Sécurité

- Protection, domaine et matrice d'accès
- Sécurité contre les attaques
- Authentification
- Sécurité des applications, attaques et logiciels malveillants
- Protection contre les attaques
- Introduction et histoire de la cryptographie
- Cryptographie symétrique et asymétrique
- Signatures cryptographiques

Protection, domaine et matrice d'accès

Lorsque l'on parle de **protection**, on parle de l'ensemble des mécanismes mis en place pour l'accès, la gestion des ressources systèmes par les processus, etc. Cette protection doit être fournie autant par le système que par les applications.

La **sécurité** en revanche concerne un spectre plus large incluant les virus, les attaques et la cryptographie.

La protection a pour but de prévenir les violations d'accès et d'améliorer la fiabilité (en détectant des erreurs humaines par exemple).

Si une ressource n'est pas protégée, elle peut être mal utilisée par des utilisateur·ice·s autorisé·e·s (ou non).

Une **politique** de protection correspond à la définition de ce qui doit être protégé. Tandis qu'un **mécanisme** de protection indique comment protéger.

Zones de protections

Sur l'ordinateur, il faut pouvoir protéger les processus et les **objets**.

Les objets peuvent être autant matériel (CPU, mémoire, imprimante, disque, etc) que logiciel (fichiers, programmes, sémaphore, etc).

Pour ce qui est des processus, il faut s'assurer que chaque processus ne peut accéder qu'aux ressources auxquelles il a l'autorisation. Il ne doit accéder qu'aux ressources qui sont nécessaires pour terminer sa tâche.

Domaine de protection

Un **domaine** définit un ensemble d'objet et de type d'opération qui peut être exécutée sur les objets (la possibilité d'exécuter une action sur un objet s'appelle un **droit d'accès**).

Un domaine est donc finalement une collection de droits d'accès.

Chaque processus opère à l'intérieur d'un domaine de protection. Les mêmes objets peuvent être référencés dans plusieurs domaines.

Un domaine peut être créé de plusieurs manières (par utilisateur, processus, procédure, etc). Et des opérations pour changer de domaine sont prévues.

L'association entre un processus et un domaine peut être **statique** (ne change pas, les droits d'accès restent donc tout le temps les mêmes), ou **dynamique** (les droits d'accès peuvent changer).

Matrice d'accès

La matrice d'accès est une représentation des domaines et de leurs droits d'accès.

C'est un tableau à deux dimensions où chaque ligne représente un domaine (aussi appelé rôle ou sujet), et chaque colonne représente un objet (aussi appelé asset).

Voici un exemple de matrice d'accès :

	Objets				
Sujets	Fichier 1	Segment 1	Segment 2	Processus 2	Editeur
Processus 1	Lire	Executer	Lire Ecrire		Entrer
Processus 2	Lire Ecrire				Entrer
Processus 3		Lire Ecrire Executer		Entrer	Entrer

La matrice d'accès permet de vérifier les politiques de protection voulues. Il faut donc définir les politiques pour chaque objet, assigner chaque domaine à l'exécution d'un processus.

Voyons maintenant quelques droits spécifiques,

Droit au changement de domaine

object / domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch control
D_3		read	execute					
D_4	write		write		switch			

Pour représenter la permission de changer vers un autre domaine, il suffit de considérer les domaines aussi comme des objets. Ensuite pour permettre à D1 de pouvoir avoir les droits accès de D2, il suffit de mettre **switch** dans l'intersection de D1 et D2.

Droit à la copie de son droit

Le droit **copy** permet à un domaine de copier son droit pour un objet donné à un autre domaine.

Ce droit est représenté par une `*`.

object / domain	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute		

Par exemple, ici, D2 peut copier le droit lecture de F2 sur un autre domaine que le sien.

Il existe aussi une variante du droit copy qui est le droit **transfert** (ou copie limitée). Si dans l'exemple précédent, il s'agit d'un droit de transfert, alors lorsque D2 passe son droit de lecture de F2 à un autre domaine, il perd le droit de lecture.

Droit à la modification des droits d'un objet

Le droit **owner** permet à un domaine de modifier n'importe quel droit pour un certain objet.

domain \ object	F_1	F_2	F_3
D_1	owner execute		write
D_2		owner read* write*	read* owner write
D_3		write	write

Dans cet exemple, D1 possède un accès total à F1. De cette manière, D1 peut par exemple s'accorder un droit d'écriture sur F1 ou encore accordé un droit de lecture de F1 à D2.

Droit au contrôle des droits d'un domaine

Le droit **control** permet à un domaine de supprimer des droits d'accès à un autre domaine.

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch control
D_3		read	execute					
D_4	write		write		switch			

Dans cet exemple, D2 peut par exemple supprimer le droit d'accès en écriture (F1) de D4.

Sous UNIX (setuid)

Sous UNIX, chaque domaine correspond à un utilisateur·ice, et lorsque qu'un exécutable est lancé, le processus prend le domaine de l'utilisateur·ice qui l'a lancé.

Sauf lorsque le bit `setuid` est mis. Ce bit est représenté par un `s` lorsque l'on regarde les permissions d'un fichier. Lorsque c'est le cas, l'exécutable va se lancer en tant que l'utilisateur·ice qui possède le fichier, à la place de s'exécuter en tant qu'utilisateur·ice qui l'a lancé.

Par exemple, avec le fichier sudo, lorsque l'on fait un `ls -l` dessus, on peut voir les permissions à gauche.

```
-r-s--x--x 1 root root 63432 Jan  6 09:22 /run/wrappers/bin/sudo
```

On peut voir qu'il y a un `s`, cela signifie que lorsque j'exécute ce fichier en tant qu'utilisateur `snowcode`, le fichier est réellement exécuté en tant que `root`.

Implémentation

Utiliser une matrice d'accès dans le système ne serait pas très pratique, car prendrait beaucoup de place et ne pourrait pas être mise en mémoire centrale.

C'est pourquoi, on utilise des **access list** à la place. L'access list est une liste associée à chaque objet. Elle contient des pairs de domaines et de droits. Un objet dont le domaine n'a pas de droit n'est pas présent sur la liste.

Il est également possible d'étendre la liste avec des droits par défaut (dans quel cas, pour vérifier les droits d'une opération, on vérifie d'abord les droits par défaut avant de rechercher la paire correspondante au domaine).

Révocation des droits

Ensuite, chaque système doit aussi prévoir comment révoquer des droits. Par exemple pour définir si c'est immédiat ou décalé, pour tous les utilisateurs ou seulement certains, si cela est temporaire ou permanent, etc.

Chaque système définit les stratégies possibles et laisse le choix à l'utilisateur·ice.

Sécurité contre les attaques

Nous avons vu comment protéger le système et déterminer les droits d'accès. Maintenant, il faut trouver des mécanismes pour protéger le système contre le vol d'information, la modification/destruction non autorisée d'information et surveiller l'utilisation du système.

Les mécanismes de sécurité ont pour but d'empêcher ou de ralentir le plus possible toute violation du système, il faut que le cout pour pénétrer un système soit plus important que le gain que l'on peut en retirer.

Niveaux de protections

Il y a 4 niveaux de protections différents :

1. Physique, protéger le matériel dans des endroits protégés
2. Humains, surveillance pour l'accès au matériel
3. Réseau, protection adéquate, firewalls, etc
4. Système d'exploitation

Dangers d'une attaque

La sécurité est évidemment primordiale, les entreprises stockent toutes leurs informations sur des systèmes informatiques et les informations personnelles valent de l'or. Des informations perdues ou volées peuvent conduire une entreprise à la faillite.

