Niveau : 2 **Orange**

Quelques primitives cryptographiques

1 Introduction

Les **protocoles cryptographiques**, dont les fonctionnements exacts sont précisés dans des **standards** utilisent un certain nombre de primitives cryptographiques aptes à assurer les fonctions de base dont ils ont besoin.

Si on veut se faire une idée des primitives dont on a besoin en priorité, il suffit de regarder quelques standards connus, par exemple *openssh*, *gnupg*, *openssl*, *ISO-18033-2*.

Précisons aussi que la plupart des objets cryptographiques dans les protocoles courants sont dans un format de type *suite d'octets*. C'est le cas des textes clairs, des textes chiffrés, des clés publiques, privées, secrètes, des empreintes, des codes d'authentification, des apendices de signature. Quand on a besoin de ces données sous une autre forme, par exemple sous forme d'entiers pour mener à bien un certain nombre de calculs arithmétiques, on utilise alors des fonctions de traduction ainsi qu'il est expliqué dans la *fichecrypto_109*.

Pour toutes ces raisons, introduisons $B = \{0,1\}^8$ l'ensemble des octets et B^* l'ensemble des mots construits sur l'alphabet B, c'est-à-dire l'ensemble des suites finies d'octets.

2 Générateur pseudo-aléatoire

Un générateur pseudo-aléatoire est en général construit à partir d'une fonction mathématique déterministe, qui calcule par récurrence une suite d'états, dont est tirée la suite pseudo-aléatoire. La suite d'états est elle même une suite de valeurs internes cachées. La valeur initiale de l'état est appelée le germe. Si un ennemi ignore la valeur du germe, il doit lui être impossible en pratique, à partir des premiers termes d'une **suite pseudo-aléatoire cryptographique**, de prévoir le terme suivant. Un cas typique est celui où on dispose d'une fonction f et d'une fonction à sens unique F. À partir du germe s_0 on calcule une suite cachée d'états : $s_k = f(s_{k-1})$, et la suite pseudo-aléatoire $x_k = F(s_k)$.

Voici l'exemple du générateur de Blum-Blum-Shub :
On se donne un produit $n = pq$ de deux grands nombres premiers de Blum (de la forme $4k+3$) p et q gardés secrets. On calcule alors à partir d'un germe s_0 gardé lui aussi secret les états successifs s_k par la formule de récurrence :

$$s_{k+1} = s_k^2 \mod n.$$

De l'état s_k (qui est une valeur interne secrète) on tire le k^{ieme} bit u_k de la suite pseudo-aléatoire en prenant le bit de poids faible de s_k :

$$u_k = lsb(s_k).$$

Le germe lui-même est construit en utilisant un procédé physique qui renvoie un nombre imprévisible. Par exemple sous le système d'exploitation linux on dispose d'un device /dev/random qui accumule des aléas en fonction des clics de souris, des déplacements de souris, des touches frappées au clavier

etc. La lecture de ce device renvoie une suite d'octets (nombre occasionnel) pouvant servir de germe à un générateur pseudo-aléatoire.

On peut réaliser des générateurs pseudo-aléatoires avec des circuits de chiffrement ou des fonctions de hachage (cf. par exemple la construction de KDF). Cependant, on essaie de préférence de construire des procédés spécifiques qui peuvent travailler par flot et non par bloc.

3 Fonction de hachage cryptographique

Soit k un nombre fixé qui représentera un nombre d'octets. Une fonction de hachage est une application :

$$h : B^* \longrightarrow B^k$$

qui vérifie un certain nombre de propriétés :
- résistance à la préimage : soit $y \in B^k$. Il est impossible en pratique de trouver un x tel que $h(x) = y$.
- résistance aux collisions : il est impossible de calculer en pratique deux élément distincts u et v tels que $h(u) = h(v)$.

4 Générateur de masque

C'est en quelque sorte un générateur pseudo-aléatoire, en ce sens qu'à partir d'une suite d'octets (qu'on peut considérer comme un germe) on récupère une suite d'octets pseudo-aléatoire d'une longueur donnée. C'est aussi proche d'une fonction de hachage dans la mesure où à partir d'une suite d'octets en entrée, on récupère une suite d'octets de taille donnée, à part ici que la taille peut être variable. De manière plus formelle un générateur de masque est une application H de $B^* \times \mathbb{N}$ dans B^* qui à une suite finie d'octets x et à un entier n, fait correspondre une suite d'octets $H(x, n)$ de longueur n.

