analysis of cryptographic mechanisms by DCSSI

In order to analyze the strength and the suitability of cryptographic mechanisms used in security products, the DCSSI requires a technical document written according to the following rules :

1. *Firstly, this document must describe all the functionalities of the product, without restricting to cryptographic aspects. This part must be as* ***short*** *and* ***easy to read*** *as possible. The description must be complete, whatever the security target may be. The aim is to enable an overall understanding of what the product is intended to do. Appendix A reminds the information that has to be done. For complex products, a presentation talk may be useful.*

2. *Secondly, the security functions and the aimed security level are detailed. All the properties of each transmission must be described, the cryptographic ones (confidentiality, integrity) and the others (error detection and correction, disponibility, protection against replay attacks,...).*

3. *Then* **the cryptographic mechanisms that implement the security functions** *are detailed. We do not give any formal definition of what is a cryptographic mechanism, in order to avoid restrictions. We require a precise description of the algorithms and protocols used to provide confidentiality, integrity, authentication of data or entities, but also how the keys are generated and used, the modes of operation, revocation procedures,...* **We also add to this classical list the mechanisms used to generate and test random data**, *which are not exactly cryptographic mechanisms but that have crucial importance in security analysis.*

4. *The cryptographic mechanisms must be described in such a precise way that an implementation may be possible. Examples of such descriptions are proposed in appendix D and E. It may be useless to remind well-known mechanisms described in open documentation such as standards (see appendix C). However, the reference to such documentation must be extremely precise.*

5. *Any kind of manipulated data must be clearly specified. Precise details on encoding formats, headings and paddings, kind of messages (fix of variable length, formated messages, smart-card commands,...) must be provided. Appendix B gives examples of classical notations.*

6. *If constant data are used, their generation must be explained, for example giving how they have been derived from mathematical constants, in order to show that they do not add any intentional weakness. Those data must also be supplied in numerical format.*

7. *The key infrastructure (key usage, life cycle, impact of each key, …) must be described. Some ideas on the size of the system must be precised (amount of data enciphered with the same key for example).*

8. *If the mechanisms implementations are intended to defeat side-channel attacks (Timing attacks, SPA, DPA, DFA,...), a complete description of countermeasures must be given.*

9. *The document we ask is just a description of mechanisms. We do not require any justification, for example about the choice of algorithms and protocols, or about the size of the keys. Such information may however be given. Appendix G explains which kind of argument may be developed.*

10. *We do not prescribe any precise organization of the document we require, in order to make the writter free to choose any kind of appropriate presentation.*

It must be obvious that the quality of the description of cryptographic mechanisms has a direct impact on the evaluation time.

The sequel of this document is made of **appendix***, written in french. They are supplied to help the redaction of the documents but are not mandatory. Furthermore, the examples have been selected for their shortness and simplicity. They are often over-simplified and incomplete, and sometimes cryptographically weak. The DCSSI do not advice the use of those mechanisms.*

A Description générale d'un produit

La description d'un produit permet de situer les mécanismes cryptographiques dans leur contexte d'emploi. Cette description est fondamentale car elle permet de juger de l'impact d'éventuelles faiblesses cryptographiques sur la sécurité globale du produit. En cas de doute sur le contexte d'emploi d'une primitive cryptographique, l'analyse de sécurité est effectuée en faisant les hypothèses les plus contraignantes.

La description générale doit être minimale mais suffisante, la juste mesure étant laissée à l'intelligence du rédacteur. Elle doit en particulier décrire :

- **les acteurs**, c'est-à-dire l'ensemble des participants pris au sens large (autorités, utilisateurs, attaquants...). On s'attachera à décrire clairement le rôle de chacun, ses privilèges, ses capacités de calcul ainsi que les données qu'il conserve (clés secrètes, privées, publiques, certificats,...)
- **le réseau**, c'est-à-dire les moyens de communications entre acteurs. On précisera en particulier les propriétés recherchées (confidentialité, intégrité,...) ainsi que les possibilités d'attaque envisageables (écoute passive ou attaque active, attaques à clairs connus, à chiffrés choisis,...).
- **l'initialisation du système**, consistant le plus souvent à distribuer des données à chaque participant (notamment des clés et des certificats).
- **le fonctionnement du système**, i.e. les différentes phases (authentification, établissement de tunnel,...) et états du système ou de ses composantes. On décrira le mode normal de fonctionnement mais également, et c'est là une tâche souvent plus ardue, ce qui se produit en cas de problème (détection de défauts d'intégrité ou d'authentification par exemple).
- **la gestion des clés**, i.e. les différents éléments permettant de suivre l'ensemble des clés du système tout au long de leur cycle de vie. Ces éléments spécifiques pourront faire l'objet d'un document séparé, en tant que de besoin. Pour chaque clé, on précisera son usage, son cycle de vie (demande, génération, affectation, introduction, utilisation, fin de vie, renouvellement et éventuellement recouvrement), les environnements de confiance qui l'utilisent ou la stockent, les mécanismes de sécurité utilisés pour son transport ou son stockage, son étude d'impact comprenant l'estimation des différentes durées associées (cryptopériode, durée d'utilisation durée d'impact, durée d'archivage).

B Notations et conventions

La représentation des données, généralement des entiers ou des chaînes de symboles en cryptographie, peut parfois prêter à confusion. Voici une liste, non exhaustive, de recommandations visant à éviter toute ambiguïté :

- Les entiers représentés dans une base non décimale doivent être clairement identifiés, par exemple en utilisant une fonte particulière et/ou une syntaxe semblable à celle du langage C.

  Exemple.
  x=`0xAE0B77`

- Les entiers notés en binaire ou en hexadécimal seront dans la mesure du possible notés, de gauche à droite, des poids forts vers les poids faibles afin de rendre non ambigus des termes tels que “*bit de gauche*” ou “*bit de poids fort*”.
- On précisera les procédures de conversion entre différentes représentations. On veillera tout particulièrement à éviter toute ambiguïté dans la conversion entre octets et mots machine selon l’une des conventions *little-endian* ou *big-endian*.

Exemple. [Conversion octet/mot de 32 bits]
Soit B_1, B_2, B_3 et B_4 quatre octets (8 bits) consécutifs que l'on souhaite interpréter comme un mot de 32 bits de valeur numérique W. Selon la convention *little-endian*,

$$W = 2^{24}B_4 + 2^{16}B_3 + 2^8B_2 + B_1$$

alors que selon la convention *big-endian*,

$$W = 2^{24}B_1 + 2^{16}B_2 + 2^8B_3 + B_4$$

– Toute notation mathématique qui n'est pas standard et communément reconnue devra être définie avec précision.

Exemple.
On note $\mathbb{Z}_{2^{16}}$ l'ensemble des entiers 0 à $2^{16} - 1$ inclus
On note $a \oplus b$ d'addition bit à bit des mots a et b modulo 2.