Authentification

Les systèmes actuels ne fonctionnent que lorsque l'utilisateur·ice est authentifié·e.

Nous allons parler ici de plusieurs types d'authentification différents et de quelques bonnes pratiques pour chacun d'entre eux.

Identification par mot de passe

Le mot de passe est une méthode courante d'authentification, cependant elle n'est pas forcément très sûre, notamment, car elle dépend beaucoup de l'utilisateur·ice.

Comment un mot de passe peut être découvert

Un mot de passe peut être volé de plusieurs manières, soit en **connaissant l'utilisateur** et en devinant le mot de passe (c'est pourquoi il faut que les mots de passes soient aléatoires), ou de façon **brutale** (par dictionnaire ou énumération).

Il est aussi possible de voler un mot de passe en utilisant un **keylogger**, c'est-à-dire un programme ou appareil qui va enregistrer toutes les entrées clavier de l'utilisateur·ice.

Un mot de passe peut aussi être **sniffé** en écoutant tout ce qui se transmet sur le réseau, si le mot de passe n'est pas transféré de manière chiffrée, alors on peut récupérer le mot de passe.

Un mot de passe peut encore être découvert en analysant des bases de données d'anciens mots de passe découvert, car mal stocké par d'autres entreprises. En effet, comme beaucoup d'utilisateur·ice·s ne changent pas de mot de passe, il y a de grandes chances que ce dernier n'ait pas changé.

Enfin, le mot de passe peut encore être découvert à cause d'autres maladresses de l'utilisateur·ice·s, tel que tomber dans une attaque de **fishing** ou encore l'écrire sur un post-it sur son bureau.

Stockage d'un mot de passe

> Si vous voulez en savoir plus sur les dangers et les différentes manières de stocker des mots de passes, regardez cette vidéo.

Plain text

La pire manière de stocker des mots de passes est de juste les stocker en base de donnée. Car lorsque l'on fait cela, ça signifie que toute personne ayant accès à la base de donnée a accès à tous les utilisateurs et mots de passes.

Pire encore, puis ce que les utilisateur·ice·s réutilisent souvent les mêmes mots de passes, pour beaucoup d'entre eux, cela donne accès à l'entièreté de leur vie en ligne.

Chiffré

Une autre manière est de chiffrer les mots de passes. Cela signifie que si quelqu'un a accès à la base de donnée, il ne pourra rien en faire. Cependant, cela signifie aussi que si quelqu'un a accès à la clé de déchiffrement, cette personne a toujours accès à tous les mots de passes. C'est donc également une chose à éviter.

Un autre problème avec cette méthode est que si certains mots de passes sont identiques, on pourra le reconnaitre, car on verra le même mot de passe chiffré dans la base de donnée plusieurs fois.

Hashing

Cette manière consiste à hasher (faire passer à travers une fonction de hashage) les mots de passes.

Une fonction de hashage est une fonction à sens unique, ainsi, on fait passer le mot de passe à l'intérieur, cela génère un texte, mais il est impossible de retrouver le mot de passe à partir du texte.

Ainsi, il suffit de stocker le hash dans la base de donnée, ainsi lorsque l'utilisateur·ice veut se reconnecter, on hash le mot de psase entré et on compare le hash dans la base de donnée avec celui qui vient d'être généré.

Le problème avec cette méthode est que plusieurs utilisateur·ice·s ayant le même mot de passe vont avoir le même hash. Par conséquent, on le saura directement dans la base de donnée.

Il y a notamment une attaque en particulier qui utilise cette faiblesse, c'est la **rainbow table attack**, à la place d'avoir une liste de mots de passes courants, les hasher et les essayer un par un contre chaque hash (ce qui est une attaque par **dictionaire**), on va avoir une liste de mots de passes pré-hashé et simplement comparer les hash. Ce qui fait que l'attaque est beaucoup plus rapide, surtout que beaucoup d'algorithmes de hashage sont volontairement lents afin de décourager les attaquants.

Hashing avec salt

Cette méthode est l'une des meilleures méthodes aujourd'hui. Elle consiste à utiliser un **salt** avec le hash pour se protéger contre les **rainbow table attacks**.

Un **salt** est une chaine de caractère aléatoire différente pour chaque utilisateur qui va être ajouté à chaque mot de passe. Le salt est ensuite présent juste à côté du hash en clair.

Donc, lorsqu'un·e utilisateur·ice se connecte, on va aller chercher le salt, l'ajouter à son mot de passe et le hasher. Ensuite, on compare ce hash avec celui dans la base de donnée.

Grâce à cette chaine aléatoire (le salt), les rainbow table attacks sont inutiles, car elles ne peuvent plus comparer les hash.

Cette méthode est d'autant plus puissante lorsqu'elle fonctionne avec des algorithmes de hashage lent, parce que l'attaquant n'a pas d'autre choix que de tester les mots de passe un par un et va perdre beaucoup de temps.

Ne pas stocker de mot de passe

Stocker des mots de passes est une grande responsabilité, il est possible que ce soit plus sûr de ne tout simplement pas les stocker et d'utiliser des mécanismes tel que OAuth pour à la place demander à des tiers de confiance de le gérer pour nous. Par exemple par Google ou Microsoft.

Ou encore d'utiliser des appareils physiques tels qu'une carte électronique ou un appareil FIDO2 (on va en reparler plus tard) à la place ou en plus du mot de passe.

Identification à plusieurs facteurs

Une première manière de faire de l'authentification à plusieurs facteurs est d'utiliser un système de code à usage unique tel que le TOTP. À savoir que certains de ces mécanismes peuvent aussi être utilisés seul, pas seulement en conjonction avec un mot de passe (par exemple une clé FIDO2 pour authentification SSH), on va en reparler dans l'identification password-less.

L'avantage d'utiliser l'identification à plusieurs facteurs est de palier la faiblesse du système de mot de passe en demandant à un autre système de générer un code ou de résoudre un problème (par exemple, envois de SMS, TOTP, hash chain, clé FIDO2, etc).

Hash Chain

Une autre manière de faire est d'utiliser à répétition la fonction de hashage.

Par exemple, on génère une valeur aléatoire de départ (le seed).

Ensuite, on exécute la fonction de hashage un certain nombre de fois (par exemple 1000 fois) sur ce seed, ce qui donne donc un hash de hash de hash de hash de … du seed. Le serveur fait de même et sauvegarde ce hash.

Lors de la première authentification, l'utilisateur·ice génère un code à usage unique en dérivant une fois de mois qu'avant le seed. Dans cet exemple, l'utilisateur va donc hasher le seed 999 fois. Le serveur peut alors vérifier que lorsqu'il hash le seed, il obtient bien le hash de départ, l'authentification est donc réussie. Le serveur défini alors ce nouveau hash (de 999 fois) à la place de l'ancien (de 1000 fois).

Ainsi le processus se répète jusqu'à arriver à zéro où dans quel cas il faut générer un nouveau seed.

TOTP (Time-based One Time Password)

> Cette section est en bonus et n'est pas présente dans le cours

En somme, l'utilisateur·ice a un appareil (smartphone, digipass ou autre), qui partage un secret avec le serveur (token) ainsi que le temps (le temps doit être le même que sur le serveur).

