# Sécurité des systèmes informatiques

Alex Auvolat, Nissim Zerbib

4 avril 2014

# Introduction

La sécurité est une chaîne : elle est aussi solide que le plus faible de ses maillons.

Facteurs les plus souvent en cause :

- erreur humaine
- faille dans l'implémentation

Intérêts nationaux et économiques majeurs mis en cause :

- certaines administrations et certains dirigeants des principales puissances mondiales sont encore vulnérables !
- vol d'informations et de technologies
- cybercriminalité et surveillance des populations

Bien sûr les illustrations ne sont pas de nous.

# Niveau de conscience de l'utilisateur lambda quant à l'importance de ces questions

- Une majorité de gens utilise des systèmes peu sécurisés.
- Les mots de passe les plus utilisés actuellement : 123456, password, motdepasse, qwerty, azerty etc.
- Certains chefs d'état et de services de renseignements, de polices manquent d'une formation basique à la cryptographie. Entre la corbeille non vidée de l'affaire Clear Stream, les agendas papiers de gens qui touchent 500000 euros de prime d'indic en liquide, ou un certain Paul Bismuth qui pense échapper à des écoutes en changeant de portable, il y a encore fort à faire pour éveiller les esprits.
- Pendant ce temps, les services secrets de divers pays écoutent toutes les communications mondiales, de nombreux cyber-attaquants sont aux aguets.

# Trouver le bon équilibre

Il faut adapter la sécurité de son infrastructure à l'ampleur de la menace et à la chaîne entière. Inutile de tout chiffrer si les utilisateurs sont totalement imprudents.

# I. Failles de sécurité et exploitation

# La sécurité sous Unix

Entités fondamentales pour la sécurité :

- utilisateurs
- groupes d'utilisateurs (par exemple `users` ou `wheel`)

# La sécurité sous Unix

Entités fondamentales pour la sécurité :

- utilisateurs
- groupes d'utilisateurs (par exemple `users` ou `wheel`)

Permissions pour un fichiers :

- trois bits de droits (lecture, écriture, exécution)
- attribués à trois entités (utilisateur, groupe, autre)
- plus le bit `setuid`

# Le bit `setuid` : un bon modèle de sécurité ?

- Un programme appartenant à `root` et ayant le bit `setuid` sera exécuté en tant que `root` dans tous les cas, même s'il est lancé par un utilisateur

# Le bit `setuid` : un bon modèle de sécurité ?

- Un programme appartenant à `root` et ayant le bit `setuid` sera exécuté en tant que `root` dans tous les cas, même s'il est lancé par un utilisateur
- Intérêt : permet l'implémentation de commandes telles que `su` ou `sudo`.

# Le bit `setuid` : un bon modèle de sécurité ?

- Un programme appartenant à `root` et ayant le bit `setuid` sera exécuté en tant que `root` dans tous les cas, même s'il est lancé par un utilisateur
- Intérêt : permet l'implémentation de commandes telles que `su` ou `sudo`.
- Inconvénient : si le programme est vulnérable (*buffer overflow* par exemple), ou s'il a été modifié, cela permet une escalade des privilèges.

# Le `chroot` : un bon modèle de sécurité ?

- C'est un appel système qui permet de restreindre la portion du système de fichiers qui est visible pour l'application.

# Le `chroot` : un bon modèle de sécurité ?

- C'est un appel système qui permet de restreindre la portion du système de fichiers qui est visible pour l'application.
- Intérêt : exécuter un service ou un programme dangereux dans un environnement restreint, où il ne pourra pas *faire trop de mal*.

# Le `chroot` : un bon modèle de sécurité ?

- C'est un appel système qui permet de restreindre la portion du système de fichiers qui est visible pour l'application.
- Intérêt : exécuter un service ou un programme dangereux dans un environnement restreint, où il ne pourra pas *faire trop de mal*.
- Inconvénient : un programme qui tourne en tant que `root` dans un environnement chrooté peut facilement s'échapper. D'autre part, seul `root` peut appeller l'appel système `chroot`. Il faut donc penser à toujours abandonner ses privilèges `root` après avoir fait un `chroot`.
  `http://www.bpfh.net/simes/computing/chroot-break.html`

# Le `chroot` : un bon modèle de sécurité ?

- C'est un appel système qui permet de restreindre la portion du système de fichiers qui est visible pour l'application.
- Intérêt : exécuter un service ou un programme dangereux dans un environnement restreint, où il ne pourra pas *faire trop de mal*.
- Inconvénient : un programme qui tourne en tant que `root` dans un environnement chrooté peut facilement s'échapper. D'autre part, seul `root` peut appeller l'appel système `chroot`. Il faut donc penser à toujours abandonner ses privilèges `root` après avoir fait un `chroot`.
  `http://www.bpfh.net/simes/computing/chroot-break.html`
- D'autres méthodes d'isolement ont été développées : Jails de FreeBSD, virtualisation, ...

# La faille classique : le *buffer overflow*

- Origine de la faille : erreur de programmation (omission volontaire ou erreur d'inattention)

# La faille classique : le *buffer overflow*

- Origine de la faille : erreur de programmation (omission volontaire ou erreur d'inattention)
- Tous les programmes écrits dans un langage non managé comme C ou C++ sont vulnérables

# La faille classique : le *buffer overflow*

- Origine de la faille : erreur de programmation (omission volontaire ou erreur d'inattention)
- Tous les programmes écrits dans un langage non managé comme C ou C++ sont vulnérables
- En particulier : le noyau, les services réseau (serveur mail, web, SQL, ...), etc.