– On évitera dans la mesure du possible des notations pouvant prêter à confusion comme par exemple appeler p un module RSA ou S un paramètre public.

C Description d'un algorithme

C.1 Algorithmes standard

Lorsqu'un mécanisme cryptographique est standard et clairement décrit dans la littérature scientifique ouverte, il est inutile de le décrire en détail. Il est cependant indispensable d'être particulièrement précis dans les références afin d'éviter toute ambiguïté. Il faut en particulier préciser l'algorithme proprement dit, ses modes opératoires, la taille des clés et leur mode de génération... On évitera donc les mentions du genre "*on utilisera RSA*" car ceci n'indique pas s'il s'agit de chiffrement ou de signature, quelle est la taille des clés, quel padding est employé,...

Au contraire, indiquer l'emploi de *"l'algorithme RSA décrit dans le standard PKCS#1 version 2.1 sous l'appellation RSAES-OAEP avec la fonction de génération de masque MGF1 utilisant SHA-1 et des modules de 2048 bits"* dispense de toute autre indication liée à la description de l'algorithme. Seuls devront encore être précisés le mode de génération et la gestion des clés. Il est important de réaliser qu'aucune des information fournies dans cet exemple n'est superflue s'il l'on garde à l'esprit que la référence doit être suffisamment précise afin d'implanter le mécanisme cryptographique en question.

Afin de citer un autre exemple, la référence à un MAC au standard ISO 9797 pourra se faire de la manière suivante :*"on utilisera un MAC suivant la norme ISO/IEC 9797-1 utilisant l'algorithme de chiffrement par bloc DES, la méthode de padding numéro 2, l'algorithme 3 avec des clés K et K' indépendantes et sans troncature finale"* et non par une vague indication du genre *"on utilisera un CBC-MAC"*.

C.2 Algorithmes non standard

Dans le cas de mécanismes originaux, il est nécessaire de les décrire en détail. Avant de décrire précisément différents exemples de primitives cryptographiques (voir annexe D), nous proposons ici une manière unifiée de décrire un algorithme, i.e. une suite d'instructions élémentaires permettant, à partir de données fournies en entrée, de produire un résultat. Toute description d'algorithme suivra, dans la mesure du possible, le schéma général suivant :

- **Description générale des objectifs de l'algorithmique** : notamment dans le cas d'applications originales ou peu courantes, on décrira tout d'abord le but et la sécurité recherchée de l'algorithme proposé.
- **Description des données en entrée** : on fournira la liste des données utilisées par l'algorithme en fixant leur notation, leur taille ainsi que leur format. D'éventuelles hypothèses sur ces données pourront être précisées.
- **Description du résultat** : on décrira précisément le résultat du calcul, i.e. ce qui est calculé et renvoyé par l'algorithme ainsi que la taille et le format de cette sortie.
- **Description des constantes** : certains algorithmes utilisent des tables et des constantes qu'il est souvent bon de définir séparément afin de clarifier l'exposé.
- **Instructions de calcul** : on décrira de manière aussi précise que nécessaire afin de lever toute ambiguïté les opérations élémentaires à réaliser ainsi que les éventuels appels à d'autres primitives. Dans ce dernier cas, le format d'appel de ces primitives devra être rappelé et scrupuleusement respecté.

  Voici quelques recommandations complémentaires :
  - Les calculs ne doivent faire intervenir que des données fournies en entrée de l'algorithme, en respectant les notations.
  - L'algorithme doit clairement faire apparaître les sorties.
  - Tout moyen complémentaire de description peut être utilisé afin d'aider à la compréhension du mécanisme mais ne doit en aucun cas servir à lui seul de description (schéma montrant l'organisation du calcul sous forme d'organigramme, extraits de code source dans un langage largement répandu comme le C,...)
  - Des vecteurs de test montrant l'exécution du calcul de la manière la plus détaillée possible aideront à lever toute ambiguïté résiduelle mais ne peuvent en aucun cas servir à définir le fonctionnement d'un algorithme. On pourra de plus présenter un exemple "jouet" s'il est facile de réduire la taille des paramètres, comme c'est généralement le cas pour les algorithmes cryptographiques à clé publique.

  Exemple. [Algorithme d'Euclide]

  • Syntaxe : $\texttt{Pgcd}(a, b)$

  • Entrée :
  – un entier a strictement positif,
  – un entier b positif et inférieur à a. ($0 \leq b \leq a$)

  • Sortie : le plus grand commun diviseur (PGCD) de a et b.

  • Primitives appelées :
  – $\texttt{mod}(x, y)$ retourne un entier compris dans l'intervalle $[0, y-1]$ égal au reste de la division euclidienne de l'entier x par l'entier y.

• Instructions de calcul :

1. $x := a$,
2. $y := b$,
3. tant que $y \neq 0$ faire
   3.1. calculer $r := \texttt{mod}(x, y)$,
   3.2. $x := y$,
   3.3. $y := r$,
4. **retourner** x.

• Représentation schématique : voir figure 1.

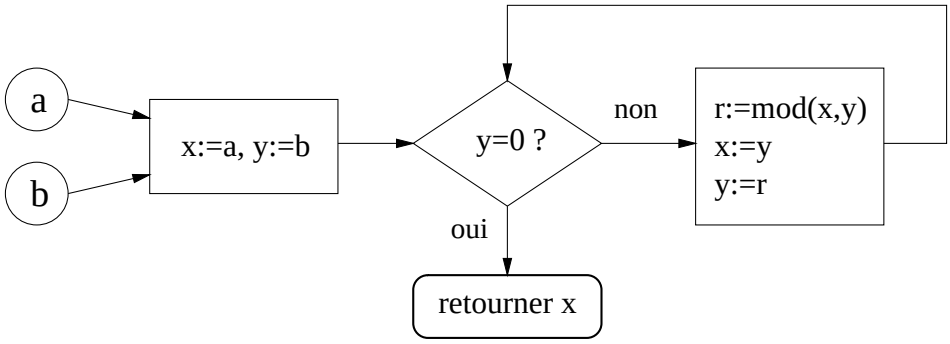


FIG. 1 – Algorithme d'Euclide : exemple de représentation schématique d'algorithme

• Exemple d'exécution : $\texttt{Pgcd}(91, 70)$:

Évolution des variables :

x	y	r
91	70	21
70	21	7
21	7	0
7	0	

D Exemples de descriptions des principales primitives cryptographiques

Cette section a pour but de préciser, pour chacune des principales primitives cryptographiques, les informations qu'il convient de préciser lors de leur description. Les exemples fournis ont pour but d'illustrer les recommandations données de manière plus concrète.

Mise en garde : Les exemples proposés sont le plus souvent incomplets ou simplifiés ; ils ne pourront donc en aucun cas servir de description de référence. De plus, certains mécanismes, choisis pour leur simplicité, sont présentés exclusivement afin de donner des exemples de présentation. Le laboratoire de cryptographie ne recommande en aucun cas leur utilisation.

