Cryptographie (1)

Introduction

Au programme

- Fondamentaux d’un crypto-système
- Types de crypto-systèmes :
  - cryptographie symétrique
  - cryptographie asymétrique
  - hachage (*hashing*)

Cryptographie

- *Cryptos* + *graphia* : « écrits cachés »
- **Chiffrer** : transformer un message pour **cacher** son contenu
- **Déchiffrer** : processus inverse (*nécessite la clé de déchiffement*)
- **Décrypter** : idem mais *sans avoir la clé*
- ~~Crypter~~ : on ne peut pas chiffrer *sans une clé*
- **Message** : *n'importe quoi* dans sa forme non chiffrée (un fichier)

Clé (*Key*)

- **Clé** : paramètre des algorithmes de cryptographie
  - ⇒ clé différente produit message chiffré différent
- Une fois les données chiffrées, elles sont protégées (confidentialité, contrôle d'intégrité)
- Seulement **du moment que la clé est correctement protégée**
- La gestion des clés devient l'élément primordial par lequel la protection est assurée

Algorithme de cryptographie

- **Entrées** :
  - Message original (Mo) et clé (K) = données binaires
- **Sortie** :
  - Message chiffré (Mc) = données binaires
- **Chiffrement** : opération sur les entrées qui produit la sortie
  - addition : 3 + 5 = 8
  - chiffrement : **Mo + K = Mc**

Chiffrement de César

- Rotation de chaque lettre du message original en fonction de la clé
- Exemple de clé : 3
  - A ⇒ D ; B ⇒ E ; C ⇒ F ; ...

Chiffrement de César - Exemple

LEA AIME ALLAN

OHD DLPH DOODQ

Chiffrement de Vigenère

- Plus élaboré :
  - la clé est un mot plutôt qu’un simple nombre
  - chaque lettre représente une rotation (ex: B = 2)
  - à chaque lettre du Mo correspond une rotation dans la clé

Chiffrement de Vigenère - Exemple

Ici, clé = `TARTE` (répétée autant de fois que nécessaire)

```
Mo : LEA AIME ALLAN
clé  : TAR TETA RTETA
Mc : FFS UNGF SFQUO
```

Principe de Kerckhoff

- « L’adversaire connaît le système »
- Les algorithmes de cryptographie sont **publics**
  - y compris les plus élaborés, utilisés tous les jours dans le monde
- **La sécurité repose sur la force et la non-divulgation de la *clé***

Force des clés

- Les clés peuvent être de taille quelconque
- Toute clé est cassable par **force brute** *en théorie*
- MAIS : la force de la crypto augmente très vite avec la taille de la clé
- Ex. : clés de 128 bits vs. 40 bits
  - protection des milliards de fois supérieure
- Aujourd'hui : crypto considérée sécurisée avec **clés 256 bits**

Protection des clés

1. **Où est la clé ?**
2. **Comment est-elle protégée ?**
3. **Qui y a accès ?**

Un **serveur de clés** répond à ces trois questions.

Le challenge de la cryptographie

1. **Protéger les données stockées**
2. **Protéger les données en transit**
3. **Protéger les clés**

Échec sur l’un des trois ⇒ *Game Over*

Crypto-systèmes

- Assurer **confidentialité**
  - Algorithmes symétriques (DES, 3DES, IDEA, ARS, RC4, RC5)
- Assurer **authentification**
  - Algorithmes asymétriques (RSA, E1 Gamel, ECC)
- Assurer **intégrité**
  - Algorithmes de hachage (MD5, SHA, RIPEMD, HMAC)
- Assurer **non-répudiation**
  - Signatures numériques (algos asymétriques + hachage)

Techniques de chiffrement symétriques

- Substitution
  - XOR
  - Rotation
  - Substitution aléatoire
- Permutation
- Techniques hybrides des précédentes

XOR

- XOR = **OU EXCLUSIF**
- Opération binaire : `A XOR B` vaut 1 si A vaut 1 *ou bien* B vaut 1 (et pas les deux)
- Autrement dit : si `A == B` alors `A XOR B` vaut 0 (sinon 1)
- Algorithme de chiffrement/déchiffrement naturel :
  - `Mo XOR K = Mc`
  - `Mc XOR K = Mo`

Rotation

- Substitution caractère par caractère
- Rotation de l’alphabet de N caractères (N est la clé)
- On appelle ROT-N l’algorithme de clé N
  - `ABCDE` ⇒ `DEFGH` (ici N = 3 ROT-3)
  - `MESSAGE` ⇒ `UMAAOM` (ROT-8)
- Chiffrement de César = ROT-3
- Usenet utilise ROT-13