# La faille classique : le *buffer overflow*

- Origine de la faille : erreur de programmation (omission volontaire ou erreur d'inattention)
- Tous les programmes écrits dans un langage non managé comme C ou C++ sont vulnérables
- En particulier : le noyau, les services réseau (serveur mail, web, SQL, ...), etc.
- L'exploitation d'un *buffer overflow* permet d'exécuter du code arbitraire (injection de code)

# Le *buffer overflow* en pratique

Désolé, pas de démo (ça marche pas...)

http://www.skullsecurity.org/wiki/index.php/Example_4

# Stratégies de protection

- *Addess space layout randomization* : on met la pile et le tas à des adresses aléatoires, pour qu'il soit plus dur d'y écrire du code qui sera à une position prévisible. Il est donc plus dur de déterminer, entre autres, l'adresse à laquelle faire un *jump*.

# Stratégies de protection

- *Addess space layout randomization* : on met la pile et le tas à des adresses aléatoires, pour qu'il soit plus dur d'y écrire du code qui sera à une position prévisible. Il est donc plus dur de déterminer, entre autres, l'adresse à laquelle faire un *jump*.
- *Data execution prevention* : on sépare les parties de la mémoire qui sont du code, que l'on met en lecture seule, et les parties qui sont des données sur lesquelles on peut ecrire (typiquement, la pile). On interdit l'exécution des données (c'est le CPU qui gère ça)

# Stratégies de protection

- *Addess space layout randomization* : on met la pile et le tas à des adresses aléatoires, pour qu'il soit plus dur d'y écrire du code qui sera à une position prévisible. Il est donc plus dur de déterminer, entre autres, l'adresse à laquelle faire un *jump*.
- *Data execution prevention* : on sépare les parties de la mémoire qui sont du code, que l'on met en lecture seule, et les parties qui sont des données sur lesquelles on peut ecrire (typiquement, la pile). On interdit l'exécution des données (c'est le CPU qui gère ça)
- Certification statique des programmes : tout le monde connaît

# Plus sur les *buffer overflow*

# Attaques sur le réseau : les attaques DoS et DDoS

- Attaque DoS : *denial of service*. Principe : on envoie un nombre massif de requêtes à un serveur pour l'empêcher d'effectuer son travail correctement.

# Attaques sur le réseau : les attaques DoS et DDoS

- Attaque DoS : *denial of service*. Principe : on envoie un nombre massif de requêtes à un serveur pour l'empêcher d'effectuer son travail correctement.
- De telles attaques peuvent être empêchées en bloquant tout le trafic venant d'une addresse IP si on voit qu'elle abuse. De plus, elles sont de puissance limitée.

# Attaques sur le réseau : les attaques DoS et DDoS

- Attaque DoS : *denial of service*. Principe : on envoie un nombre massif de requêtes à un serveur pour l'empêcher d'effectuer son travail correctement.
- De telles attaques peuvent être empêchées en bloquant tout le trafic venant d'une addresse IP si on voit qu'elle abuse. De plus, elles sont de puissance limitée.
- Attaque DDos : *distributed denial of service*. Principe : on utilise un nombre important de machines connectées au réseau pour lancer toutes ces requêtes.

# Attaques sur le réseau : les attaques DoS et DDoS

- Attaque DoS : *denial of service*. Principe : on envoie un nombre massif de requêtes à un serveur pour l'empêcher d'effectuer son travail correctement.
- De telles attaques peuvent être empêchées en bloquant tout le trafic venant d'une addresse IP si on voit qu'elle abuse. De plus, elles sont de puissance limitée.
- Attaque DDos : *distributed denial of service*. Principe : on utilise un nombre important de machines connectées au réseau pour lancer toutes ces requêtes.
- De telles attaques sont bien plus difficiles à mitiger.

# Attaques sur le réseau : les attaques DoS et DDoS

- Attaque DoS : *denial of service*. Principe : on envoie un nombre massif de requêtes à un serveur pour l'empêcher d'effectuer son travail correctement.
- De telles attaques peuvent être empêchées en bloquant tout le trafic venant d'une addresse IP si on voit qu'elle abuse. De plus, elles sont de puissance limitée.
- Attaque DDos : *distributed denial of service*. Principe : on utilise un nombre important de machines connectées au réseau pour lancer toutes ces requêtes.
- De telles attaques sont bien plus difficiles à mitiger.
- En pratique, ce sont souvent des *botnets* qui sont utilisés pour les

# Attaques sur le réseau : les attaques DoS et DDoS

- Attaque DoS : *denial of service*. Principe : on envoie un nombre massif de requêtes à un serveur pour l'empêcher d'effectuer son travail correctement.
- De telles attaques peuvent être empêchées en bloquant tout le trafic venant d'une addresse IP si on voit qu'elle abuse. De plus, elles sont de puissance limitée.
- Attaque DDos : *distributed denial of service*. Principe : on utilise un nombre important de machines connectées au réseau pour lancer toutes ces requêtes.
- De telles attaques sont bien plus difficiles à mitiger.
- En pratique, ce sont souvent des *botnets* qui sont utilisés pour les
- Les hacktivistes d'*anonymous* aiment bien ce genre d'attaques...

# Étude de cas : amplification DNS

- Le système DNS a pour but de nous permettre de nous y retrouver dans l'internet : il rend possible la résolution nom de domaine vers adresse IP.

# Étude de cas : amplification DNS

- Le système DNS a pour but de nous permettre de nous y retrouver dans l'internet : il rend possible la résolution nom de domaine vers adresse IP.
- Le protocole DNS utilise des sockets UDP. Conséquence : on peut envoyer une requête DNS en se faisant passer pour n'importe qui, et la réponse est envoyée à cette personne.