D.1 Générateur pseudo-aléatoire

Un générateur pseudo-aléatoire est un algorithme recevant en entrée une chaîne de bits (*seed*) de longueur fixe ou variable permettant son initialisation et fournissant en sortie une suite de bits dont la distribution semble aléatoire, cette suite étant calculée de manière déterministe une fois le générateur initialisé.

On décrira donc un générateur pseudo-aléatoire en fournissant :
- un éventuel algorithme permettant de choisir des paramètres généraux,
- un algorithme générant une suite pseudo-aléatoire de bits à partir d'une chaîne de bits d'initialisation. Les entrées d'un tel algorithme sont en général composées de :
  - la taille de la chaîne d'initialisation,
  - la chaîne d'initialisation proprement dite,
  - du nombre de bits générés.

  La sortie peut être soit la suite des bits générés, soit un message d'erreur.

Exemple. [Générateur pseudo-aléatoire de Blum-Blum-Shub]

Algorithme de choix des paramètres

• Syntaxe : `BBS-param`(k)

• Entrée : un entier k strictement positif.

• Sortie : un entier n produit de deux nombres premiers de k bits congrus à 3 modulo 4.

• Primitives appelées :
– `isprime`(x) retourne `vrai` si l'entier x est premier et `faux` autrement.

• Instructions de calcul :

1. choisir aléatoirement un entier α dans l'intervalle $[0, 2^{k-2} - 1]$,
2. calculer $p := 3 \times \alpha + 4$,
3. si `isprime`$(p) =$ `faux` recommencer à l'étape 1,
4. choisir aléatoirement un entier β dans l'intervalle $[0, 2^{k-2} - 1]$,
5. calculer $q := 3 \times \beta + 4$,

6. si `isprime`$(q) =$ `faux` recommencer à l'étape 4,
7. calculer $n := p \times q$,
8. **retourner** n.

Algorithme de génération d'une suite pseudo-aléatoire

• Syntaxe : `BBS`(n, s, ℓ)

• Entrée :
– un entier n généré par `BBS-param`,
– un entier s tel que $1 \leq s < n$, premier avec n, permettant d'initialiser le générateur et devant être gardé secret,
– un entier positif ℓ indiquant le nombre de bits à générer.

• Sortie : une chaîne de ℓ bits.

• Primitives appelées :
– `sqmod`(x, n) retourne un entier égal au carré de x modulo n.

• Instructions de calcul :

1. $x :=$ `sqmod`(s, n),
2. pour i variant de 1 à ℓ faire
   2.1. $x :=$ `sqmod`(x, n),
   2.2. soit z_i le bit de poids faible de x, i.e. $z_i := x \bmod 2$,
3. **retourner la suite de ℓ bits $\mathbf{z_1}, ...\mathbf{z_\ell}$.**

D.2 Fonction de hachage

Une fonction de hachage (sans clé) permet de générer à partir d'une chaîne quelconque une valeur hachée de taille fixée. Les caractéristiques d'une telle fonction sont donc :
– la taille des valeurs hachées générées (128 ou 512 bits en général),
– une éventuelle taille minimale et/ou maximale des données hachées.

On veillera tout particulièrement à décrire sans ambiguïté les procédures de conversion de suites de bits ou d'octets en entrée de la fonction de hachage selon l'une des conventions *little-endian* ou *big-endian*.

Exemple. [SHA-1]

• Syntaxe : `SHA-1`(M)

• Entrée : M une chaîne de m bits ($0 \leq m < 2^{64}$).

• Sortie : une chaîne de **160 bits**.

• Primitives appelées :
– `rotg`(x, n) effectue une rotation de n bits vers la gauche du mot x de 32 bits.

• Définition de constantes :
– 5 mots de 32 bits :
$h_1 =$ `0x67452301`, $h_2 =$ *... (omis dans cet exemple)*

– 4 mots de 32 bits :
$y_1 =$ `0x5A827999`, $y_2 =$ ... *(omis dans cet exemple)*

• Instructions de calcul :

1. Padding du message M de longueur m (voir figure 2) :
   1.1. soit $r = 512 - (m + 64 \bmod 512)$,
   1.2. ajouter au message M un bit à 1 suivi de $r - 1$ bits à 0,
   1.3. ajouter ensuite une représentation sur 64 bits de la longueur initiale m du message à hacher, le mot de poids fort de 32 bits étant placé avant celui de poids faible,
   1.4. en utilisant la convention *big-endian*, convertir le message ainsi obtenu en $16 \times k$ mots de 32 bits notés $x_0, ...x_{16k-1}$,

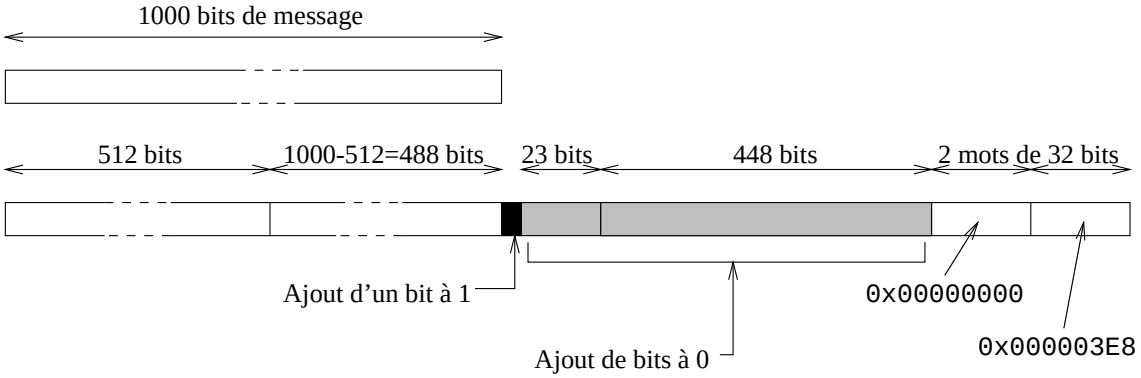


FIG. 2 – Padding de message pour SHA-1

2. initialiser $H_1 := h_1$, $H_2 := h_2$, $H_3 := h_3$, $H_4 := h_4$, $H_5 := h_5$,
3. Pour i variant de 0 à $k - 1$ faire
   3.1. Pour j variant de 0 à 15 faire $X_j := x_{16i+j}$,
   3.2. Pour j variant de 16 à 79 faire
   $X_j :=$ `rotg`$(X_{j-3} \oplus X_{j-8} \oplus X_{j-14} \oplus X_{j-16}, 1)$,
   3.3. initialiser $A := H_1$,$B := H_2$,$C := H_3$,$D := H_4$,$E := H_5$
   3.4. Pour j variant de 0 à 19 faire
      3.4.1. $t :=$ `rotg`$(A, 5) + f(B, C, D) + E + X_j + y_1$
      avec $f(u, v, w) = (u$ `et` $v)$ `ou` $($`non`(u) `et` $w)$
      3.4.2. $A := t$, $B := A$, $C :=$ `rotg`$(B, 30)$, $D := C$, $E := D$
      ... (suite du calcul omise dans cet exemple) ...
4. **retourner le mot de 160 bits formé par la concaténation de $\mathbf{H_1}$, $\mathbf{H_2}$, $\mathbf{H_3}$, $\mathbf{H_4}$ et $\mathbf{H_5}$ dans cet ordre.**