Ainsi, après que l'utilisateur·ice a entré son mot de passe, on lui demande le code à usage unique. L'utilisateur·ice peut ensuite demander à l'appareil de générer ce code.

Ce code est généré en hashant le secret partagé (token) et le temps actuel. Ainsi, le mot de passe n'est valable que pour une certaine durée de temps.

En somme, en plus d'ajouter son mot de passe, l'utilisateur·ice va également

FIDO2

> Cette section est en bonus et n'est pas présente dans le cours. Vous pouvez en apprendre plus sur FIDO2 en regardant cette vidéo.

Lorsque l'on veut s'enregistrer sur un site internet, la clé FIDO2 va créer une paire de clé (clé privée et clé publique, on va y revenir plus tard).

Une fois la paire de clé générée, la clé publique est envoyée au site avec un "handle", cet handle contient des informations uniques sur le site qui a demandé l'authentification.

Ainsi, lorsque l'on souhaite se connecter, le site va envoyer un message aléatoire à signer à la clé, la clé va ensuite vérifier que le handle correspond bien (que c'est le bon site, donc protection contre le fishing, le bon compte, sur la bonne clé).

Si tout correspond, la clé va signer le challenge envoyé par le site. Le site pourra ensuite vérifier que la signature est correcte avec la clé publique.

Ce mécanisme est donc très simple (uniquement un bouton) et très sécurisé (protection contre le fishing, aucune information personnelle à stocker).

Identification password-less

L'identification password-less repose sur l'idée que les mots de passes sont compliqués à gérer, assez faible et que l'authentification à plusieurs facteurs peut être compliquée.

L'identification password-less repose donc sur d'autres principes tels que la biométrie (empreinte digitale, reconnaissance faciale, etc), sur un code unique (par SMS ou application tierce, voir TOTP plus tôt), par lien magique (envois d'un lien de connexion par mail), par notification push, ou par clé FIDO2 (voir plus tôt) par exemple.

> Si vous voulez en savoir plus sur l'authentification password-less, vous pouvez regarder cette vidéo.

Identification biométrique

L'identification biométrique a pour but d'authentifier sur base d'une propriété de l'utilisateur·ice (sa tête, ses doigts, ou autre).

Le scanner converti et numérise la propriété, la donnée numérisée est alors traitée comme un mot de passe et est hashée et salée.

Le problème de cette méthode est que la protection de ces données devient vraiment très critique, contrairement aux mots de passes, on ne peut pas changer sa tête ou ses empreintes digitales.

De plus, ces informations sont aussi des informations très confidentielles qui peuvent aussi avoir un impact politique important.

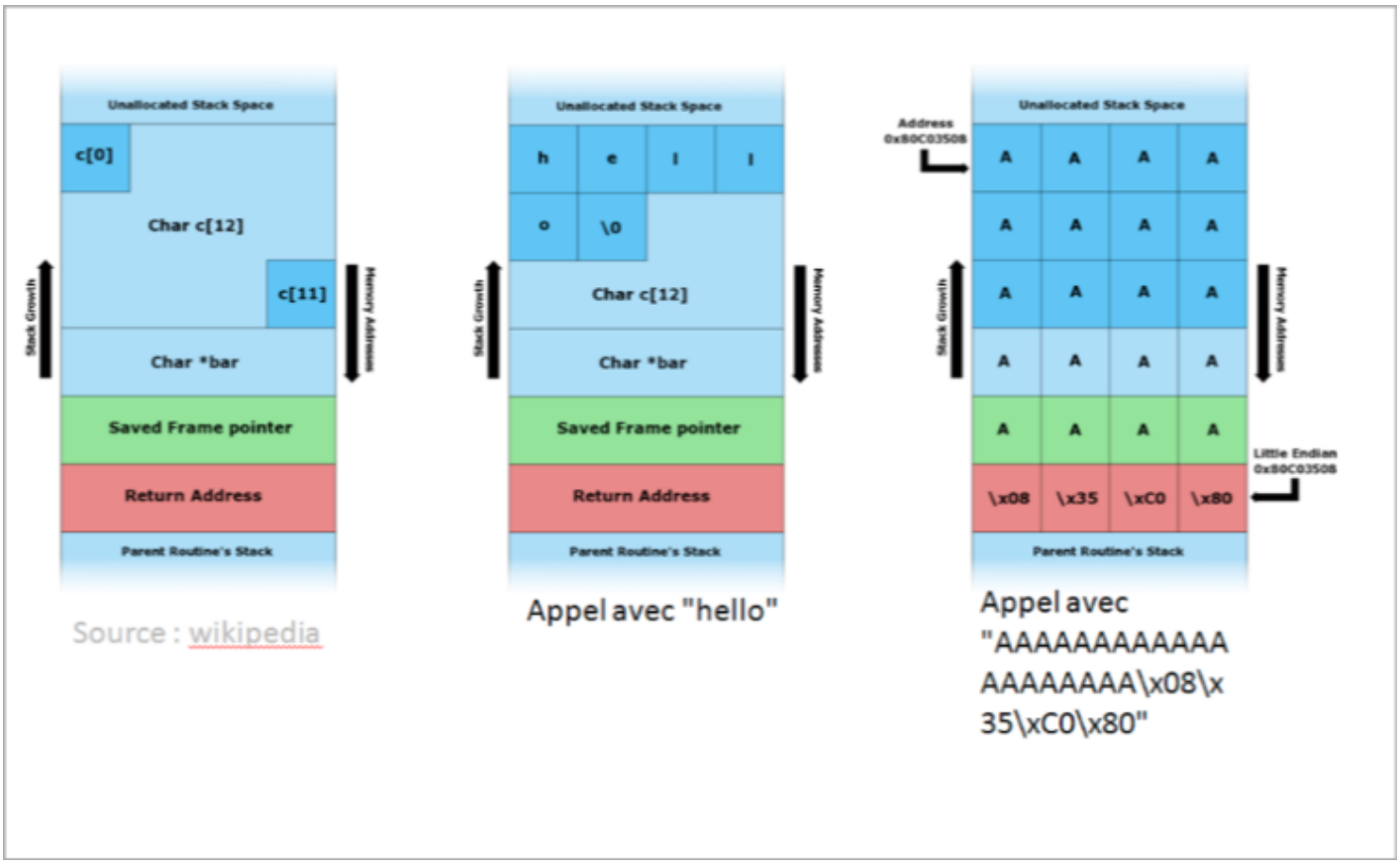
Sécurité des applications, attaques et logiciels malveillants

Écrire du code exempt d'erreur est difficile, et les erreurs peuvent conduire à des vulnérabilités qui permettent d'attaquer le programme.

L'attaque peut permettre d'obtenir des droits non accordés au départ, faire planter l'application, introduire des données incorrectes, etc.

