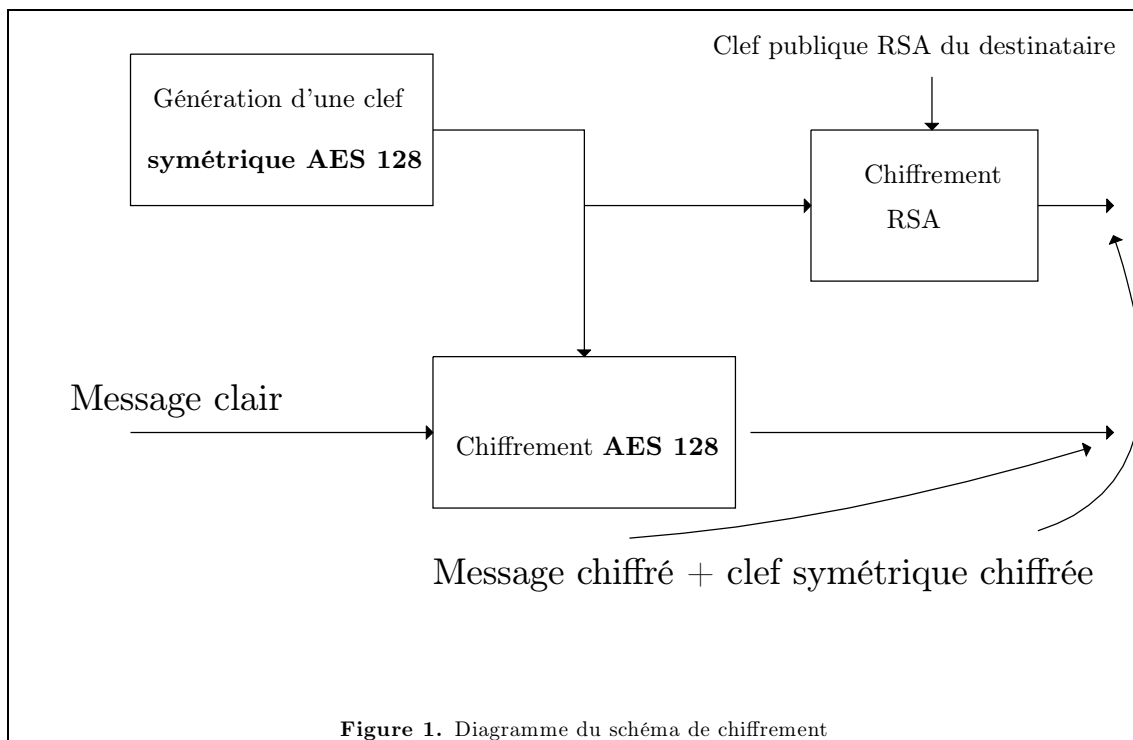
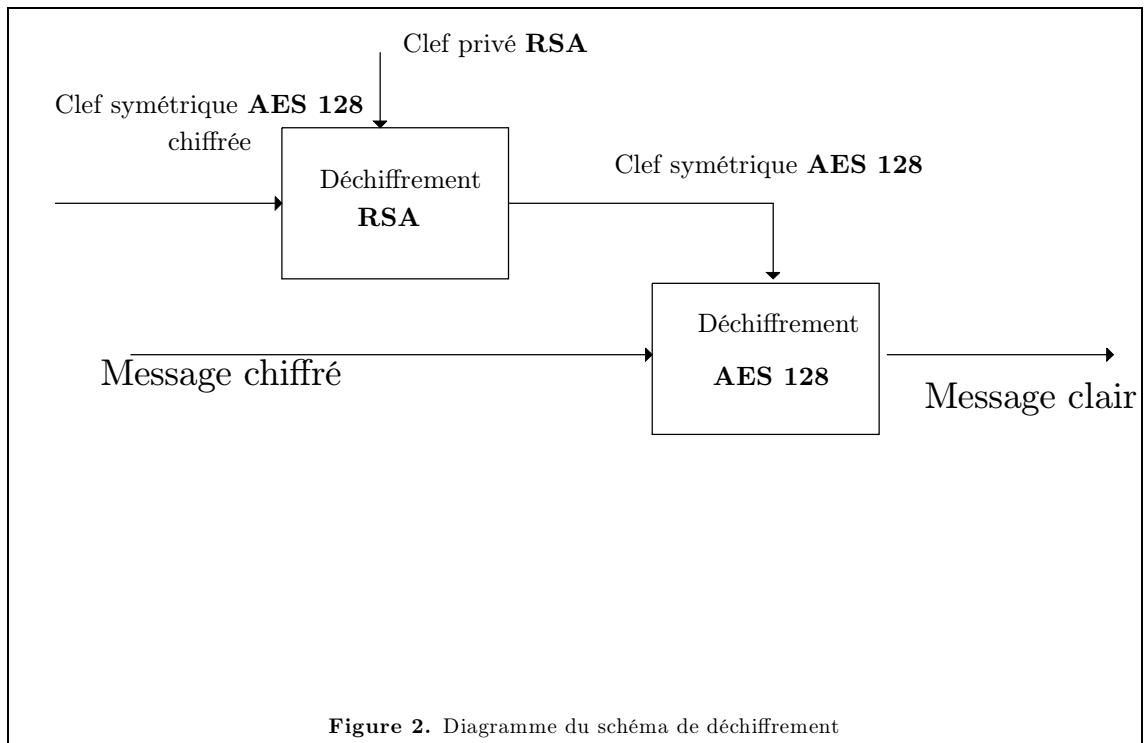