• Représentation schématique : *(omis dans cet exemple mais fortement souhaitable en pratique)*

• Code source C : *à fournir sur support informatique*

• Vecteurs de test :

message M	SHA-1(M)
“”	0xDA39A3EE5E6B4B0D3255BFEF95601890AFD80709
“abc”	0xA9993E364706816ABA3E25717850C26C9CD0D89D
...	*autres vecteurs couvrant les différents cas possibles...*

D.3 Code d’authentification de message (MAC)

Afin de garantir l’intégrité d’un message de taille quelconque une méthode consiste à lui associer un code d’authentification (appelé *MAC*) calculé à partir de ce message et d’une clé secrète connue de l’émetteur et du destinataire du message.

Une algorithme de MAC se décompose donc en :
– un algorithme de génération de clés secrètes,
– un algorithme qui à partir d’un message de longueur quelconque et d’une clé secrète génère un code de taille fixe.

Exemple. [HMAC]
• Syntaxe : HMAC(M, K)

• Entrée :
– un message M de longueur quelconque vu comme une suite de bits,
– une clé K de k octets.

• Sortie : un code d’authentification de k octets.

• Primitives appelées :
– une fonction de hachage H générant des valeurs hachées H(M) sur k octets pour toute chaîne de bits M fournie en entrée.

• Notation :
– les doubles barres ($||$) indiquent la concaténation de chaînes de bits,
– $\oplus$ désigne le *ou-exclusif*, i.e. la somme modulo 2.

• Définition de constantes :
– ipad obtenue par k répétitions de l’octet 0x36,
– opad obtenue par k répétitions de l’octet 0x5c,

• Instructions de calcul :

1. calculer $x :=$ H$((K \oplus$ ipad$)||M)$,
2. calculer $y :=$ H$((K \oplus$ opad$)||x)$,
3. **retourner** y.

D.4 Chiffrement par flot

Un algorithme de chiffrement par flot permet de chiffrer une suite de bits ou de symboles de manière continue, par opposition avec les algorithmes de chiffrement par bloc. Un tel schéma se décompose en général en :
– un algorithme de génération de clés secrètes de chiffrement et de déchiffrement,

– un algorithme de génération d'un flot de bits de clé à partir d'une clé de taille fixe ; cet algorithme peut se décomposer en une phase d'initialisation suivie d'une phase de génération proprement dite,
– un mécanisme de chiffrement combinant le flot des bits de clé avec le flot des bits de message.

On pourra de même décrire le déchiffrement associé mais en général il suffira de préciser les différences, souvent minimes, entre chiffrement et déchiffrement.

Exemple. [LFSR (Linear Feedback Shift Registers)[1]]

• Syntaxe : $\mathsf{LFSR}(\ell, K, D, delay, M)$

• Entrée :
– un entier ℓ,
– une suite K de ℓ bits (clé) indicés de 1 à ℓ,
– une suite C de ℓ bits (coefficients du LFSR) indicés de 1 à ℓ,
– un entier $delay$ indiquant le nombre de tours d'initialisation,
– une suite M de bits de longueur m quelconque.

• Sortie : une suite de bits C de même longueur que le message M, chiffrée avec la clé K.

• Notation : $\oplus$ désigne le *ou-exclusif*, i.e. la somme modulo 2.

• Instructions de calcul :

1. pour i variant de 1 à ℓ faire $X_i := K_i$
2. pour j variant de 1 à $delay$ faire
   2.1. $Y := \bigoplus_{i=1}^{\ell} C_i \times X_i$,
   2.2. pour i variant de 1 à $\ell - 1$ faire $X_i := X_{i+1}$,
   2.3. $X_\ell := Y$,
3. Pour j variant de 1 à m faire
   3.1. $Y := \bigoplus_{i=1}^{\ell} C_i \times X_i$,
   3.2. pour i variant de 1 à $\ell - 1$ faire $X_i := X_{i+1}$,
   3.3. $X_\ell := Y$,
   3.4. $C_i := M_i \oplus X_0$,
4. **retourner la chaîne $\mathbf{C}$ de $\mathbf{m}$ bits**.

• Représentation schématique : voir figure 3.

• Exemple d'exécution (minimaliste) :
évolution de la sortie du LFSR de la figure 3 initialisé avec la clé $K = 1110$ ($K_4 = 1$, $K_3 = 1$, $K_2 = 1$, $K_1 = 0$) :

[1] La description du chiffrement par flot à base de LFSR n'est qu'un exemple. Il est bien entendu que la primitive décrite ici n'est absolument pas sûre.

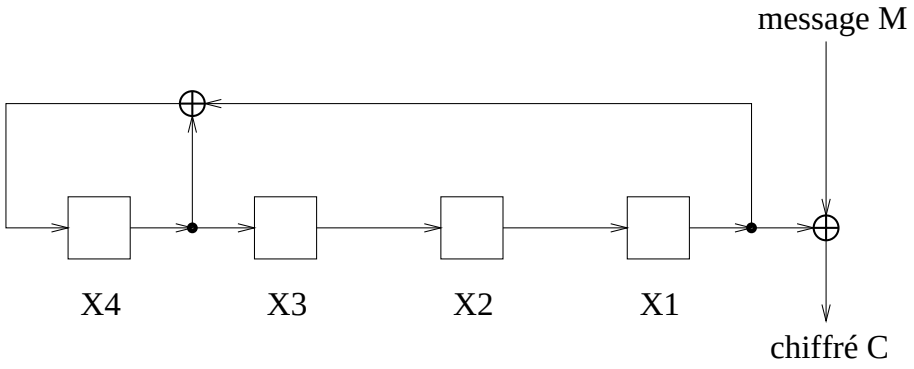


FIG. 3 – Exemple de LFSR de longueur $\ell = 4$, avec $C_1 = C_4 = 1$ et $C_2 = C_3 = 0$

X_4	X_3	X_2	X_1	sortie
1	1	1	0	
1	1	1	1	0
0	1	1	1	1
1	0	1	1	1
0	1	0	1	1
1	0	1	0	1
1	1	0	1	0
0	1	1	0	1

X_4	X_3	X_2	X_1	sortie
0	0	1	1	0
1	0	0	1	1
0	1	0	0	1
0	0	1	0	0
0	0	0	1	0
1	0	0	0	1
1	1	0	0	0
1	1	1	0	0

Chiffrement de $M = 10011011$ si $delay = 6$:

bits de clé	0	1	1	1	1	0	1	0	1	1	0	0	1	0	0
message							1	0	0	1	1	0	1	1	
chiffré							**0**	**0**	**1**	**0**	**1**	**0**	**0**	**1**	

D.5 Chiffrement symétrique par bloc

Par opposition avec le chiffrement par flot, le chiffrement par bloc permet de chiffrer et de déchiffrer des messages vus comme une suite de blocs de taille fixe, et ce avec une unique clé secrète connue de l'émetteur et du destinataire.

