Niveau : 2 **Orange**

Chiffrement mixte

1 Présentation du chiffrement mixte

Les diverses primitives cryptographiques (chiffrement à clé publique, chiffrement à clé secrète, signature, hachage, compression, authentification de messages (MAC), générateur de masque (ou KDF : key derivation functions) etc.) contibuent à établir des protocoles d'échange sécurisé de données, d'accès à des services etc. plus complexes. Nous décrivons ici les principes des protocoles de chiffrement mixtes, qui utilisent le **chiffrement à clé secrète** pour chiffrer les flux de données et le **chiffrement à clé publique** pour échanger les clés. Il existe des normes qui décrivent exactement de tels protocoles. Nous renvoyons par exemple le lecteur à la norme **iso 18033-2**. Sans rentrer dans des détails techniques d'implémentation, nous suivons ici dans ses grands principes cette norme et utilisons ses définitions et notations.

Remarquons que ce type de chiffrement est indispensable car il n'est pas envisageable de chiffrer des flux de données avec du chiffrement à clé publique, surtout s'il s'agit de chiffrer en temps réel (penser à une communication téléphonique par exemple), à cause de la lenteur de ce type de chiffrement. Il est donc nécessaire dans la plupart des applications de chiffrer avec un système à clé secrète, en échangeant la clé secrète avec un système à clé publique.

2 Les étapes du travail à faire

2.1 Mise en place d'un système à clé publique

Il s'agit tout d'abord de mettre en place un système de chiffrement à clé publique (nous allons prendre RSA) et donc chaque utilisateur A doit se constituer une paire de clés. La clé publique (n_A, e_A) et la clé privée d_A. Ceci se fait avec une procédure *RSAKeyGen()* qui renvoie un triplet (n, e, d) où, comme d'habitude, n est le module (produit de deux grands nombres premiers) ayant par exemple 2048 bits, e l'exposant de chiffrement qui doit être premier avec $\Phi(n) = (p-1)(q-1)$, et d l'exposant de déchiffrement qui vérifie

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

2.2 Génération aléatoire et encapsulation d'une clé secrète

Il s'agit ici d'engendrer la clé de session, clé secrète pour le système à clé secrète choisi (nous supposerons ici qu'il s'agit par exemple d'AES avec une clé de 128 bits, utilisé en mode counter par exemple). Cette clé secrète doit être encapsulée dans une chaîne d'octets, puis chiffrée par le système à clé publique du correspondant. Nous donnons ici le protocole KEM (Key Encapsulation Mechanism) qui permet simultanément de créer une clé secrète et de l'encapsuler. Remarquons que cette méthode est spécifique d'une encapsulation de clé, et ne peut être utilisée comme OAEP (cf. fiche-crypto_ 202), pour un chiffrement asymétrique d'intérêt général.

RSA-KEM utilise un entier positif $KeyLen$ qui représente la longueur en octets de la clé secrète à encapsuler ainsi qu'un algorithme $RSAKeyGen$ de génération de couple de clés RSA et un générateur de masque (ou Key Derivation Function) KDF.

On doit donc avant toute chose avoir établi un système RSA de la même façon que dans la "fiche-crypto_ 202" :

$$(n, e, d) = RSAKeyGen().$$

RSA-KEM définit alors deux algorithmes : $RSA{-}KEM.Encrypt(n, e)$ et $RSA{-}KEM.Decrypt(n, d, C)$. Le premier encapsule une clé secrète en utilisant le chiffrement RSA de clé publique (n, e). Le deuxième retrouve la clé secrète encapsulée dans C en utilisant le déchiffrement RSA de clé privée (n, d).

2.2.1 Encapsuler la clé secrète

L'algorithme $RSA - KEM.Encrypt(n, e)$ fonctionne de la manière suivante :

1) On génère au hasard un nombre $r \in [0..n[$.
2) On chiffre ce nombre avec RSA : $v = RSA_{(n,e)}(r)$.
2) On transforme ce nombre en suite d'octets :

$$C = I2OSP(v, \mathcal{L}(n))$$

3) On calcule la clé secrète $K = KDF(I2OSP(r, \mathcal{L}(n)), KeyLen)$.
4) On renvoie C et K.

Remarque : bien entendu, seul C est destiné à être transmis au destinataire.