Attaques courantes

- **Remote Code Execution** (RCE), exécution de code à distance en soumettant une donnée précise à l'application
  - Un exemple qui a fait beaucoup de bruit est celui de la vulnérabilité log4shell dans le système de log Java "log4j" qui faisait qu'il était possible d'exécuter du code sur toute application utilisant la librairie. Cette vulnérabilité était si dangereuse qu'elle fut considérée par certains gouvernements comme l'un des problèmes de sécurité informatique la plus sérieuse des 20 dernières années.
- **SQL Injection**, injection de code SQL dans la base de donnée en soumettant des données précises.
- **Format String vulnerabilities** qui consiste à soumettre du code qui est compris comme une commande par l'application. Pour en savoir plus, des exemples de code en C sont donnés dans cet article.
- **Cross-Site Scripting** (XSS), qui est encore dû à une non-vérification des soumissions de l'utilisateur·ice qui peut mener à intégrer du code HTML dans une page. Ce qui signifie que l'on peut aussi injecter du code JavaScript dans la page qui seront exécutés par tous les visiteurs de celle-ci.
- **Username enumeration**, si le système mentionne que le nom d'utilisateur est introuvable, il est possible de tester plusieurs noms d'utilisateurs pour savoir lesquels sont présents sur le système. Il est parfois aussi possible de faire cela via l'API du site (genre en regardant `/api/user/01` et les énumérer comme ça).
- **Stack/buffer overflow**, l'attaquant soumet une chaine de caractère spécifique qui provoque un débordement de la pile. Grâce à ce débordement, le code exécuté devient celui de l'attaquant, cela peut donc lui faire prendre le contrôle du code de l'application, et si l'application s'exécute dans le domaine administrateur, lui faire devenir administrateur (**privilege escalation**).



- **Déni de service** (DoS), le fait de rendre inaccessible un système en le bombardant de requêtes. Le système devient ainsi inaccessible durant le temps de l'attaque, bien qu'aucun dommage ne soit opéré, cela cause un problème en terme l'image pour l'entreprise et peut lui faire perdre de l'argent à cause de l'inaccessibilité du site.
  - Si une attaque DoS est fait depuis plein d'ordinateurs différents (ce qui fait qu'elle est plus compliquée à arrêter), on parle d'attaque de **déni de service distribuée** (DDoS). Chaque ordinateur est appelé un **zombie** et l'ensemble des ordinateurs infectés utilisé est appelé un **botnet**.

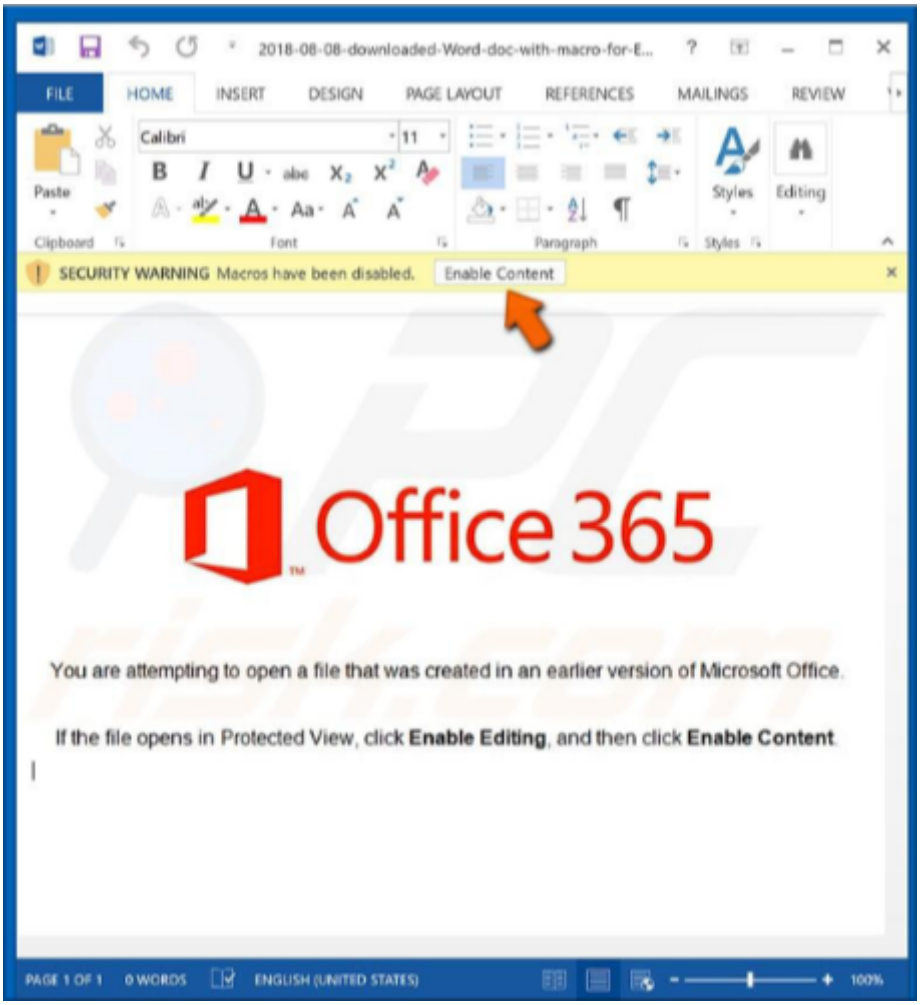
Programmes malveillants

Maintenant, on va parler de types de programmes malicieux,

- **Cheval de troie** (ou **trojan**), est un programme qui se fait passer pour ce qu'il n'est pas pour déclencher une action hostile. Cela peut par exemple être un faux anti-virus.
- **Backdoor** (porte dérobée), le programme installe une porte d'accès à l'utilisateur, il est ainsi possible de faire exécuter des commandes à la machine à distance.
  - C'est une méthode assez utilisée pour constituer des attaques DDoS, car la machine devient un zombie qui fait partie d'un botnet auquel l'attaquant demande de faire des requêtes à répétition sur un site.
  - Cette méthode peut aussi être utilisée pour installer un cryptomineur qui va miner des cryptomonnaies pour l'attaquant sur toutes les machines victimes
- Les **vers informatiques** (ou **WORMS**) sont des programmes qui se propagent par les réseaux informatiques. Les vers utilisent des vulnérabilités d'autres systèmes pour se propager, ainsi chaque machine infectée infecte les autres machines du réseau à son

tour.

  - Un exemple très connu de WORM est celui de WannaCry, un ransomware qui a infecté plus de 300000 ordinateurs sur plus de 150 pays différents et qui aurait causé des pertes de l'ordre de plusieurs centaines de millions de dollars. Ce WORM se propageait via une vulnérabilité dans un port de communication du système Windows.
- Les **virus informatiques** sont des ensembles d'instructions qui utilisent un programme pour se reproduire. Ainsi, comme un virus biologique, un virus informatique a pour principal but de se reproduire en infectant un programme hôte. Certains virus ont également une charge active qui attaque le programme à un moment déterminé. Certains virus utilisent des techniques avancées pour se cacher des systèmes de protection.
  - Les **virus script** sont des virus écrit dans un langage interprété (par exemple le VBA, langage de programmation de Microsoft Office), le virus utilise un environnement tiers pour s'exécuter et se reproduire. Cela peut par exemple être sous la forme d'une macro d'un document Microsoft Office.



Protection contre les attaques

Pour se protéger contre des attaques, il est important de protéger le **périmètre** (tout ce qui est vers l'extérieur, tel que le réseau), en installant un firewall au niveau du réseau.

Ensuite, il est aussi important de protéger les **machines** en installant un firewall personnel et un anti-virus.

D'autres choses sont importantes telles que ne jamais travailler avec un compte administrateur.

Sécurité d'un parc informatique

Pour assurer la sécurité d'un parc informatique, on peut utiliser des **scanners de vulnérabilités**, ce sont des programmes qui vont regarder si le système est vulnérable à certaines attaques afin de pouvoir mettre à jour les composants qui en ont besoin.