La description d'un schéma de chiffrement par bloc se compose en général :
- d'un algorithme de génération de clés,
- d'un algorithme de chiffrement dont on décrira en particulier :
  - l'éventuelle initialisation,
  - les éventuelles tables et données constantes employées,
  - l'algorithme de diversification des clés (*key schedule*),
  - le chiffrement d'un bloc de message,
  - le déchiffrement d'un bloc de message,
- éventuellement le ou les modes d'opération (ECB,CBC,CFB,OFB, ou autre...) envisagés pour chiffrer des messages de taille quelconque. On précisera alors le padding utilisé.

Exemple. [DES en mode CBC]

Génération de clés

• Syntaxe : `CLE-DES`$()$

• Entrée : rien.

• Sortie : une clé de 64 bits choisie aléatoirement parmi les 2^{56} clés possibles.

• Primitives appelées :
– `randombit`$()$ renvoie un bit aléatoire.

• Instructions de calcul :

1. Pour i variant de 0 à 7 faire
   1.1. Pour j variant de 1 à 7 faire $K_{8i+j} :=$ `randombit`$()$,
   1.2. Calculer le bit de parité
   $K_{8i+8} := K_{8i+1} \oplus K_{8i+2} \oplus K_{8i+3} \oplus K_{8i+4} \oplus K_{8i+5} \oplus K_{8i+6} \oplus K_{8i+7}$,
2. **retourner** $\mathbf{K_1..K_{64}}$.

Chiffrement DES d'un bloc

• Syntaxe : `DES`(K, M)

• Entrée :
– une clé K générée par `CLE-DES`$()$,
– un bloc M de 64 bits.

• Sortie : un bloc chiffré de 64 bits.

• Définition de constantes :
– $PC1$ une injection de $\{1..64\}$ dans $\{1..56\}$ définie par
$PC1(1) = 57$, $PC1(2) = 49$,... $PC1(64) = 4$ (*à compléter*),

– $PC2$ une injection de $\{1..56\}$ dans $\{1..48\}$ définie par $PC2(1) = 14$, $PC2(2) = 17$,... $PC2(56) = 32$ (*à compléter*),
– *... on omet dans cet exemple la description de la permutation initiale IP, de la fonction d'expansion E ainsi que de la permutation interne P ...*
– huit fonctions S_1,...S_8 de $\{0,1\}^6$ dans $\{0,1\}^4$ définies par $S_1(000000) = 1110$, $S_1(000001) = 0000$,... $S_8(111111) = 1011$.
(*une description réelle fournirait bien entendu l'intégralité de la description des boites S, de préférence à l'aide d'une représentation intelligente sous forme de tableau*).

• Instructions de calcul :

...

... omis dans cet exemple

...

• Représentation schématique :
la description d'un algorithme aussi complexe que DES doit bien entendu être illustrée des schémas les plus pertinents possibles afin de clarifier son fonctionnement.

• Vecteurs de test :
– $K =$ `0x0123456789ABCDEF`
– $M =$ `0x4E6F772069732074`
– `DES`$(K, M) =$ `0x3FA40E8A984D4815`

Chiffrement DES en mode CBC

• Syntaxe : `DES-CBC`(K, M)

• Entrée :
– une clé K générée par `CLE-DES`$()$,
– un message M de $64 \times m$ bits.

• Sortie : un message chiffré C de $64\times(m+1)$ bits dont le premier bloc représente l'IV transmis en clair.

• Primitives appelées :
– `randombit`$()$ renvoie un bit aléatoire,
– `DES`(K, M) chiffre un bloc M de 64 bits avec une clé K générée par `CLE-DES`$()$.

• Instructions de calcul :

1. Pour i variant de 1 à 64 faire $IV_i :=$ `randombit`$()$,
2. $C_0 := IV$,
3. Pour i variant de 1 à m faire $C_j :=$ `DES`$(K, C_{j-1} \oplus M_j)$,
4. **retourner** $\mathbf{C} := \mathbf{C_0}...\mathbf{C_m}$.

• Représentation schématique : voir figure 4.

• Vecteurs de test : *omis dans cet exemple mais indispensables*

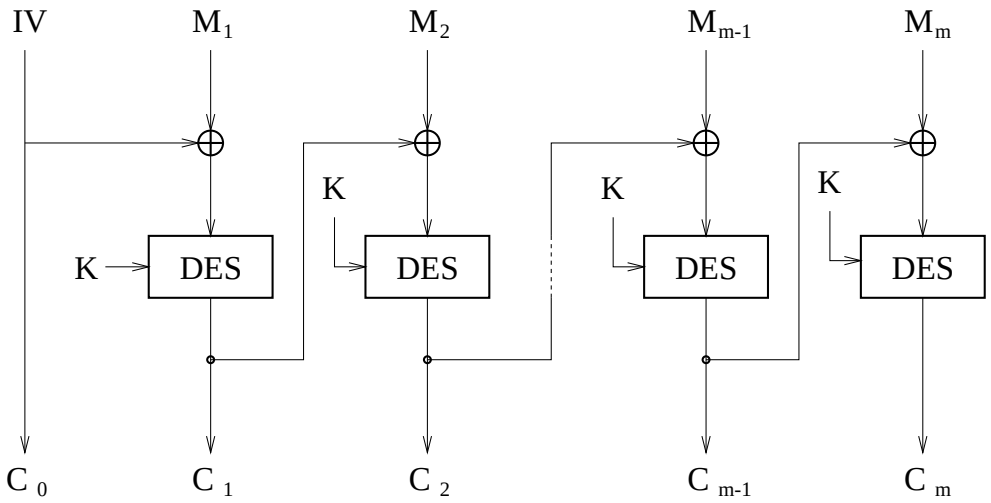


FIG. 4 – Chiffrement DES en mode CBC

D.6 Authentification mutuelle par mécanisme symétrique

Les protocoles d'authentification mutuelle par mécanisme symétrique permettent à deux interlocuteurs partageant une même clé secrète de s'authentifier l'un l'autre. Ces protocoles ont l'avantage de n'utiliser que des mécanismes cryptographiques symétriques. Les mécanisme asymétriques, tels que ceux évoqués dans la section D.10, les remplacent cependant avantageusement.