# Étude de cas : amplification DNS

- Le système DNS a pour but de nous permettre de nous y retrouver dans l'internet : il rend possible la résolution nom de domaine vers adresse IP.
- Le protocole DNS utilise des sockets UDP. Conséquence : on peut envoyer une requête DNS en se faisant passer pour n'importe qui, et la réponse est envoyée à cette personne.
- Il existe une requêtes DNS qui demande une liste de *tous les domaines gérés par le serveur en question*...

# Étude de cas : amplification DNS

- Le système DNS a pour but de nous permettre de nous y retrouver dans l'internet : il rend possible la résolution nom de domaine vers adresse IP.
- Le protocole DNS utilise des sockets UDP. Conséquence : on peut envoyer une requête DNS en se faisant passer pour n'importe qui, et la réponse est envoyée à cette personne.
- Il existe une requêtes DNS qui demande une liste de *tous les domaines gérés par le serveur en question*...

vous voyez où je veux en venir ?

# Les failles du web : les injections SQL

Prenez par exemple la requête SQL suivante :

```
statement = "SELECT * FROM users ".
            "WHERE name ='" + userName + "';"
```

# Les failles du web : les injections SQL

Prenez par exemple la requête SQL suivante :

```
statement = "SELECT * FROM users ".
            "WHERE name ='" + userName + "';"
```

Maintenant, supposons que la variable `userName` soit définie librement par l'utilisateur. Par exemple :

```
' or '1'='1
```

# Les failles du web : les injections SQL

Prenez par exemple la requête SQL suivante :

```
statement = "SELECT * FROM users ".
            "WHERE name ='" + userName + "';"
```

Maintenant, supposons que la variable `userName` soit définie librement par l'utilisateur. Par exemple :

```
' or '1'='1
```

Ou pire encore :

```
a';DROP TABLE users; SELECT * FROM groups WHERE 't' = 't
```

# Les failles du web : les injections SQL

- Ce genre de failles arrive souvent sur des sites Web codés avec les pieds en PHP (mais pas que !)

# Les failles du web : les injections SQL

- Ce genre de failles arrive souvent sur des sites Web codés avec les pieds en PHP (mais pas que !)
- Conséquence : on peut voler des informations (typiquement, liste de nom d'utilisateurs/emails/mots de passes), et exécuter n'importe quelle modification sur le serveur.

# Les failles du web : les injections SQL

- Ce genre de failles arrive souvent sur des sites Web codés avec les pieds en PHP (mais pas que !)
- Conséquence : on peut voler des informations (typiquement, liste de nom d'utilisateurs/emails/mots de passes), et exécuter n'importe quelle modification sur le serveur.
- La solution la plus élémentaire consiste à rajouter les caractères d'échapement qu'il faut au bon endroit, avec une fonction du type `mysql_real_escape_string` (fonction dépréciée de PHP).

# Les failles du web : les failles XSS

- XSS : *cross-site scripting*

# Les failles du web : les failles XSS

- XSS : *cross-site scripting*
- Vulnérabilité attaquée : Javascript, ou autres possibilités de scripter le client (Flash, Java).

# Les failles du web : les failles XSS

- XSS : *cross-site scripting*
- Vulnérabilité attaquée : Javascript, ou autres possibilités de scripter le client (Flash, Java).
- Il est parfois possible d'injecter du code Javascript dans les pages d'un site web mal sécurisé.

# Les failles du web : les failles XSS

- XSS : *cross-site scripting*
- Vulnérabilité attaquée : Javascript, ou autres possibilités de scripter le client (Flash, Java).
- Il est parfois possible d'injecter du code Javascript dans les pages d'un site web mal sécurisé.
- Schéma de l'attaque : insérer un code javascript malicieux sur une page que d'autres utilisateurs sont susceptibles de visiter. Ce script peut, par exemple, capturer les cookies de l'utilisateur et les transmettre à l'attaquant, ce qui lui permettera d'usurper l'identité de la victime.

# Les failles du web : les failles XSS

- XSS : *cross-site scripting*
- Vulnérabilité attaquée : Javascript, ou autres possibilités de scripter le client (Flash, Java).
- Il est parfois possible d'injecter du code Javascript dans les pages d'un site web mal sécurisé.
- Schéma de l'attaque : insérer un code javascript malicieux sur une page que d'autres utilisateurs sont susceptibles de visiter. Ce script peut, par exemple, capturer les cookies de l'utilisateur et les transmettre à l'attaquant, ce qui lui permettera d'usurper l'identité de la victime.
- Sécurisation : il faut empêcher l'utilisation de balises `<script>` par les utilisateurs du site, avec un filtrage approprié (on peut vouloir garder d'autres balises HTML par ailleurs).

# Les failles du web : les failles CSRF

- CSRF : *cross-site request forgeries*

# Les failles du web : les failles CSRF

- CSRF : *cross-site request forgeries*
- Idée : on veut effectuer une requête sur un site pour, par exemple, supprimer un message, mais on n'a pas les droits nécessaires pour effectuer l'action. On veut donc faire faire cette action par quelqu'un qui a les droits, à son insu.

# Les failles du web : les failles CSRF

- CSRF : *cross-site request forgeries*
- Idée : on veut effectuer une requête sur un site pour, par exemple, supprimer un message, mais on n'a pas les droits nécessaires pour effectuer l'action. On veut donc faire faire cette action par quelqu'un qui a les droits, à son insu.
- Méthode : envoyer à l'administrateur du forum un message privé avec une image dont l'adresse est l'adresse de la page à appeller pour supprimer le message cible. Cette image sera automatiquement chargée par le navigateur, ce qui causera la suppression du message.