2.2.2 Décapsuler la clé secrète

1) À partir de C on retrouve r puis $I2OSP(r, \mathcal{L}(n))$ par déchiffrement RSA utilisant la clé privée (n, d).
2) On calcule $K = KDF(I2OSP(r, \mathcal{L}(n)), KeyLen)$.
3) On renvoie K.

2.3 Encapsulation des données

L'encapsulation des données passe par un mécanisme appelé DEM (Data Encapsulation Mechanism) qui définit tout d'abord une longueur de clé $DEM.KeyLen$. Le mécanisme DEM définit aussi l'algorithme de chiffrement

$$DEM.Encrypt(K, L, M)$$

ainsi que l'algorithme de déchiffrement

$$DEM.Decrypt(K, L, C_1).$$

Plusieurs implémentations voisines du mécanisme DEM sont admises dans la norme. Nous décrirons ici pour fixer les idées l'implémentation appelée DEM1.

2.3.1 Le chiffrement

L'algorithme de chiffrement $DEM.Encrypt(K, L, M)$ a pour entrées une clé secrète K, une étiquette L, un texte clair M, qui sont des suites d'octets. En sortie on récupère un texte chiffré $C1$ qui est lui aussi une suite d'octets. La clé K doit être de longueur $Dem.KeyLen$.

Dans DEM1, ce mécanisme est implanté comme suit. On dispose d'un système de chiffrement symétrique SC ainsi que d'un code d'authentification de message MA. La valeur $DEM1.KeyLen$ est définie par :

$$DEM1.KeyLen = SC.KeyLen + MA.KeyLen.$$

On effectue alors les opérations suivantes :

1) On analyse K comme $K = k||k'$ où k sera la clé secrète du système de chiffrement SC et aura pour longueur $|k| = SC.KeyLen$ et où k' sera la clé secrète de MA et aura pour longueur $|k'| = MA.KeyLen$.
2) On chiffre M : $c = SC.Encrypt(k, M)$.
3) On construit $T = c||L||I2OSP(8.|L|, 8)$.
4) On calcule $MAC = MA.eval(k', T)$.
5) Puis $C_1 = c||MAC$.
6) La sortie est C_1

2.3.2 Le déchiffrement

L'algorithme de déchiffrement $DEM.Decrypt(K, L, C1)$ a pour entrées une clé secrète K, une étiquette L, un texte chiffré $C1$, qui sont des suites d'octets.En sortie on récupère un texte clair M qui est lui aussi une suite d'octets.

Dans DEM1, ce mécanisme est implanté comme suit. On effectue alors les opérations :

1)On analyse K comme $K = k||k'$ où k sera la clé secrète du système de chiffrement SC et aura pour longueur $|k| = SC.KeyLen$ et où k' sera la clé secrète de MA et aura pour longueur $|k'| = MA.KeyLen$.
2) Si $C_1 < MA.MACLen$ alors erreur.
3) On analyse C_1 sous la forme $C_1 = c||MAC$ (la longueur de MAC est connue).
4) On construit $T = c||L||I2OSP(8.|L|, 8)$.
5) On calcule $MAC' = MA.eval(k', T)$.
6) Si $MAC' \neq MAC$ alors erreur.
7) On calcule $M = SC.Decrypt(k, c)$.
8) La sortie est M.

2.4 Synthèse permettant le chiffrement mixte

Nous disposons maintenant de toutes les étapes qui permettent d'implémenter un chiffrement mixte. On suppose que le destinataire a établi sa clé publique et sa clé privée.

2.4.1 La partie chiffrement

1) On calcule $(K, C_0) = KEM.Encrypt(n, e)$ où (n, e) est la clé publique du destinataire.
2) On calcule $C_1 = DEM.Encrypt(K, L, M)$ où M est le texte clair et où L est une étiquette (identifiant la transaction par exemple) connue des deux parties et éventuellement publique.
3) On construit $C = C_0||C_1$.
4) On envoie C au destinataire.

2.5 La partie déchiffrement

1) À partir de C, on analyse $C = C_0||C_1$ ce qui est facile puisque la longueur de C_0 est connue. Si C ne peut pas être analysé sous cette forme alors erreur.
2) On calcule $K = KEM.Decrypt((n, d), C_0)$ où (n, d) est la clé privée du destinataire.
3) On calcule $M = DEM.Decrypt(K, L, C_1)$.
4) On renvoie M.