Un générateur de masque est aussi appelé un **KDF** (Key Derivation Function) car il permet de construire à partir d'une suite d'octets, une ou plusieurs clés de taille donnée.

Voici une façon classique de construire un générateur de masque H à partir d'une fonction de hachage h. Notons t la taille en octets de l'empreinte calculée par cette fonction de hachage :

$$t := \text{TailleOct}(h(u)).$$

La valeur de $H(x, n)$ se calcule de la façon suivante :
- on pose :
$$k = \left\lceil \frac{n}{t} \right\rceil$$
- pour $0 \leq j \leq k - 1$ on note y_j la suite de 4 octets dont la valeur correspondante en nombre entier est le nombre j. c'est-à-dire par exemple :

$$y_0 = 00000000000000000000000000000000,$$

$$y_1 = 00000000000000000000000000000001$$
- puis on calcule $z_j = h(x||y_j)$,
- et enfin $H(x, n)$ est obtenu en prenant les n premiers octets de :

$$z_0||z_1||\cdots||z_{k-1}.$$

5 Chiffrement

On n'insistera pas trop sur ces primitives très importantes qu'on a plus longuement étudiées par ailleurs. On rappelle que le chiffrement est divisé en deux grandes catégories toutes deux indispensables : le chiffrement à clé secrète et le chiffrement à clé publique.

5.1 Clé secrète

Le chiffrement à clé secrète est lui-même divisé en deux types : le chiffrement à flot (au fil de l'eau) et le chiffrement par blocs. Dans les deux cas, on dispose de deux fonctions publiques : la fonction de chiffrement $\mathcal{E}$ qui à un texte clair et une clé scrète K fait correspondre un texte chiffré y, et la fonction de déchiffrement $\mathcal{D}$ qui à un texte y et une clé K fait correspondre le texte clair x si y est le chiffré de x avec la clé K, et part en erreur si y n'est pas un chiffré valide avec la clé K. Ains l'expéditeur et le destinataire doivent posséder la même clé secrète K. Celle-ci est dans la plupart des protocoles construite aléatoirement à la volée (clé de session) et échangée avec une méthode d'échange de clé basée sur de la cryptographie à clé publique.

5.2 Clé publique

Pour le chiffrement à clé publique, chaque utilisateur X posséde une paire de clés : la clé publique e_X mise à la disposition de tous (sur un serveur par exemple) et la clé privée d_X connue seulement de son propriétaire X. On dispose alors de deux fonctions publiques, une fonction de chiffrement $\mathcal{E}$ qui permet à tout le monde de chiffrer un message x à destination de l'utilisateur X en calculant $y = \mathcal{E}(x, e_X)$ et une fonction de déchiffrement $\mathcal{D}$ qui permet à X de déchiffrer grâce à sa clé privée en calculant $x = \mathcal{D}(y, d_X)$. Ces primitives qui s'appliquent à des messages courts, à cause de leur temps d'exécution relativement long, servent essentiellement à échanger des clés.

6 Échange de clé

Une primitive de chiffrement à clé publique peut servir à l'échange d'une clé secrète. Cependant plusieurs protocoles préfèrent utiliser des méthodes spécifiques (échange de clé de Diffie-Hellman par exemple) qui sont souvent mieux à même d'assurer plus de sécurité rétroactive.

7 Signature

La signature garantie l'intégrité, l'authentification et la non-répudiation. Chaque utilisateur X dispose d'une clé publique e_X et d'une clé privée d_X. On dispose aussi d'une fonction de hachage publique h. Lorsque X veut signer un message x, il utilise sa clé privée et la fonction publique de signature $\mathcal{S}$ pour calculer l'appendice $s = \mathcal{S}(h(x), d_X)$. Il transmet alors le couple (x, s) constitué du message x et de l'appendice s. Tout le monde peut vérifier que la signature est valide en utilisant la fonction publique de vérification $\mathcal{V}$ et la clé publique de X, par le calcul de $\mathcal{V}((x, s), e_X)$. La signature peut être utilisée pour s'authentifier par signature de messages tests envoyés par le correspondant.

8 Code d'authentification de message

Les MAC (ou encore fonctions de hachage à clé secrète) permettent à un utilisateur de s'authentifier et en même temps d'assurer l'intégrité d'un message auprès d'un correspondant qui partage la même

clé secrète que lui. On peut construire un MAC avec un circuit de chiffrement à clé secrète. Mais on préfère souvent utiliser un circuit spécifique.