# Les failles du web : les failles CSRF

- CSRF : *cross-site request forgeries*
- Idée : on veut effectuer une requête sur un site pour, par exemple, supprimer un message, mais on n'a pas les droits nécessaires pour effectuer l'action. On veut donc faire faire cette action par quelqu'un qui a les droits, à son insu.
- Méthode : envoyer à l'administrateur du forum un message privé avec une image dont l'adresse est l'adresse de la page à appeller pour supprimer le message cible. Cette image sera automatiquement chargée par le navigateur, ce qui causera la suppression du message.
- Sécurisation : utiliser des *tokens* à usage unique pour toutes les actions sensibles de ce type ; vérifier l'en-tête *referer* ; nécessiter des requêtes POST pour les actions sensibles.

# Les attaques Man-in-the-Middle

Illustration avec Wireshark.

# Exploitation de failles humaines : l'ingéniérie sociale

- Exemple classique : le phishing...

# Exploitation de failles humaines : l'ingéniérie sociale

- Exemple classique : le phishing...
- Principe : abuser de la crédulité des gens pour obtenir des informations qui nous seraient autrement privées.

# Exploitation de failles humaines : l'ingéniérie sociale

- Exemple classique : le phishing...
- Principe : abuser de la crédulité des gens pour obtenir des informations qui nous seraient autrement privées.
- Méthode : le plus souvent, se faire passer pour quelqu'un que l'on n'est pas (ex : un administrateur système).

# Exploitation de failles humaines : l'ingéniérie sociale

- Exemple classique : le phishing...
- Principe : abuser de la crédulité des gens pour obtenir des informations qui nous seraient autrement privées.
- Méthode : le plus souvent, se faire passer pour quelqu'un que l'on n'est pas (ex : un administrateur système).
- Sécurisation : impossible (il y aura toujours quelqu'un de suffisament abruti pour donner son mot de passe)

# Introduction de failles dans le code source

La plupart des logiciels propriétaires possèdent (probablement) des backdoors :
https://www.schneier.com/blog/archives/2013/09/the_nsa_is_brea.html

Mais les logiciels libres ne sont pas non plus à l'abris :

http://www.infoq.com/news/2013/10/Linux-Backdoor?utm_source=tuicool

Pour ceux qui voudraient utiliser un maximum de services et logiciels libres, sûrs etc. :
https://prism-break.org/fr/

# Exploitation de failles humaines : l'ingénierie sociale

Mêmes les infrastructures les mieux protégées ne sont pas à l'abri :

http://www.thesecurityblogger.com/?p=1903

# Failles et fuites matérielles

Tempest

Article Shamir : Unsafe and sound :
https://www.cs.tau.ac.il/~tromer/acoustic/

Composants électroniques

# II. Cryptographie

# Objectifs de la cryptographie

- Confidentialité : le message reste secret

# Objectifs de la cryptographie

- Confidentialité : le message reste secret
- Authentification : le message vient bien de la bonne entité

# Objectifs de la cryptographie

- Confidentialité : le message reste secret
- Authentification : le message vient bien de la bonne entité
- Intégrité : le message n'a pas été modifé

# Objectifs de la cryptographie

- Confidentialité : le message reste secret
- Authentification : le message vient bien de la bonne entité
- Intégrité : le message n'a pas été modifé
- Anonymat : on ne peut trouver de qui vient le message

%


YOU SHOULD NOT PUBLISH YOUR CRYPTO ALGORITHM.
IT MAKES IT HARDER TO BREAK.
memegenerator.net

%


USE YOUR OWN
"CRYPTO"
HOW ABOUT NO
memegenerator.net

# Fondamentaux

- On n'invente pas son propre algorithme de cryptographie (à moins d'être cryptologue) ! Il est probable qu'il soit vulnérable. Cela vaut aussi pour les protocoles, implémentations, etc. . .
- Principe numéro 2 de Kerchoffs : Il [le cryptosystème] faut qu'il n'exige pas le secret, et qu'il puisse sans inconvénient tomber entre les mains de l'ennemi.

Le seul secret doit être un paramètre de l'algorithme : la clé.

