# Chapitre 15 - Chiffrement asymétrique- SSH

## 1- EXEMPLE DU CHIFFREMENT ASYMETRIQUE RSA :

Le cryptage RSA, du nom de ses concepteurs, Ron Rivest, Adi Shamir et Leonard Adleman, est le premier algorithme de chiffrement asymétrique. Il a été découvert en 1977 au Massachusetts Institute of Technology. Un chiffrement asymétrique est un cryptage où l'algorithme de chiffrement n'est pas le même que celui de déchiffrement, et où les clés utilisées sont différentes. L'intérêt est énorme : il n'y a plus besoin de transmettre la clé à son destinataire, il suffit de publier librement les clés de cryptage. N'importe qui peut alors crypter un message, mais seul son destinataire, qui possède la clé de décodage, pourra le lire. En quelques années, RSA s'est imposé pour le cryptage comme pour l'authentification et a progressivement supplanté son concurrent, le DES. Le RSA est basé sur la théorie des nombres premiers et a été vu en exercice. Sa robustesse tient du fait qu'il n'existe aucun algorithme de décomposition d'un nombre en facteurs premiers. Alors qu'il est facile de multiplier deux nombres premiers, il est très difficile de retrouver ces deux entiers si l'on en connaît le produit.

## 2- PROTOCOLE DE COMMUNICATION SECURISE SUR LE WEB :

Les protocoles de sécurisation du transfert des données sur les réseaux, portent le nom de SSH pour un transfert direct entre 2 machines distantes bien identifiées ou TLS pour une communication HTTPS sur le web (connexion à un site internet). Ces protocoles utilisent à un moment le chiffrement asymétrique avec une combinaison clé privée / publique. On détaille ci-dessous l'établissement d'une communication sécurisée avec TLS :

a. le navigateur du client envoie au serveur une demande de mise en place de connexion sécurisée par TLS.

b. Le serveur envoie au client son certificat : celui-ci contient sa clé publique, ses informations (nom de la société, adresse postale, pays, e-mail de contact...) ainsi qu'une signature numérique.

c. Le navigateur du client tente de vérifier la signature numérique du certificat du serveur en utilisant les clés publiques contenues dans les certificats des autorités de certifications intégrées par défaut dans le navigateur.

   Si l'une d'entre elles fonctionne, le navigateur web en déduit le nom de l'autorité de certification qui a signé le certificat envoyé par le serveur. Il vérifie que celui-ci n'est pas expiré puis envoie une demande OCSP à cette autorité pour vérifier que le certificat du serveur n'a pas été révoqué. Si aucune d'entre elles ne fonctionne, le navigateur web tente de vérifier la signature numérique du certificat du serveur à l'aide de la clé publique contenue dans celui-ci. En cas d'échec, le certificat est invalide, la connexion ne peut pas aboutir.

d. Le navigateur du client **génère une clé de chiffrement symétrique**, appelée **clé de session**, qu'il chiffre à l'aide de la clé publique contenue dans le certificat du serveur puis transmet cette clé de session au serveur.

e. Le serveur déchiffre la clé de session envoyée par le client grâce à sa clé privée.

f. Le client et le serveur commencent à s'échanger des données en chiffrant celles-ci avec la clé de session qu'ils ont en commun. **On considère à partir de ce moment que la connexion TLS est alors établie entre le client et le serveur.**

g. Une fois la connexion terminée (déconnexion volontaire de l'utilisateur ou si durée d'inactivité trop élevée), le serveur révoque la clé de session.

## 3- HACHAGE DES DONNEES :

Le **chiffrement** permet de chiffrer un message avec une clé. Il sera possible ensuite de la déchiffrer avec la même clé (chiffrement symétrique) ou avec une autre clé (chiffrement asymétrique).

Le **hachage** est un **processus à sens unique** qui transforme des données en une chaîne unique de longueur fixe appelée **valeur de hachage**. Le hachage permet de garantir que les données n'ont pas été altérées. Par exemple, lorsqu'un mot de passe est stocké dans une base de données, c'est son **hachage** qui est enregistrée. Même si des pirates parviennent à pénétrer dans la base de données, ils ne verront qu'un désordre brouillé, et non les vrais mots de passe.

Voici comment cela fonctionne :

- Une **fonction de hachage** prend une entrée (comme votre mot de passe) et produit une valeur de hachage unique.
- Si une seule lettre change dans l'entrée, le hachage sera complètement différent. C'est ce qu'on appelle la **résistance aux collisions :** deux entrées **ne** doivent **pas** produire le même hachage.

Les **algorithmes de hachage** les plus courants sont **SHA-256**, **MD5**, **RIPEMD** et **BLAKE3**.

Le hachage n'est pas seulement utilisé pour les mots de passe. Il permet de vérifier l'intégrité des fichiers (par exemple, de s'assurer que les téléchargements ne sont pas corrompus) et de protéger les données dans la **technologie blockchain.**

En bref : **le hachage ne verrouille pas vos données,** il en crée **une empreinte digitale** appelée aussi **empreinte numérique.** Il est parfait pour les situations où on a un besoin de vérification et non de secret.

On donne ci-dessous, une comparaison rapide entre chiffrement et hachage :

| Fonctionnalité | Cryptage | Hachage |
|---|---|---|
| **Processus** | Bidirectionnel (peut être inversé) | Unidirectionnel (irréversible) |
| **Objectif** | Protection de la confidentialité | Garantir l'intégrité |
| **Clé requise ?** | Oui, pour le cryptage et le décryptage | Non |
| **Sortie** | Longueur variable (en fonction de la taille des données) | Longueur fixe (par exemple, 256 bits pour SHA-256) |
| **Cas d'utilisation** | Sécurisation des données en transit (courriels, fichiers) | Vérification de l'intégrité des données (mots de passe) |