Il faut aussi pouvoir assurer l'**intégrité des programmes**, on peut donc garder de manière sécurisée les empreintes de tous les programmes et vérifier celles-ci lors de leur lancement. Le seul souci, c'est qu'il faut tenir compte des mises à jour.

On peut aussi utiliser un **système de détection d'intrusion** (IDS), c'est un programme qui tente de repérer des activités anormales sur un système en se basant sur des plans d'attaque connus, en observant l'activité et les journaux systèmes.

Par exemple, `fail2ban` est un IDS qui bannit les adresses IP qui réalisent un certain nombre de tentatives de connexions illicites.

Un IDS peut aussi analyser le comportement de l'utilisateur pour détecter des comportements anormaux (exemple un secrétaire qui compile un logiciel). Il faut cependant faire attention à configurer l'IDS correctement pour éviter les faux positifs. Un autre exemple d'IDS est `Snort`

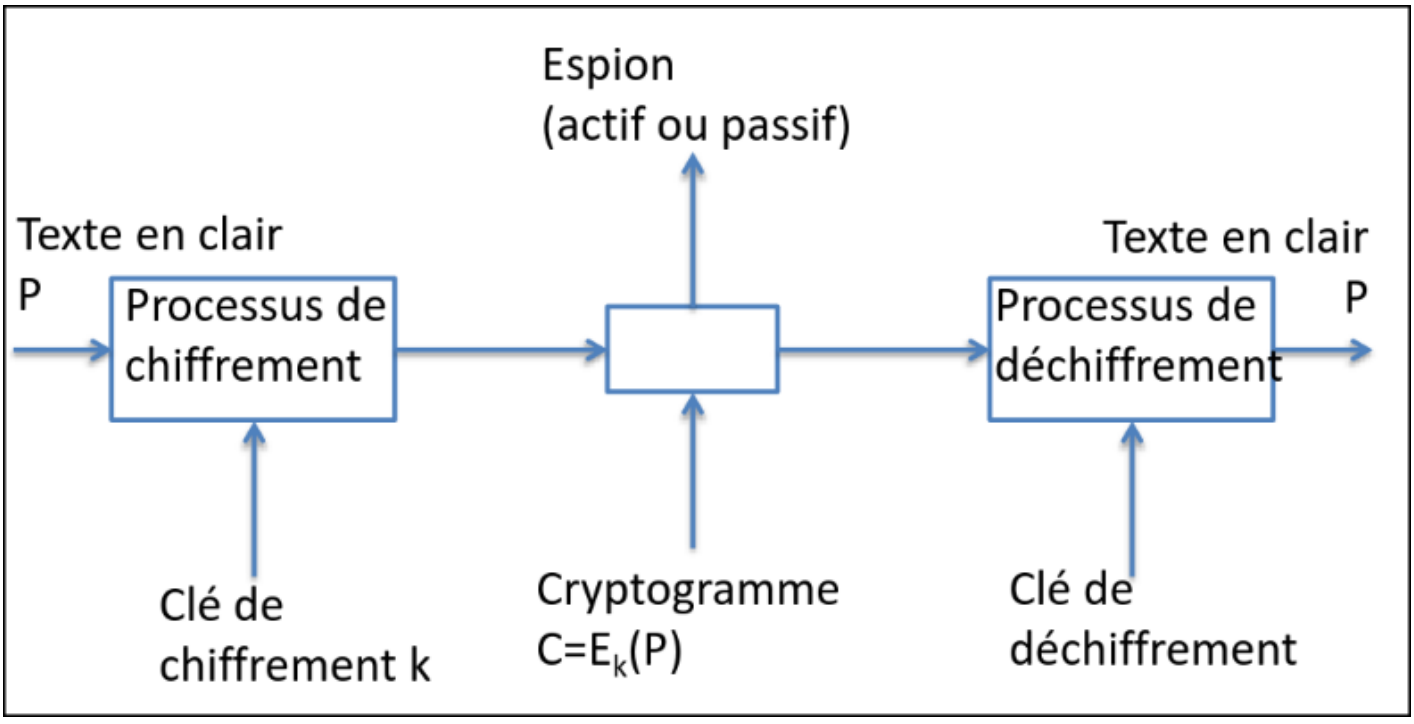
Enfin, un denier élément important est l'analyse de fichiers journaux. Tous les systèmes d'exploitation consignent des informations sur ce qu'il se passe sur le système, on peut donc utiliser des outils pour analyser automatiquement ces fichiers journaux tel que Splunk, Grafana Loki ou encore Crowdsec.

Introduction et histoire de la cryptographie

La **cryptographie** consiste à cacher des informations en utilisant des algorithmes. Il ne faut pas le confondre avec la **stéganographie** qui consiste à cacher des messages dans d'autres messages.

Par exemple, si on prend un message tel que `BONJOUR` (**texte en clair**), et que pour chaque lettre, on la décale d'un certain nombre de positions dans l'alphabet (**processus de chiffrement**), ici, on va choisir 13 positions, (13 sera ici la **clé de chiffrement**), on obtient `OBAWBHE` qui est notre **cryptogramme** (message chiffré).

Pour déchiffrer le message, il suffit alors de faire le processus inverse, ce processus inverse est donc le **processus de déchiffrement** qui retournera `BONJOUR` (notre texte en clair).



Histoire

La cryptographie est utilisée depuis très longtemps par les militaires, diplomates et amants.

Avant l'avènement de l'informatique, la cryptographie était limitée aux capacités du cerveau humain, il fallait pouvoir changer de méthode de chiffrement si quelqu'un se faisait prendre.

Avec l'avènement de l'informatique est venu la capacité de calculer des résultats beaucoup plus complexe.