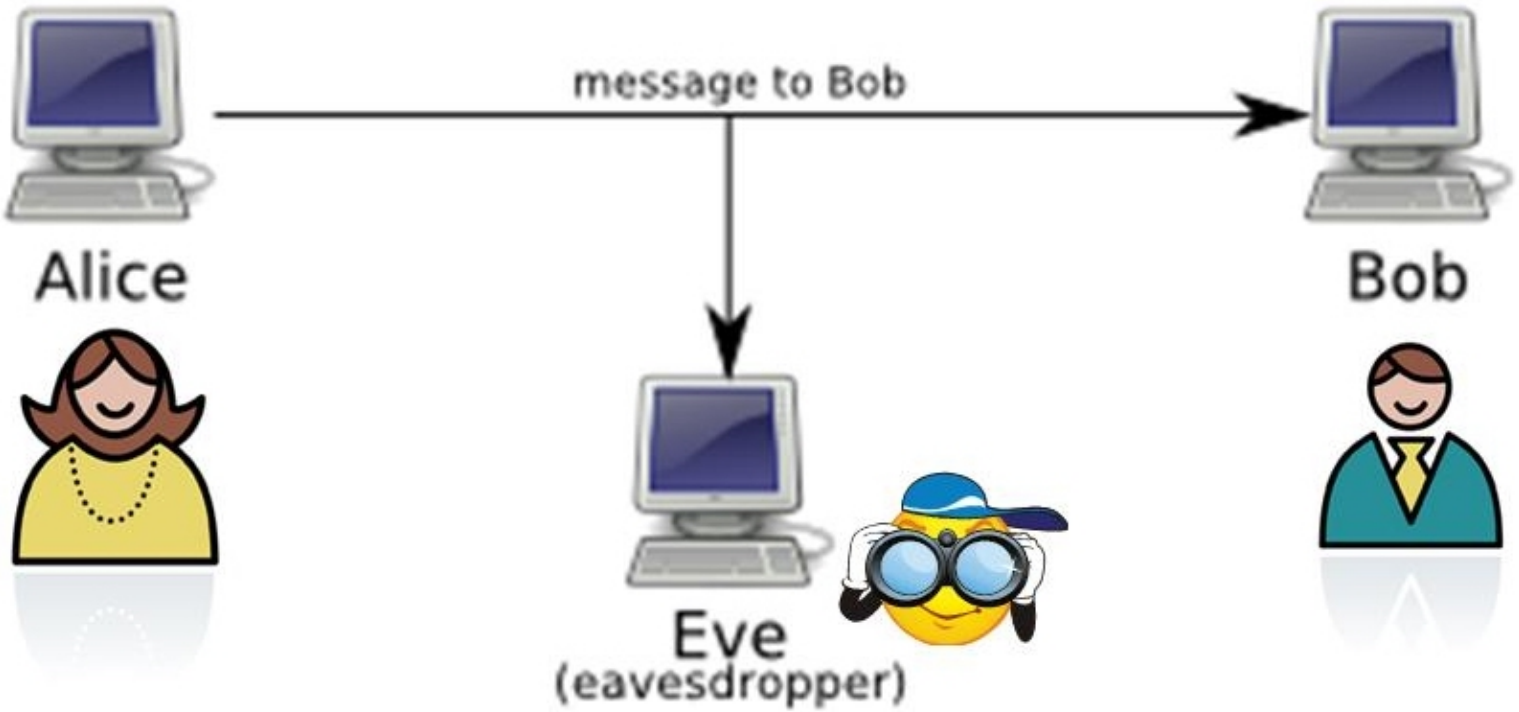
# Cryptographie symétrique

Le principe général : le chiffrement doit cacher les propriétés statistiques du message. Chaque bit du message crypté doit idéalement dépendre de tous les bits du message original. Le nombre de clés disponibles doit être grand pour empêcher une recherche exhaustive.

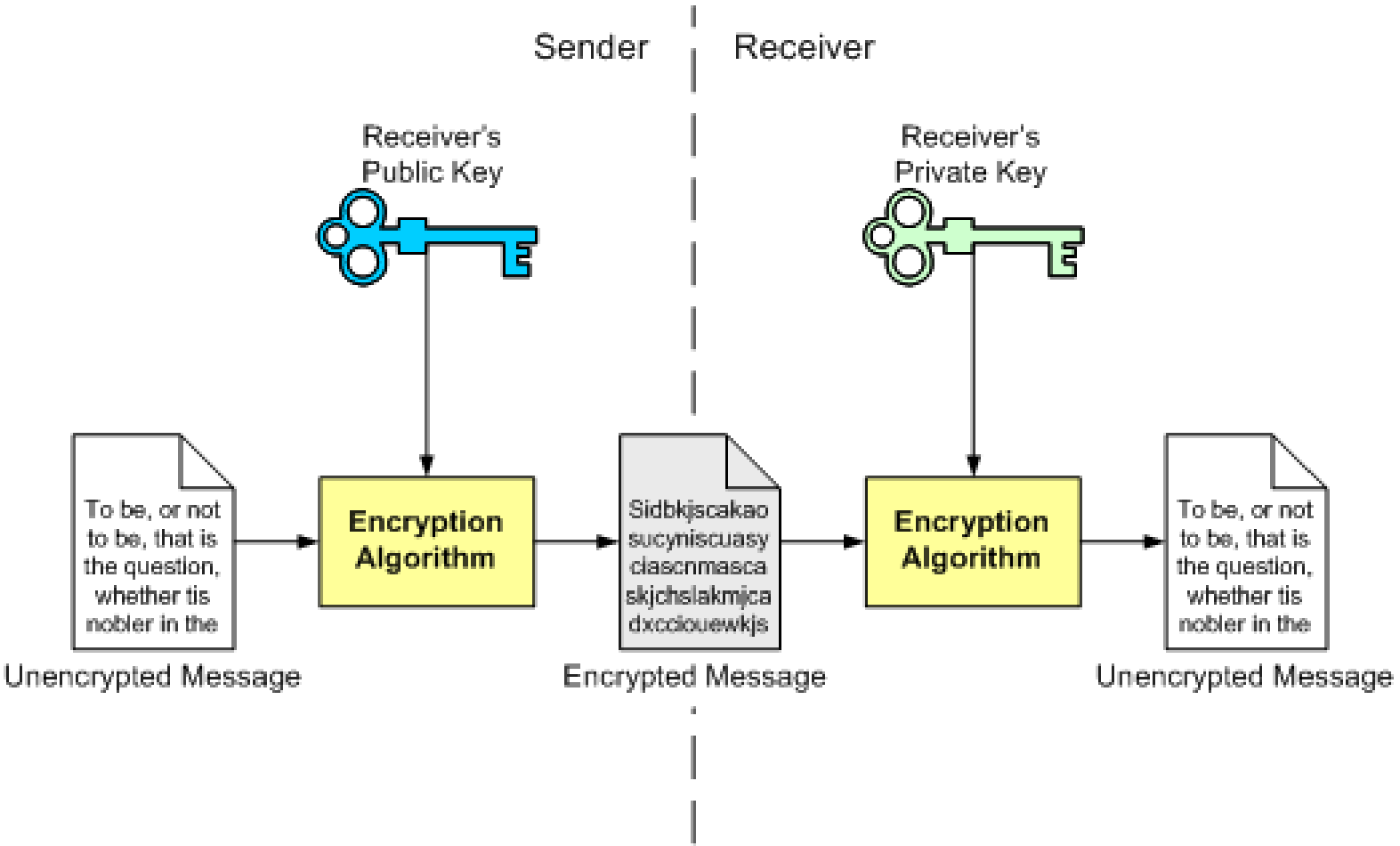
- Primitif : César, Vigenère
- Standards : DES, AES et autres
- Modes de chiffrement