La description de tels protocoles devra donc tout particulièrement détailler :

- la description précise des différents **participants** et notamment de la confiance nécessaire entre chacun d'entre eux,
- description des **moyens de communication** entre participants,
- description des **données** initialement détenues par chacun ainsi que de la manière dont elles sont générées,
- description des différentes **phases de communication et de calcul** en mettant bien en évidence lors de chaque étape ce qui est transmis entre les intervenants et les données finalement partagées.

Exemple. [Protocole d'authentification mutuelle ISO 9798-2]

• Participants : 2 personnes notées A et B.

• Réseau : les participants sont reliés par une liaison non sécurisée, i.e. susceptible d'être écoutée mais également totalement contrôlée par un adversaire.

• Primitives appelées :

- un algorithme de chiffrement symétrique noté $E_K(M)$ chiffrant des messages M de taille quelconque avec une clé K,
- un processus de génération de données aléatoires `rand()` fournissant 128 bits aléatoires par appel.

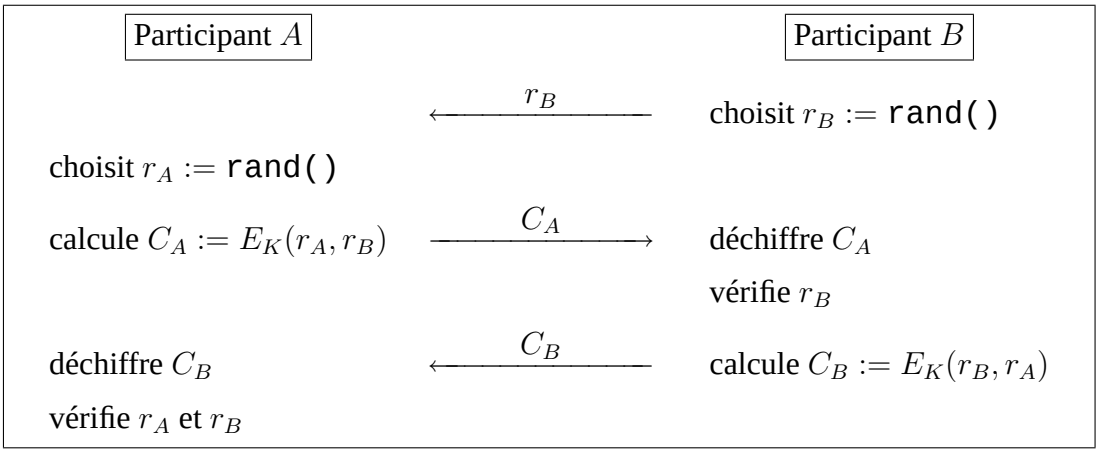
• Données initiales :

- une clé secrète K partagée par A et B.

• Protocole :

1. B choisit aléatoirement r_B par appel `rand()`, et le transmet à A,
2. A choisit aléatoirement r_A par appel `rand()`, calcule $C_A := E_K(r_A, r_B)$ et le transmet à B,
3. B déchiffre C_A, vérifie r_B, calcule $C_B := E_K(r_B, r_A)$ et le transmet à A,
4. A déchiffre C_B, vérifie r_A et r_B.

• Représentation schématique :



D.7 Mise en accord de clé

Les protocoles de transport ou de mise en accord de clé permettent à deux interlocuteurs de partager une clé pouvant ensuite être utilisée afin de chiffrer leurs communications. Ces protocoles sont nombreux, basés soit sur des mécanismes symétriques, soit sur des mécanismes à clé publique. Ils peuvent faire intervenir des tiers de confiance, permettre une authentification des participants,...

La description de tels protocoles devra donc tout particulièrement détailler :

– la description précise des différents **participants** et notamment de la confiance nécessaire entre chacun d'entre eux,
– description des **moyens de communication** entre participants,
– description des **données** initialement détenues par chacun ainsi que de la manière dont elles sont générées,
– description des différentes **phases de communication et de calcul** en mettant bien en évidence lors de chaque étape ce qui est transmis entre les intervenants et les données finalement partagées.

Exemple. [Protocole de mise en accord de clé de Diffie-Hellman[2]]

• Participants : 2 personnes notées A et B.

[2]Ce schéma n'est décrit qu'à titre d'exemple. En particulier, il ne garantit nullement l'authentification des participants et est vulnérable à des attaques par le milieu.

• Réseau : les participants sont reliés par une liaison non sécurisée, i.e. susceptible d'être écoutée mais également totalement contrôlée par un adversaire.

• Données initiales :
– p un nombre premier,
– g un générateur du groupe multiplicatif des entiers inversibles modulo p ($2 \leq g \leq p-2$).

• Primitives appelées :
– une fonction de hachage H générant des valeurs hachées $\mathsf{H}(M)$ sur h bits pour toute chaîne de bits M fournie en entrée.

• Protocole :

1. A choisit un entier s_A dans l'intervalle $[1, p-2]$,
2. A calcule $P_A := g^{s_A} \bmod p$ et le **transmet à B**,
3. B choisit un entier s_B dans l'intervalle $[1, p-2]$,
4. B calcule $P_B := g^{s_B} \bmod p$ et le **transmet à A**,
5. A calcule la clé partagée $K := H(P_B^{s_A} \bmod p)$,
6. B calcule la clé partagée $K := H(P_A^{s_B} \bmod p)$,
7. **La clé partagée par A et B est K**.

• Représentation schématique :

Paramètres :	p un nombre premier g un élément d'ordre $p-1$ modulo p	
Participant A		Participant B
choisit $s_A \in [1, p-2]$		choisit $s_B \in [1, p-2]$
calcule $P_A := g^{s_A} \bmod p$	$\xrightarrow{\quad P_A \quad}$	
	$\xleftarrow{\quad P_B \quad}$	calcule $P_B := g^{s_B} \bmod p$
calcule $K := H(P_B^{s_A} \bmod p)$		calcule $K := H(P_A^{s_B} \bmod p)$

• Exemple d'exécution (minimaliste) : soit $p = 97$ et $g = 5$.
– A choisit $s_A = 19$; $P_A = 5^{19} = 38 \bmod p$,
– B choisit $s_B = 28$; $P_B = 5^{28} = 73 \bmod p$,
– la clé secrète partagée est $K = P_A^{28} = 54 = P_B^{19} \bmod p$.

D.8 Chiffrement à clé publique

La description d'algorithmes de chiffrement à clé publique se décompose principalement en :
– un algorithme de génération de paramètres globaux (courbe elliptique, modules,...),
– un algorithme de génération de clés secrètes et de clés publiques associées,

– un algorithme de chiffrement,
– éventuellement une méthode de padding permettant de spécifier précisément le chiffrement d'un message quelconque,
– un algorithme de déchiffrement.