Substitution et transposition

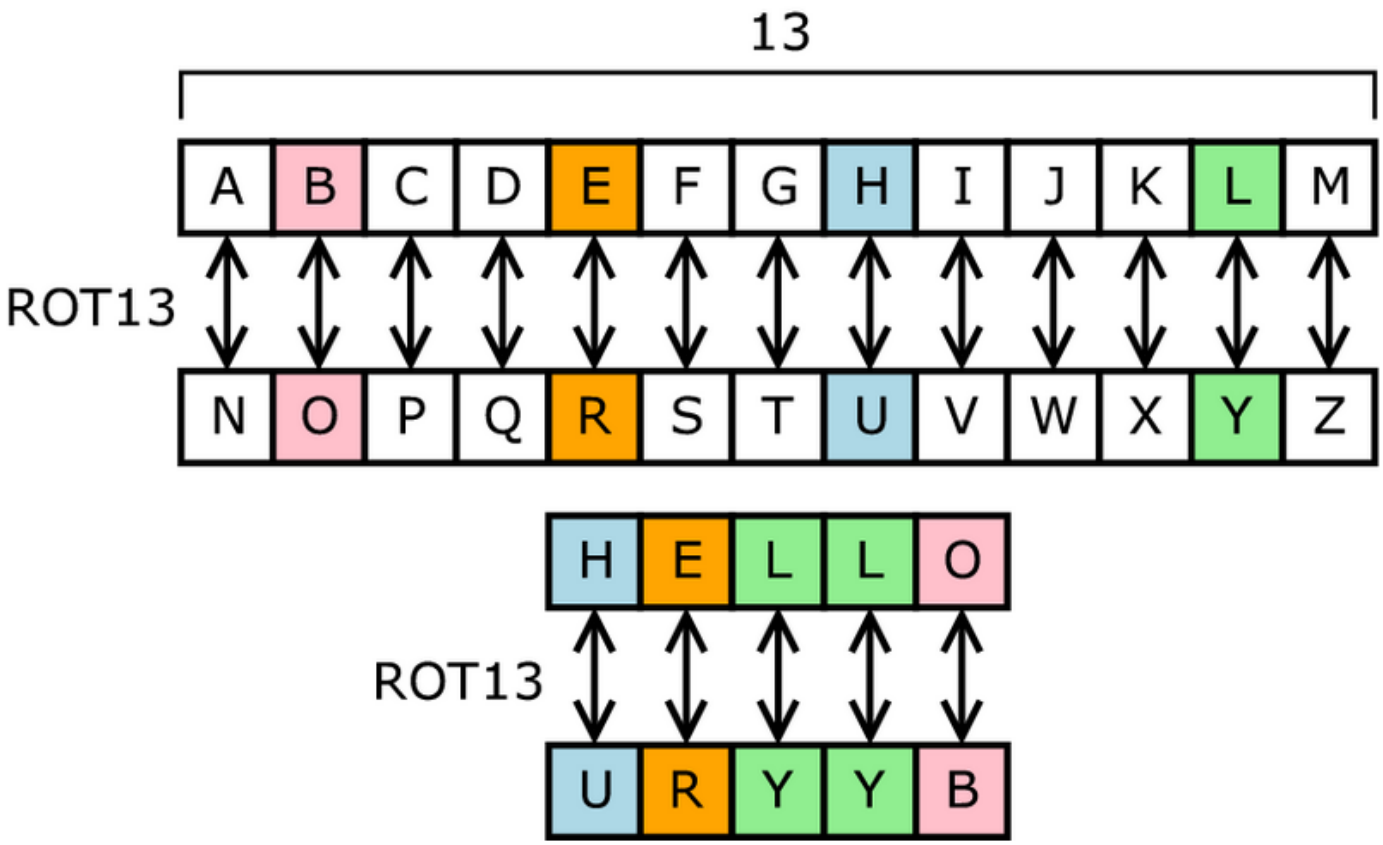
La substitution et la transposition sont deux méthodes historiques de sécurisation des données. Ces méthodes ne devraient plus être utilisées aujourd'hui pour protéger des données.

> **En sécurité, il n'y a rien de pire que d'avoir l'illusion qu'on est protégé**

Substitution

La **substitution** est un mécanisme par lequel chaque caractère du texte clair est remplacé par un autre caractère dans le texte chiffré. Cela signifie qu'on a une table de correspondance de chiffres, sons, mots ou autre à quelque chose. C'est par exemple le cas du code de césar dont nous avons parlé juste avant.

Voici, par exemple, la table de substitution du ROT13, qui est un code de César auquel la clé est à 13.

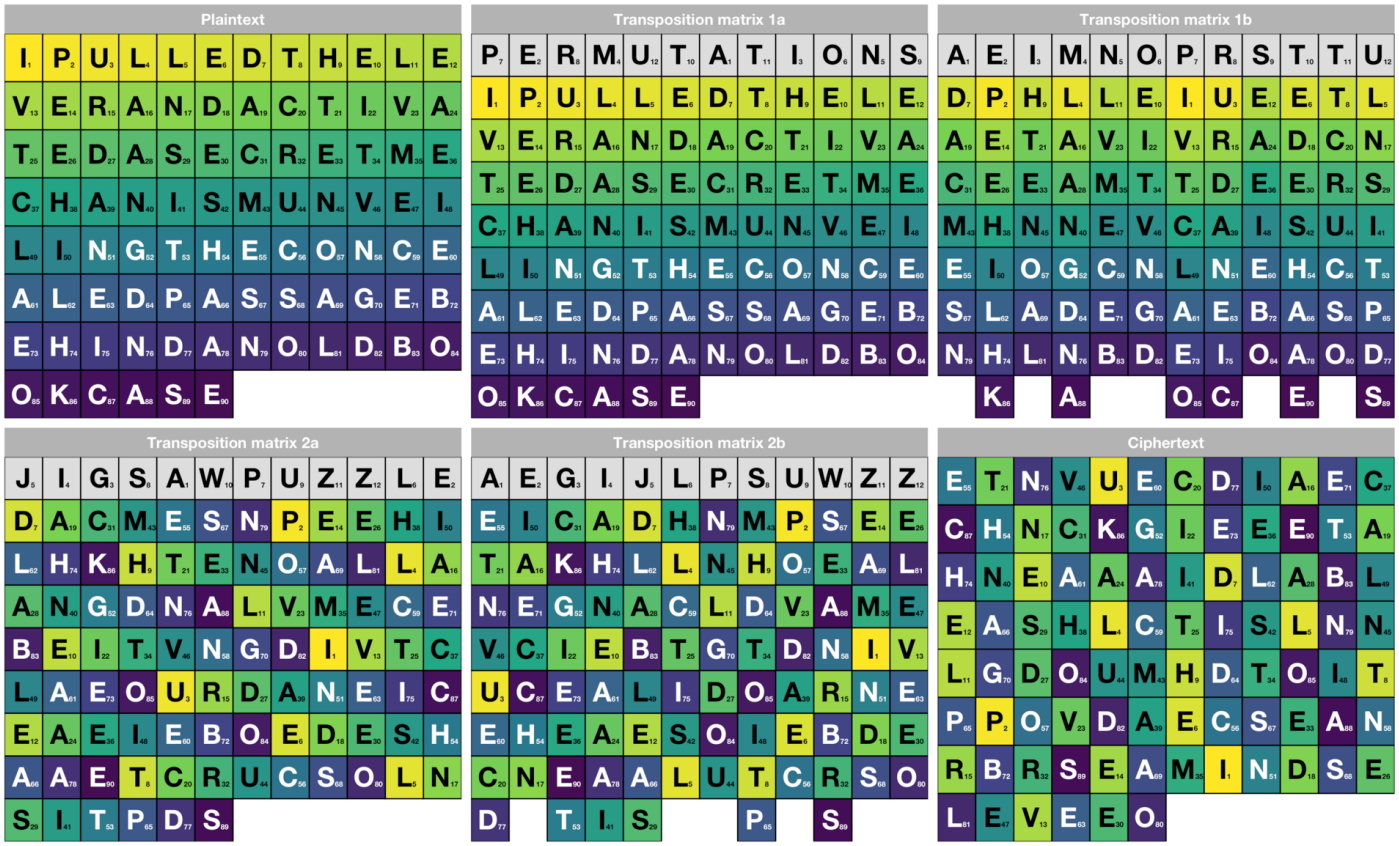


Le problème avec ce système est que l'on peut utiliser la linguistique et un contexte pour supposer la présence de certains mots ou la fréquence de certaines lettres.

Par exemple, si on sait que le texte est rédigé en français, on peut compter la lettre qui revient le plus souvent, supposer que c'est un E et compter la différence entre la lettre du cryptogramme et la lettre E pour obtenir la clé.

Transposition

La **transposition** consiste à mélanger les lettres d'une certaine manière.



> Ici par exemple, on commence par mettre le texte dans une grille, puis on ajoute un premier mot clé au-dessus et on trie les colonnes par ordre alphabétique de la clé.
>
> Ensuite, on transforme les colonnes en lignes et on met un deuxième mot clé au-dessus. On peut ensuite procéder de la même manière en triant les colonnes par ordre alphabétique de la clé. Le résultat de la grille donne donc le cryptogramme final.