# Scénario 1 : envoi de mail chiffré et cryptographie asymétrique

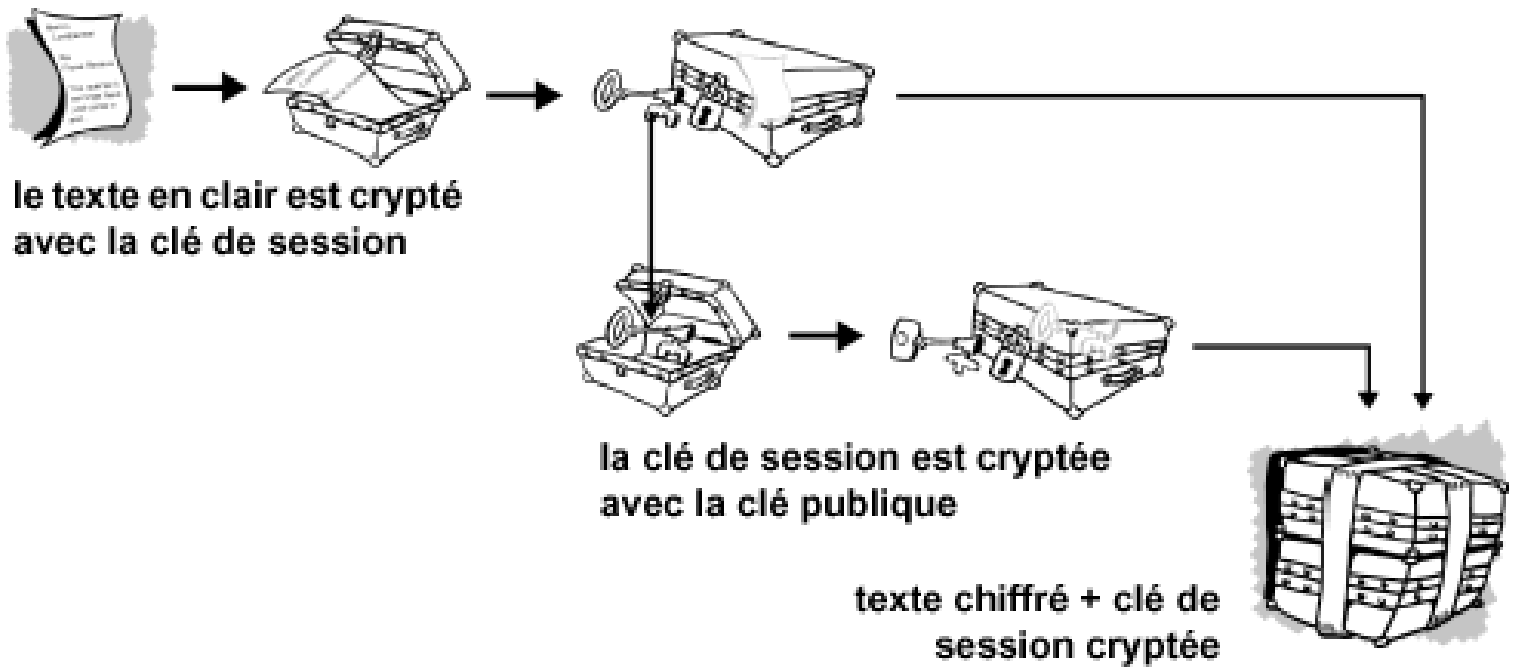
## Comment échanger la clé ?