Exemple. [RSA/OAEP (PKCS#1-v.2)]

Génération de clés

• Syntaxe : RSA-key(k, e)

• Entrée :
– un entier k indiquant la taille du module,
– une clé publique e (entier de moins de k bits).

• Sortie :
– une module RSA n de k bits,
– une clé secrète d de k bits.

• Instructions de calcul :

1. choisir un nombre premier p inférieur à $2^{k/2}$,
2. choisir un nombre premier q inférieur à $2^{k/2}$,
3. calculer $n := p \times q$,
4. si pgcd$((p-1)(q-1), e) \neq 1$ retourner en 1,
5. calculer $d := e^{-1} \bmod (p-1)(q-1)$,
6. **retourner** $\mathbf{n}$ **et** $\mathbf{d}$.

Notons que suivant le degré de précision de la description de cet algorithme, le calcul de pgcd ainsi que l'inversion modulaire pourraient soit être spécifiquement effectués par des primitives externes, soit totalement spécifiés en indiquant chaque étape de l'algorithme d'Euclide étendu.

• Exemple d'exécution : RSA-key$(10, 17)$
– $p := 17$,
– $q := 31$,
– $n := 527$,
– $d := 113$.

Chiffrement RSA

• Syntaxe : RSA(n, e, M)

• Entrée :
– n un module RSA de k bits et e une clé publique RSA,
– M un entier inférieur à n.

• Sortie : un chiffré C du message M

• Instructions de calcul :

1. **retourner** $\mathbf{C} := \mathbf{M^e} \bmod \mathbf{n}$.

• Exemple d'exécution : RSA$(527, 17, 666)$

– $C := 666^{17} \bmod 527 = 54$.

Chiffrement RSA avec padding OAEP

• Syntaxe : $\mathsf{RSA\text{-}OAEP}(n, e, M)$

• Entrée :
– n un module RSA de k bits et e une clé publique RSA,
– M un entier de moins de $k - 256$ bits.

• Sortie : un chiffré C du message M utilisant le padding OAEP.

• Primitives appelées :
– G une fonction aléatoire associant à toute chaîne de 128 bits une chaîne de $k - 128$ bits,
– H une fonction aléatoire associant à toute chaîne de $k - 128$ bits une chaîne de 128 bits.

• Instructions de calcul :

1. choisir aléatoirement une chaîne r de 128 bits,
2. soit M' la chaîne obtenue en concaténant le message M avec 128 zéros,
3. calculer $\alpha := M' \oplus G(r)$,
4. calculer $\beta := \mathsf{RSA}(n, e, \alpha || r \oplus H(\alpha))$ (où $||$ désigne la concaténation de chaînes de bits),
5. **retourner** β.

• Représentation schématique : voir figure 5.

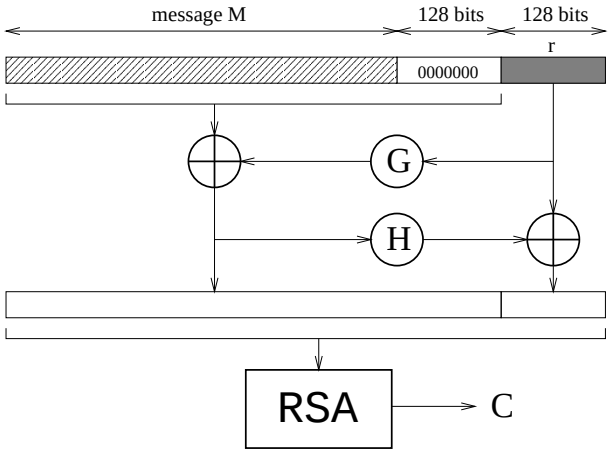


FIG. 5 – Padding OAEP

D.9 Signature

La description d'algorithmes de signature est formellement assez semblable à celle d'algorithmes de chiffrement à clé publique et se décompose principalement en :

- un algorithme de génération de paramètres globaux (courbe elliptique, modules,...),
- un algorithme de génération de clés secrètes et de clés publiques associées,
- un algorithme de signature,
- un algorithme permettant de vérifier la validité d'une signature.

D.10 Authentification interactive

Contrairement aux schémas de chiffrement et de signature, les protocoles d'authentification sont en général fondamentalement interactifs. Leur description se fera donc au moyen :

- d'un algorithme de génération de paramètres globaux,
- d'un algorithme de génération de clés secrètes et des clés publiques associées,
- d'un **protocole** interactif d'authentification au cours duquel un *prouveur* tente de convaincre un *vérifieur* de son identité.

Exemple. [Protocole d'authentification de Fiat-Shamir]

• Participants : 2 personnes, un prouveur noté P et un vérifieur V.

• Réseau : les participants sont reliés par une liaison quelconque, non sécurisée.

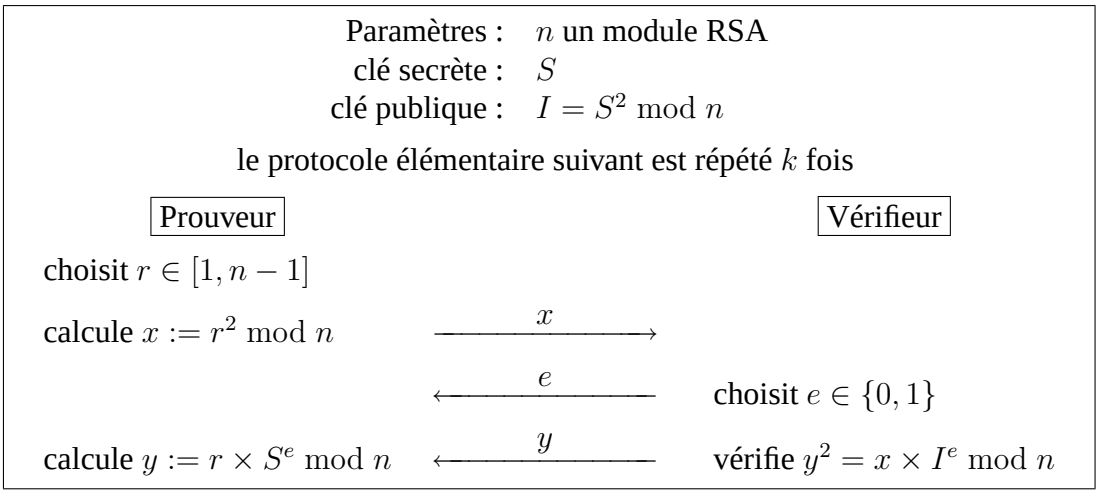
• Données initiales :

- n un entier public produit de deux grands nombres premiers dont la factorisation est inconnue de tous,
- S un entier inférieur à n et premier avec n ; S est la **clé secrète** du prouveur et doit être connue de lui seul,
- $I := S^2 \bmod n$ la **clé publique** du prouveur qui est révélée à tous,
- k un entier indiquant la sécurité requise.