Le problème avec cette méthode est similaire à celui de la substitution, on peut faire des inversions et tenter d'identifier des mots.

XOR (ou exclusif)

Le XOR est une opération de base du CPU, ce qui la rend très rapide.

L'idée ici est de faire une opération XOR entre un texte en clair et une clé de chiffrement. De la même manière, on peut déchiffrer le texte en faisant le cryptogramme XOR la clé.

Par exemple, si le message en clair est la lettre `A`, on convertit cela en binaire, ce qui donne `00001010`. Si notre clé est la lettre `H` que l'on convertit en binaire `01001000`. On peut faire un XOR dessus pour obtenir le cryptogramme :

```
   00001010 => A (texte en clair)
XOR 01001000 => H (clé)
   --------
   01000010 => B (cryptogramme)
```

Maintenant pour déchiffrer, il suffit de prendre le cryptogramme et la clé et de refaire un XOR

```
   01000010 => B (cryptogramme)
XOR 01001000 => H (clé)
   --------
   00001010 => A (texte en clair)
```

Le problème avec cette méthode est que si on connait un exemple dans lequel on a à la fois le texte clair et le cryptogramme, on peut retrouver la clé en utilisant le même mécanisme.

De la même manière, on peut facilement trouver la clé en faisant de l'analyse par fréquence sur base de la longueur de la clé.

Cependant, si la clé est complètement aléatoire et de la même longueur que le message, il s'agit alors d'un "one-time pad" ou "masque jetable" et c'est théoriquement un code incassable.

Masque jetable

Un masque jetable est une technique de chiffrement se basant sur une longue liste non répétitive et aléatoire de lettres (le masque). Chaque lettre est utilisée pour code une lettre du texte clair.

Par exemple, on peut additionner le rang de la lettre du masque avec celui du texte clair modulo 26 pour obtenir le rang de la lettre du texte chiffré.

Ou alors, on peut procéder en utilisant un XOR comme précédemment.

Puis ce que la clé est de la même longueur que le message et qu'elle est entièrement aléatoire, il est impossible de la déchiffrer. Cependant, cette technique a le gros désavantage d'avoir des clés très longues et peu pratiques.

Niveaux de chiffrements

Le niveau de chiffrement sera établi en fonction des groupes de personne contre lesquels on désire se protéger. Le niveau sera différent si on souhaite se protéger contre ses concurrents ou de la NSA.

C'est également une question de longueur de clé. Plus la clé est longue, plus ce sera sécurisé. Par exemple, un cadenas à trois chiffres aura 1 000 combinaisons possibles tandis qu'un cadenas à six chiffres aura 1 000 000 combinaisons possibles.

Algorithme comme garant de la sécurité

L'algorithme qui est utilisé pour faire de la cryptographie est le garant (avec la ou les clés) de la sécurité des messages.

Un bon algorithme doit être public (pour être vérifiable), sûr (éprouvé pendant plusieurs années et par des experts) et indépendant (sans coopération avec des organismes ayant des intérêts contradictoires tels que la NSA).

Dans un algorithme sûr, que l'espion ne soit qu'avec du texte chiffré, avec des correspondances texte en clair et texte chiffré ou même avec du texte clair choisi, l'espion ne peut pas trouver la clé.

Cryptographie symétrique et asymétrique

Entre les deux extrêmes que nous venons de voir (code de césar d'un côté et le one-time pad). Divers mécanismes ont été développés.

Crypto système à clé secrète (cryptographie symétrique)

Une clé secrète est partagée entre toutes les personnes qui doivent communiquer.

Les systèmes historiques correspondent à cette catégorie, mais aujourd'hui, on a également des manières plus sécurisées telles que AES.

Bien que ce crypto système ait été le standard pendant plusieurs siècles, il a quelques problèmes.

Par exemple, les clés doivent être distribuées et rester sûr, si la clé est compromise, tous les messages sont compromis. Et si une clé différente est utilisée par paire d'utilisateur, le nombre de clés nécessaires pour rester sûre devient très élevé.

Crypto système à clé publique (cryptographie asymétrique)

À la place d'avoir une clé secrète partagée par tout le monde, on va avoir une clé qui ne peut faire que du chiffrement (la clé publique), et une clé de déchiffrement (la clé privée).

La clé publique, comme son nom l'indique, peut être partagée partout. De cette manière, si on veut communiquer avec 100 personnes, à la place d'avoir 100 clés, on va avoir une seule clé publique partagée partout.

N'importe qui peut chiffrer un message avec cette clé publique, mais seule la clé privée pourra permettre de la déchiffrer.

Cela règle donc les problèmes de la cryptographie asymétrique, cependant ce système a le désavantage d'être beaucoup plus lourd et de demander beaucoup de calcul (ce qui est la raison pour laquelle il est si récent).

Il reste tout de même un problème, celui de la confiance. Comment pouvons-nous être sûrs que la personne qui donne la clé privée est bien qui elle prétend être ?

Pour cela, on peut utiliser des tiers de confiance déjà connus à l'avance qui pourront certifier des clés (c'est par exemple le cas de l'entreprise Let's Encrypt qui permet de certifier les clés TLS pour le HTTPS).

RSA

> Pour apprendre et comprendre le fonctionnement de RSA, allez voir cette playlist, pour avoir des détails sur le calcul uniquement, vous pouvez consulter cette vidéo et si vous voulez utiliser quelque chose de plus simple que l'algorithme d'Euclide étendu, vous pouvez utiliser le théorème de Bachet-Bézout à la place.

Voici comment calculer les clés publiques et privées en RSA,

1. Tout d'abord, on choisit deux nombres premiers, que l'on va appeler $ p $ et $ q $. Par exemple $ p = 3 $ et $ q = 5 $.
2. Ensuite, on fait le produit de ces deux nombres, que l'on va appeler $ n $, soit $ n = pq = 3 * 5 = 15 $
3. Ensuite, on calcule la fonction phi tel que $ \Phi (n) = (p - 1)(q - 1) = (3 - 1)(5 - 1) = 2 * 4 = 8$
4. Ensuite, on choisit un entier $ e $ dont le PGCD avec $ \Phi (n) $ vaut 1; autrement dit, il faut trouver un nombre $ e $ tel que $ e $ et $ \Phi (n) $ soient premiers entre eux (aucun facteurs premiers communs)
5. Enfin, il faut trouver le nombre de déchiffrement $ d $ tel que $ ed \mod \Phi (n) = 1 $, soit $ ed - kn = 1 $.
   - On peut ici utiliser le théorème de Bachet-Bézout qui dit que si deux nombres ($ a $ et $ b $) sont premiers entre eux, alors on peut trouver des entiers $ x $ et $ y $ tel que $ ax + by = 1$. Ici, on a $ e $ et $ \Phi (n) $ qui sont premiers entre eux, par conséquent on peut appliquer l'algorithme. En considérant $ x $ comme étant $ d $ et $ y $ comme étant $ k $. Note: si la valeur de d est négative on peut faire $ d + \Phi (n) $ pour avoir une valeur positive utilisable.
6. La clé publique est $ { e, n } $ et la clé privée est $ { d, n }$.

On peut donc maintenant chiffrer un message $ m $ (qui doit être inférieur à $ n $) en faisant $ m^e \mod n = c $. Et on peut déchiffrer un message en faisant $ c^d \mod n = m $. $

De même on peut signer un message en "déchiffrant" un texte en clair, $ m^d \mod n = s $ et on peut le vérifier en "chiffrant" la signature, $ s^e \mod n = m $.