CONCEPTION D'UN OUTIL POUR L'ÉCHANGE DE FICHIER

Chiffrer-déchiffrer avec la bibliothèque crypto

L'objectif de cette partie est de concevoir un programme C basé sur la bibliothèque `crypto` de `openssl` pour faire des fonctions de chiffrement et de déchiffrement basées sur une approche PGP.





Vous disposez de fichiers C donnant des exemples d'utilisation pour AES 128 (fichiers `testsym.c`) et RSA (fichier `ExempleRSA.c`).

Vous devez implanter une petite bibliothèque de fonctions qui permet de faire facilement des échanges de fichiers en utilisant un chiffrement type PGP basé sur les deux schémas précédents. Il faut décomposer le problème de façon à écrire facilement deux fonctions :

- La fonction de chiffrement prend le fichier à chiffrer, la clef publique du destinataire, génère la clef symétrique et son chiffré puis écrit le fichier chiffré, la clef symétrique chiffré.
- La fonction de déchiffrement prend en entrée le fichier chiffré, la clef symétrique chiffré et

la clef privé du destinataire, elle déchiffre la clef symétrique et déchiffre le fichier.

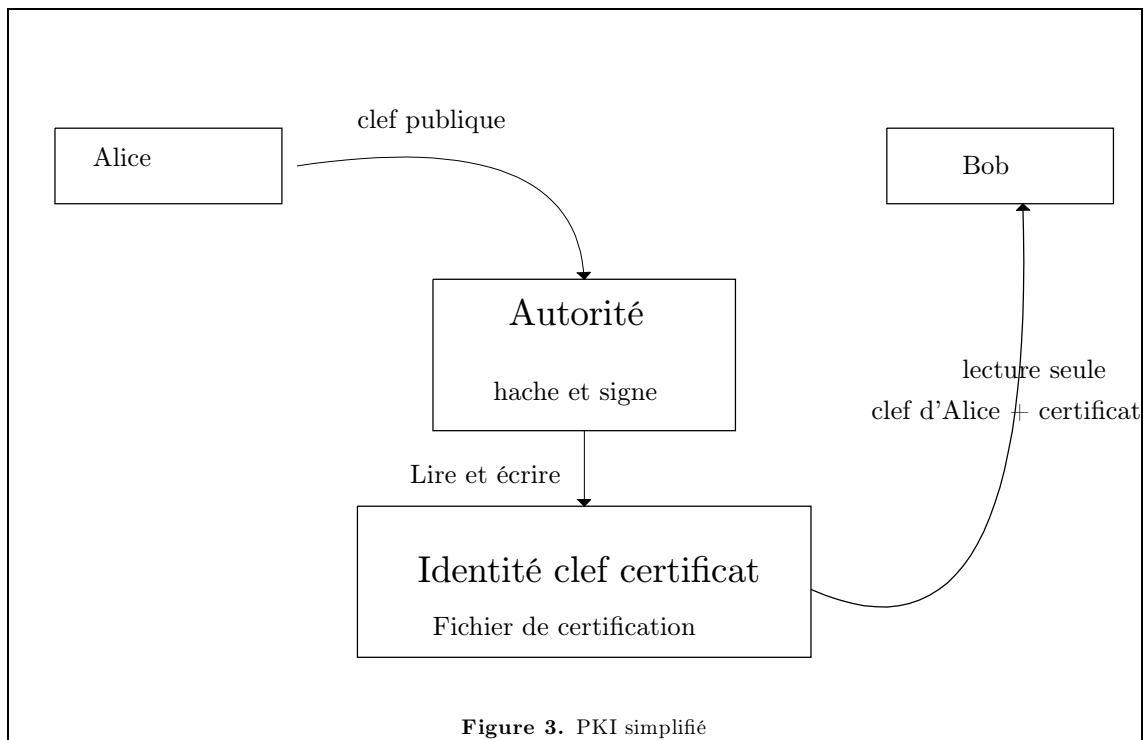
Le chiffrement asymétrique est trop lent pour être utilisé pour chiffrer des fichiers, c'est pourquoi on utilise cette approche. Vous disposez d'un `makefile` pour commencer à construire votre bibliothèque. La seule chose qui n'est pas dans le code fourni est la génération de clef symétrique pour que vous découvrirez la joie de lire la documentation de `openssl` ! Regardez la documentation de `rand`.

PKI simplifié