Substitution aléatoire

- Substitution caractère par caractère
- Chaque caractère est associé à un caractère aléatoire
  - `A ⇒ X ; B ⇒ T ; C ⇒ M ; D ⇒ P ;` …
  - `BABA ⇒ TXTX`

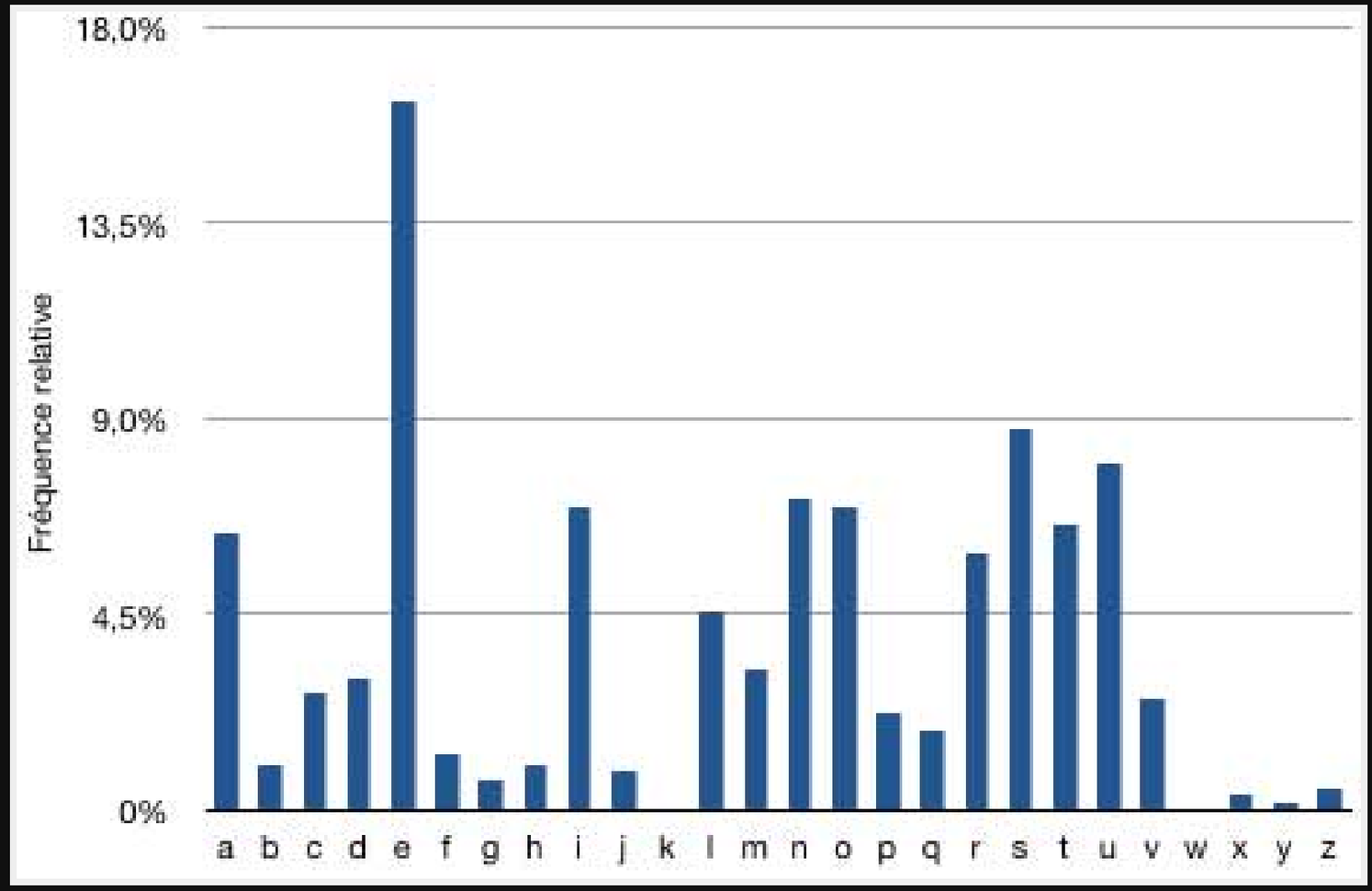
Permutation

- On garde les mêmes lettres exactement, mais on change leur position dans le message
  - `MESSAGE ⇒ ASGMESE`

Analyse de fréquences de caractères

Individuellement, toutes ces techniques (XOR, Rotation, Substitution aléatoire, permutation) sont facilement cassables par **analyse de fréquences de caractères**.