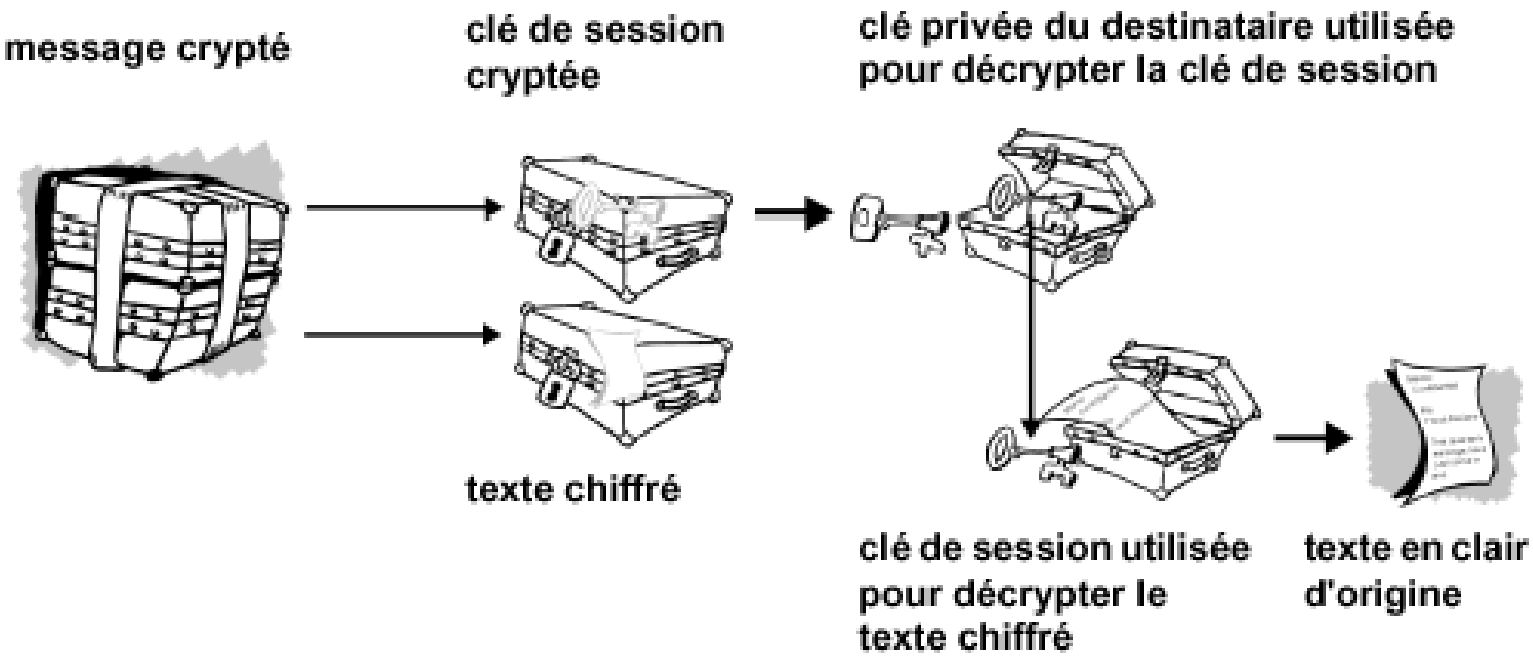
# Solution : cryptographie asyymétrique, GPG, explication technique