• Protocole :

1. Répéter k fois
   - 1.1. P choisit un entier r dans l'intervalle $[1, n-1]$,
   - 1.2. P calcule $x := r^2 \bmod n$ et **le transmet à V**,
   - 1.3. V choisit un bit $e \in \{0, 1\}$ et **le transmet à P**,
   - 1.4. P calcule $y := r \times S^e \bmod n$ et **le transmet à V**,
   - 1.5. V refuse l'authentification si $y^2 \neq x \times I^e \bmod n$.
2. **Si la totalité des vérifications est correcte, P est correctement authentifié par V.**

• Représentation schématique :



E Description d'un protocole interactif

Afin de définir des protocoles cryptographiques faisant intervenir des calculs mais également des communications entre divers participants, on utilisera l'organisation suivante :

- description précise des différents **participants**,
- description des **moyens de communication** entre participants en précisant les hypothèses faites sur la sécurité de ces moyens (confidentialité, intégrité,...),
- description des **données** initialement détenues par chacun ainsi que de leur degré de confidentialité (données secrètes, publiques, partagées,...),
- description des différentes **phases de communication et de calcul** en mettant bien en évidence lors de chaque étape ce qui est transmis entre les intervenants.

Tout comme pour la description des algorithmes, tout moyen complémentaire est le bienvenu afin de clarifier l'explication mais ne remplace en rien la description précédente. Un schéma résumant les interactions et les données transmises ainsi qu'un exemple, éventuellement réduit, d'exécution peuvent ainsi être très utiles.

Exemple. [Partage de secret de Shamir]

• Objectif :
– Un partage de secret à seuil permet de distribuer un secret S entre n personnes de manière à ce que la collaboration d'au moins k personnes soit ensuite nécessaire afin de pouvoir reconstituer le secret en question.

• Participants :
– un distributeur D chargé de choisir les paramètres communs et de partager un secret S entre n personnes,
– n personnes $P_1, ...P_n$ recevant chacune une partie du secret de la part du distributeur.

• Réseau :
– le distributeur est relié à chaque personne par un canal sécurisé garantissant la confidentialité et l'intégrité des communications.

• Données initiales :

– S un entier **secret** connu uniquement du distributeur,
– n le nombre de personnes entre lesquelles le secret est partagé,
– k le nombre minimal de personnes devant coopérer afin de retrouver le secret S à partir de leurs parts respectives.

• Protocole :

1. D choisit un nombre premier p supérieur à S et à n qu'il rend public,
2. D choisit aléatoirement $k-1$ entiers $a_1, ...a_{k-1}$ dans l'intervalle $[0, p-1]$ qu'il garde secrets,
3. D calcule la part $b_i = S + \sum_{j=1}^{k-1} a_j \times i^j \mod p$ de chacune des n personnes,
4. **D transmet de manière confidentielle la part $\mathbf{b_i}$ à $\mathbf{P_i}$, et ce pour tout $\mathbf{i} \in [\mathbf{1}, \mathbf{n}]$.**

• Représentation schématique : voir figure 6.

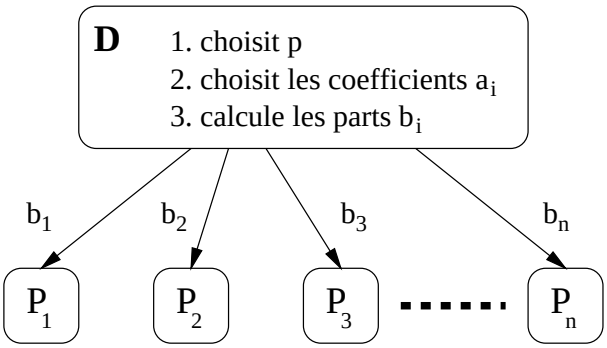


FIG. 6 – Partage de secret de Shamir : exemple de représentation schématique d'un protocole interactif

• Exemple d'exécution (minimaliste) : distribution de $S = 6$ entre $n = 3$ personnes de manière à ce que $k = 2$ d'entre elles aient à coopérer afin de retrouver le secret.
– D choisit $p = 11$ et $a_1 = 4$,
– D calcule $b_1 = 6 + 4 \times 1 = 10 \mod 11$, $b_2 = 6 + 4 \times 2 = 3 \mod 11$ et $b_3 = 6 + 4 \times 3 = 7 \mod 11$,
– D transmet b_1 à P_1, b_2 à P_2 et b_3 à P_3.

Le protocole de reconstruction de secret n'est pas décrit dans cet exemple.

F Description d'une interface de librairie

La description d'une interface pour une librairie cryptographique dont les sources ne sont pas divulguées suivra les mêmes recommandations que dans les sections précédentes, notamment en ce qui concerne les formats d'appel des différentes fonctions, les types des données manipulées, la description minutieuse des résultats renvoyés,... Seule la description des mécanismes internes sera bien entendu omise.

G Arguments liés à la sécurité

Quels que soient la primitive cryptographique ou le protocole décrit, il peut s'avérer utile de justifier les choix effectués en terme d'architecture, de choix des primitives, de taille des paramètres,... **La fourniture de ces informations n'est cependant nullement obligatoire dans le cadre de l'analyse de mécanismes cryptographiques par la DCSSI. De plus, les justifications fournies ne le sont qu'à titre informatif et en conséquence ne sauraient préjuger de l'avis final de la DCSSI.**

La liste suivante, non exhaustive, résume les principaux critères habituellement retenus afin de justifier les choix effectués :

- **Résistance face aux attaques connues**. On établira par exemple l'effort nécessaire pour attaquer un schéma de chiffrement par bloc au moyen d'une recherche exhaustive sur la clé ou bien la résistance face à des classes d'attaques classiques comme la cryptanalyse différentielle ou linéaire.
- **Justification de la taille des paramètres retenus** en se référant aux records connus en terme de résolution de problèmes difficiles (factorisation, logarithme discret) ou de recherche exhaustive.
- **Justification du choix des constantes** en expliquant par exemple comment elles ont été dérivées de constantes mathématiques, afin de justifier qu'elles n'ont pas été choisies afin d'introduire volontairement des faiblesses, ou bien obtenues par un processus d'optimisation.
- **Arguments empiriques de sécurité**. Afin de justifier le bon fonctionnement d'un générateur pseudo-aléatoire, on pourra par exemple fournir les résultats de tests classiques permettant d'estimer la qualité de l'aléa obtenu.
- **Preuves mathématiques de sécurité**. Certains schémas, notamment en cryptographie à clé publique, peuvent être validés par des preuves formelles permettant de ramener la sécurité à la difficulté d'un problème reconnu difficile comme par exemple celui de la factorisation de grands entiers. On pourra donc fournir ou faire référence à des preuves de sécurité, éventuellement dans des modèles particuliers.

Fin du document