Analyse de fréquences de caractères



Techniques hybrides

- Les bons algorithmes de **chiffrement symétrique** utilisent simplement un panaché des techniques précédentes, en les répétant un grand nombre de fois
- Comme chacune de ces étapes est réversible, on peut déchiffrer en appliquant exactement la même séquence *en sens inverse*
- Ce sont ces **techniques hybrides** qui permettent de rendre le message chiffré très difficile à analyser, si la clé est suffisamment forte

Retour sur les catégories de crypto-systèmes

- Algorithmes symétriques
  - 1 clé (clé secrète)
- Algorithmes asymétriques
  - 2 clés (clé publique, clé privée)
- Algorithmes de hachage
  - transformation en sens unique (non-réversible)
  - données (taille quelconque) ⇒ *hash* (petite taille fixe)

Cryptographie symétrique

Clé unique

- **Une seule clé** : la clé secrète
  - utilisée pour **chiffrer** le message original
  - et pour **déchiffrer** le message chiffré
- (Très) **rapide**
- Utilisée pour assurer la **confidentialité**
- (Gros) problème : gestion de la clé entre Alice et Bob
  - Requiert un **canal de distribution sécurisé de la clé**

Sécurisation de l’échange de la clé

- Trois possibilités :
  - **clé pré-partagée** (échange **en présence**, par exemple pour configuration VPN site-à-site)
  - utilisation du **chiffrement asymétrique** juste pour l’échange de clé
  - échange de clé **Diffie-Hellman** (non étudié ici)
- Pourquoi ne pas utiliser ces canaux sécurisés pour faire transiter directement le message ?
  - ⇒ pas pratique, lent, inefficace

Cryptographie asymétrique

Deux clés au lieu d’une seule

- **Paire de clés** pour une entité :
  - **clé publique** accessible à tout le monde (**Kpe**)
  - **clé privée** connue uniquement de l’entité (**Kve**)

Principe

Tout ce qui est chiffré avec l'une des deux clés ne peut être déchiffré qu'avec l'autre clé

Applications

- (Très) **lente**
- Donc impraticable pour les larges volumes de données
- Deux applications principales :
  1. Canal sécurisé pour l'**échange de clé symétrique**
  2. **Signatures numériques** (authentification/non-répudiation)

Application 1 - Échange de clé de chiffrement

- Alice veut partager sa clé symétrique **`Ks`** avec Bob
- Alice chiffre la clé **`Ks`** avec la clé publique **`Kpb`** de Bob
- Elle obtient le message chiffré **`Mc`**
- Bob reçoit **`Mc`** et le déchiffre avec sa clé privée **`Kvb`** (et récupère **`Ks`**)
- **Bob peut maintenant échanger avec Alice par cryptographie symétrique en utilisant la clé unique et commune `Ks`**
- Technique utilisée aussi par TLS (SSH, HTTPS...), email, chiffrement de disque dur...

Application 2 - Authentification

- Alice chiffre **Mo** avec sa clé privée **Kva** (obtient **Mc**)
- Alice envoie **Mc** à Bob
- Bob utilise la clé publique **Kpa** d'Alice pour déchiffrer **Mc**
- Si déchiffrement OK, on **sait** que le message vient bien d'Alice (seule à avoir pu chiffrer ce message avec **Kva**)
  - *(pour peu que l'on ait un moyen de certifier que Kpa est la vraie clé publique d'Alice)*

Authentification - Problème

- **Rien n’empêche ici Eve de récupérer le message et de le déchiffrer !**
  - On peut juste prouver que le message *vient bien d’Alice*
- Cette technique, seule, n’est pas là pour assurer la confidentialité
  - Mais couplée avec le hachage, elle est la base de la **signature numérique**

Crypto asym - Points cruciaux

1. **Canal sécurisé** (Alice ⇒ Bob)
   - Alice chiffre avec **`Kpb`**
   - Bob déchiffre avec **`Kvb`**
2. **Authentification** (Alice est-elle bien émettrice ?)
   - Alice chiffre avec **`Kva`**
   - Bob déchiffre avec **`Kpa`**

Hachage (*hashing*)

Le hachage n’est pas du chiffrement !