Maintenant que nous disposons de fonction pour chiffrer et déchiffrer en utilisant des clefs RSA, il faut gérer ces clefs. En effet, quand Bob charge la clef publique de Alice, qu'est-ce qui lui garantie que c'est bien la clef publique d'Alice ? Nous allons créer une autorité (vous) qui dispose de sa propre clef pair clef publique/privée. A chaque fois qu'un utilisateur veut utiliser vos fonctionnalité, il dépose sa clef publique auprès de l'autorité qui certifie alors la clef. Ceci permet d'avoir une garantie de l'appariement clef publique \leftrightarrow identité déclarée. Attention, ce n'est pas satisfaisant (voir plus bas). L'autorité maintient un fichier où chaque ligne est composée d'une identité, d'une clef publique et du certificat associé. Ainsi, si Bob veut donner un fichier à Alice, il récupère la clef publique d'Alice, il vérifie celle-ci à l'aide du certificat fourni par l'autorité, puis il lance son PGP. Pour cette partie vous devrez écrire les fonctionnalité permettant à l'autorité de générer le certificat (l'autorité signe un haché de la clef publique, mais ça ne garanti pas l'identité puisqu'il faudrait signer la pair haché de la clef/identité, mais pour l'instant simplifions). Et là, il faut lire la documentation `openssl` pour voir les fonctions de hachage et les signatures. Cherchez `SHA1` ...

L'autre fonction à implanter est celle qui permet à Bob de vérifier que la clef publique est bien celle d'Alice.

Ce mini-PKI est très insatisfaisant, d'où la partie pour les « pro ».



Partie pour les « pro »

En fait, généralement, ce n'est pas l'autorité ne fourni pas les clefs, mais seulement les certificats de la pair identité/clef publique. Comment feriez-vous et faites le !

Enfin, si on « garanti » la clef publique du destinataire, rien n'est fait pour l'expéditeur. La solution est qu'avec le chiffré de la clef symétrique, l'expéditeur signe un haché de celle-ci. A vous de jouer.