# Fonctionnement : chiffrage



# Fonctionnement : déchiffrage



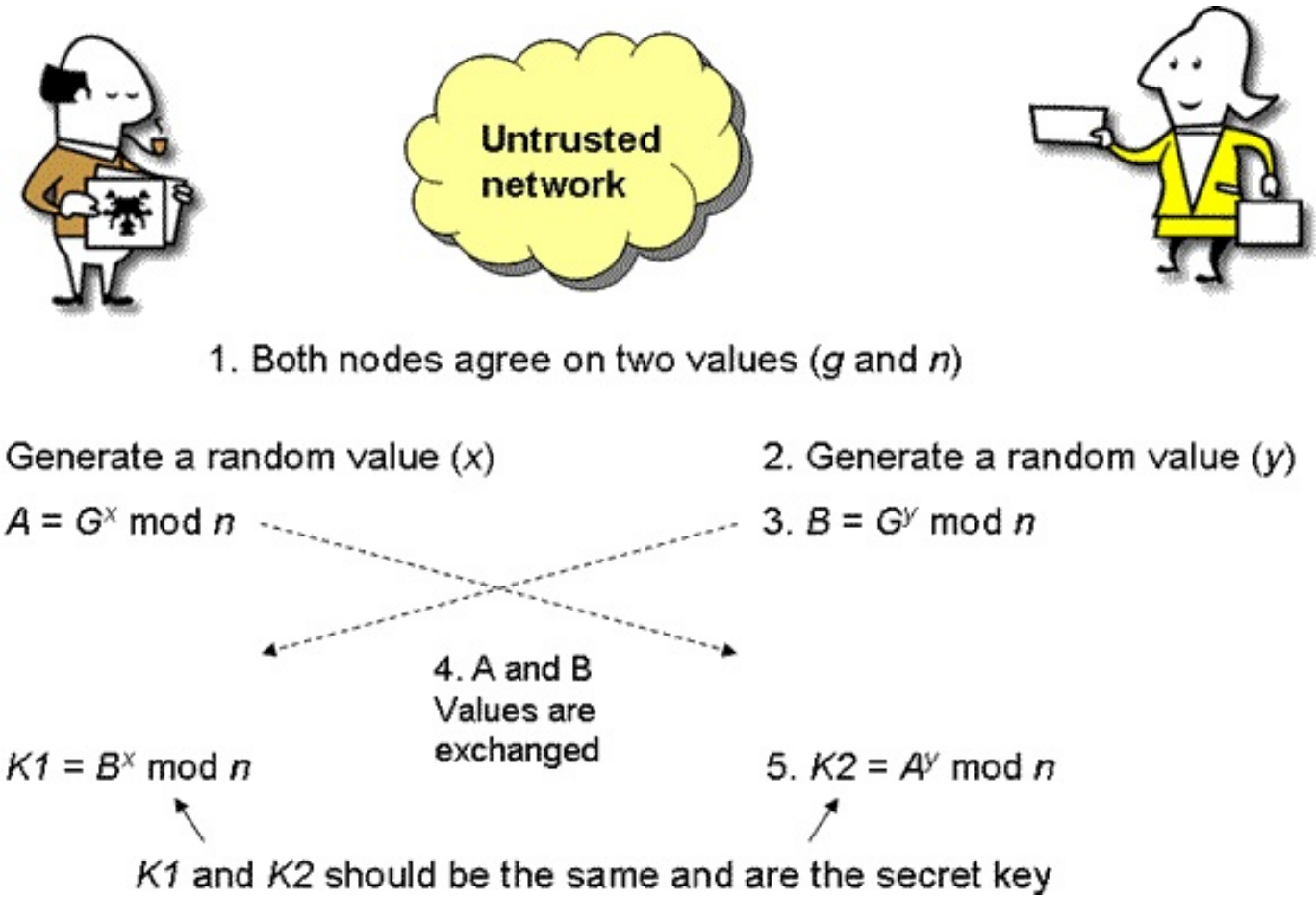
# L'exemple d'EL Gamal

- Repose sur le logarithme discret.
- Proche de Diffie-Hellman, mais une des parties conserve un des paramètres.

# Scénario 2 : communication sécurisée

Empêcher les attaques de type Man in the Middle : comment être sûr qu'on se connecte bien au bon serveur.

# Solution : SSL, explication technique



# Scénario 3 : stockage de mots de passe

Pour les services webs : L'astuce est de ne jamais les stocker !

Faille Adobe : les mots de passe étaient chiffrés et pourtant...

http://nakedsecurity.sophos.com/2013/11/04/anatomy-of-a-password-disaster-adobes-giant-sized-cryptographic-blunder/

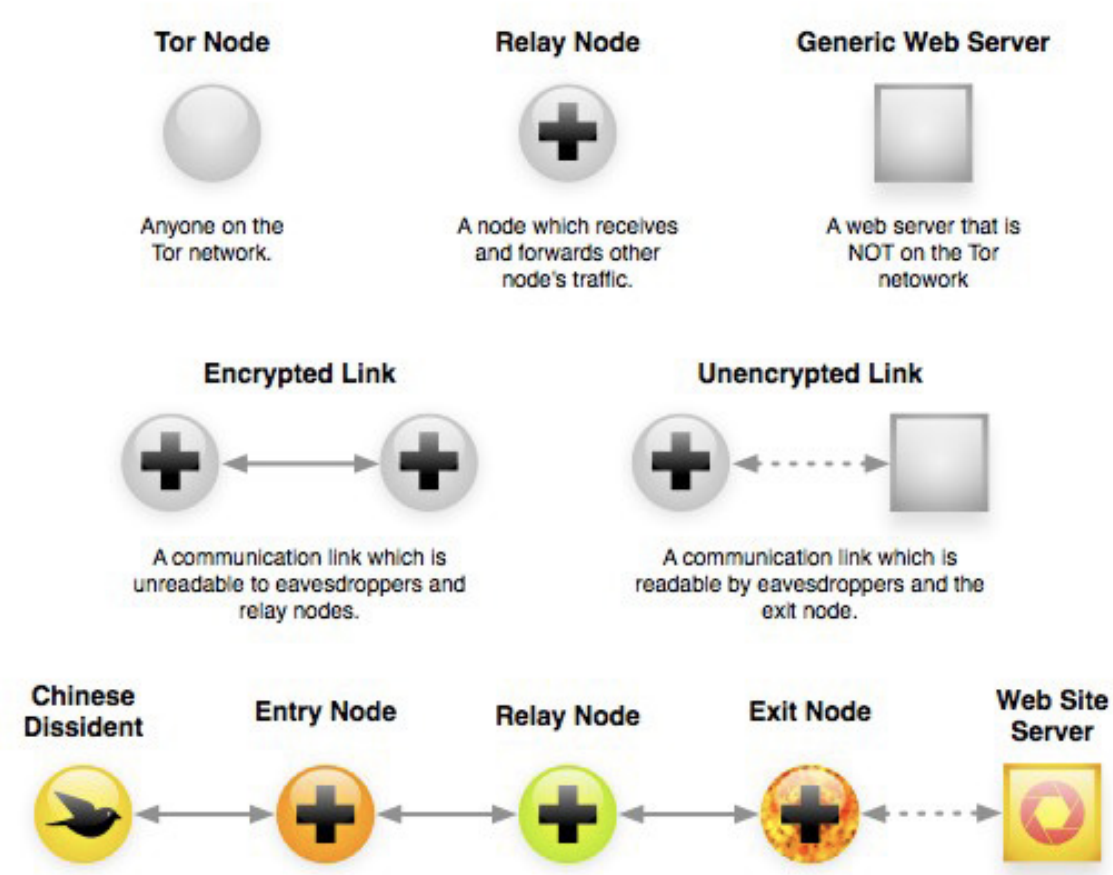
# Solution : le hachage, explication technique

- Utiliser des algorithmes à jour : attaques connu sur SHA-1
- Ajouter du sel pour empêcher l'utilisation efficace de tables précalculées.

https://security.stackexchange.com/questions/211/how-to-securely-hash-passwords

# Scénario 4 : anonymiser ses actions sur Internet

L'exemple de Tor : à partir de primitives cryptographiques de base, on peut réaliser différentes architectures.



# III. Logiciels malicieux

# Les virus

- Un virus est un programme auto-reproducteur.
- Charge finale : le virus est vecteur de l'attaque.
- Polymorphisme : résister à la détection. Dissimuler son code et son comportement.

# Autres malwares

# Conclusion

La mise en place et le maintien d'une infrastructure sécurisée repose à la fois sur une expertise (respect à la lettre des standards cryptographiques, utilisation de logiciels reconnus) et sur une bonne connaissance des utilisateurs.

On n'est jamais à l'abri...