Les chapitres ci-dessous sont optionnels pour le cours, mais aident à mieux comprendre le fonctionnement de RSA.

Exponentiation modulaire

Pour pouvoir avoir une clé pour déchiffrer et une clé pour chiffrer, il faut pouvoir trouver un moyen de faire une opération facilement (chiffrement avec clé secrète) mais de rendre l'opération inverse très compliquée (déchiffrement) si on ne connait pas une valeur supplémentaire (clé privée).

Cette fonction pour RSA c'est l'exponentiation modulaire, l'idée est que si on fait $ m^e \mod n = c $, cela demande beaucoup d'essai-erreur pour pouvoir en partant de e, n et c revenir à m.

Cependant, si on a un autre exposant (d) on peut l'inverser simplement en faisant $ c^d \mod n = m $.

Si on applique e et d ne même temps, le message ne change donc pas, ainsi $ m^{ed} \mod n = m $, cela sera important pour plus tard.

Factorisation de nombres premiers

Maintenant, il faut trouver un moyen de trouver e, d et n de manière à rendre tout cela possible. Pour cela, il faut trouver une autre fonction qui simple à faire dans un sens et compliquée à faire dans l'autre.

Cette fonction dans RSA c'est la factorisation de nombres premiers (pour rappel, un nombre premier est un nombre qui ne peut être divisé entièrement que par un ou lui-même). On sait que tous les nombres ont exactement une factorisation de nombres premiers, cependant, cette factorisation de plus en plus compliquée en fonction de la grandeur du nombre.

Cette propriété fait que la factorisation est un très bon candidat, car si on utilise des nombres premiers assez grands, il sera impossible de le factoriser avec nos moyens actuels.

Ainsi, on peut trouver deux nombres premiers très grands et les multiplier ensemble. Le produit de ces deux nombres premier sera très simple à calculer, mais très difficile à inverser parce que la multiplication est simple, mais la factorisation est elle très complexe.

Maintenant, il faut trouver une fonction qui dépend de la connaissance de la factorisation de n.

Indicatrice d'Euler, fonction Phi

Cette fonction, c'est indicatrice d'Euler que l'on va ici appeler phi. Ainsi la fonction $ \Phi (n) $ donne le nombre d'entiers positifs plus petit que $ n $ qui ne partagent pas de facteurs premiers avec $ n $. Par exemple $ \Phi (8) = 4 $ car huit ne partagent pas de facteurs communs avec 1, 3, 5 et 7, mais partagent des facteurs communs avec 2, 4 et 6.

Cette fonction est donc très compliquée à calculer pour des grands nombres, mais vraiment simple à calculer pour des nombres premiers. Puisqu'un nombre premier ne peut être divisé que par 1 ou lui-même, la fonction phi revient à dire $ \Phi (n) = n - 1 $.

De même la fonction est multiplicative, donc si $ a $ et $ b $ sont premiers, $ \Phi (a*b) = (a-1)(a-1) $.

Il faut maintenant trouver un moyen de lier la fonction phi à l'exponentiation modulaire.

Théorème d'Euler

le théorème d'Euler indique que $ a^{\Phi (n)} \mod n = 1 $ si a et $ \Phi (n) $ sont premiers entre eux.

On sait qu'un exposant peut être multiplié par n'importe quel nombre car cela ne changera pas le résultat du modulo. Donc $ a^{k \Phi (n)} \mod n = 1$ est vrai également. Une autre propriété est que si on incrémente l'exposant, alors cela devient $ a^{k \Phi (n) + 1 \mod n = a $.

Cela donne donc une forme similaire à $ m^{ed} \mod n = m $. On peut donc en déduire que $ ed = k \Phi (n) + 1 $, que l'on peut reformuler sous la forme $ 1 = ed + k \Phi (n) $, ou $ 1 = ed \mod \Phi (n) $.

Il suffit alors de trouver une valeur e qui soit premier avec $ \Phi (n) $ et trouver *d* en utilisant l'algorithme d'Euclide étendu ou le théorème de Bachet-Bezou.

Signatures cryptographiques

Une signature permet d'identifier que quelqu'un a bien écrit quelque chose. Une signature doit être authentique, non falsifiable, non réutilisable. De même, un document signé ne peut pas être modifié et une signature ne peut pas être reniée.

Il faut pouvoir faire toutes ces propriétés dans les signatures numériques (cryptographiques).

Avec de la cryptographie symétrique

Pour faire un système de signature avec une seule clé secrète, il faut trois acteurs, le signataire, le destinataire et un tiers de confiance. C'est le tiers de confiance qui donne toute sa sécurité à la structure.

Chaque acteur partage une clé avec le tiers de confiance. Ainsi lorsque le signataire va partager le document avec sa clé au tiers de confiance.

Le tiers de confiance peut donc confirmer la signature puisque qu'il connait la clé secrète. Le tiers de confiance peut donc ajouter une certification sur le document et le signer en utilisant la clé partagée avec le destinataire.

Le destinataire peut donc ensuite valider que le tiers de confiance a authentifié la signature et donc considérer la signature comme correcte, car le tiers de confiance a utilisé sa clé.

Le problème ici est que toute la sécurité du système réside dans le tiers de confiance, donc si le tiers de confiance est compromis, toutes les signatures sont compromises.

Avec de la cryptographie asymétrique

Avec de la cryptographie asymétrique, on a uniquement besoin du signataire et du destinataire. Le destinataire connait la clé publique du signataire. Le rôle d'un potentiel tiers de confiance ici est seulement d'authentifier le propriétaire de la clé (cela n'a donc besoin d'être fait qu'une seule fois).

Nous avons vu que lorsque quelque chose est chiffré avec une clé publique, seule la clé privée peut la déchiffrer. Mais à l'inverse, lorsque quelque chose est signé avec une clé privée, elle peut être validée par les clés publiques.

La procédure de signature équivaut globalement à appliquer l'algorithme de déchiffrement sur le texte à signer. Ainsi, il suffira au destinataire de chiffrer le résultat pour retrouver le texte d'origine.

Systèmes de confiances

Certificats numériques X509

Les certificats numériques x509 lient une clé publique à une identité (nom de domaine, adresse email, etc) en utilisant une signature cryptographique.

Toute personne faisant confiance au tiers connait la clé publique de ce dernier afin de pouvoir vérifier les certificats. C'est notamment cela qui est utilisé sur internet pour vérifier les connexions HTTPS. Le navigateur possède une liste de clés publiques de tiers de confiances pour vérifier les certificats.

Certains tiers de confiances offrent leurs services de vérification gratuitement, c'est par exemple le cas de "Let's Encrypt", cependant la vaste majorité sont payants, mais ont l'avantage d'offrir une vérification beaucoup plus rigoureuse.

Let's Encrypt ne fait que vérifier que la demande de certificat est bien faite depuis une machine sous le nom de domaine demandé, alors que d'autres tiers de confiance vont aller se renseigner sur l'entreprise et l'appeler pour avoir une confirmation de l'authenticité de la demande.

Système de PGP (Pretty Goog Privacy)

C'est un système à clé publique sans tiers de confiance. L'identité des personnes est garantie de manière transitive.

Ainsi, si Alice connait Bob, elle peut certifier sa clé publique en la signant. Bob peut alors partager la clé signée par Alice. De cette manière, toute personne faisant confiance aux fréquentations d'Alice pourra faire confiance à Bob en vérifiant la signature.