- Le hachage d’un fichier permet d’obtenir une **empreinte** de celui-ci
  - ⇒ l’empreinte peut être recalculée et certifie ainsi qu’il s’agit exactement du même fichier non altéré

Principe du hachage

- **Pas de clé du tout** (pas du chiffrement)
- Transformation en sens unique ⇒ **non-réversible**
- Message (taille quelconque) passe par une *fonction de hachage*
- On obtient le **condensat** (empreinte, *hash*) du message (petite taille fixe)
- Utilisation principale : **intégrité** (si le ***hash* est identique** au départ et à l'arrivée, le **message n'a pas été modifié**)

Taille du hash et algorithmes

- Comme pour la taille des clés, la taille du *hash* résultant est important pour la sécurité du hachage
- Contrairement au chiffrement, la taille du message va en général être beaucoup plus grande que le *hash*
- Cela induit forcément la présence de **collisions**
- **Plus la taille du *hash* est grande, moins on a de collisions**
- Algorithmes : MD2, MD4, MD5, RIPEMD, SHA-1, SHA-2, HMAC

Collisions

- Si deux messages différents produisent le **même *hash*** ⇒ **collision**
- **Inévitable** par définition du hachage
- MAIS **risque acceptable** (pas grave si c’est une coïncidence)
- CAR si la fonction de hachage est bonne, elle garantit deux choses qui rendent le concept sécurisé

Garanties d’un bon hachage

1. **Deux messages proches ne seront pas en collision**
   - la moindre modification du message entraînera un condensat complètement différent
2. Il est **impossible de prédire**, par force brute, quand une **collision** va arriver
   - on ne pourra pas forger un message qui produit le même condensat

Intégrité

- Alice hache **`Mo`**, elle obtient un condensat **`H`**
- Alice envoie à Bob **`Mo + H`**
- Bob hache le message reçu et obtient **`Hb`**
- Bob compare **`Hb`** au condensat reçu
- Si les condensats correspondent, l'intégrité de **`Mo`** a pu être vérifiée ?
- NON : Mallory peut intercepter **`Mo + H`** et les remplacer par **`Mm + Hm`** qu'elle aura elle-même construits

Solution : signature numérique

- **Signature numérique** : permet de garantir la provenance d’un message
- Garantit donc l'**authenticité** et la **non-répudiation**
- Basée sur **chiffrement asymétrique + hachage**
- Le message est « signé » en chiffrant son condensat avec la clé privée de l’émetteur

Signature numérique

- Alice hache **`Mo`**, elle obtient son condensat **`H`**
- Alice chiffre **`H`** avec **`Kva`**, elle obtient **`sign`**
- Alice envoie à Bob **`Mo + sign`**
- Bob hache le message reçu et obtient **`Hb1`**
- Bob déchiffre **`sign`** avec **`Kpa`** et obtient **`Hb2`**
- Bob compare **`Hb1`** et **`Hb2`**
- Cette fois, si les deux *hash* correspondent, Bob est **certain que le message vient bien d'Alice**

Cryptographie - Résumé

- Assurer **confidentialité**
  - Algorithmes symétriques (DES, 3DES, AES, IDEA, ARS, RC4, RC5)
- Assurer **authentification**
  - Algorithmes asymétriques (RSA, E1 Gamel, ECC)
- Assurer **intégrité**
  - Algorithmes de hachage (MD5, SHA, RIPEMD, HMAC)
- Assurer **non-répudiation**
  - Signatures numériques (algos asymétriques + hachage)

Cryptographie (2)

Algorithmes et déploiement

Au programme

- Retour sur les trois crypto-systèmes d’un point de vue des algos existants
  - Sûr ou pas sûr ? Que faut-il utiliser en entreprise ?
- Techniques d’attaque de la cryptographie

Cycle de vie d’un algo crypto

- Naissance
- 10 ans avant d’être considéré
- encore 5 ans avant d’être utilisé
- puis 25-30 ans d’utilisation effective
- et finalement un nouvel algorithme prend le pas

DES (symétrique)

- *Data Encryption Standard*, 1975
- Symétrique, clé 56 bits
- Largement utilisé, devient un standard
- **Considéré non sécurisé aujourd'hui** (clé trop petite)
  - 1975 : 10 ans pour casser DES avec un ordinateur
  - 2024 : instantané

2DES / 3DES (symétrique)

- Chiffrement multiple DES : plusieurs passes augmentent l'efficacité (pas le cas pour tous les algos)
- 2DES : pas sécurisé (attaque *Meet in the Middle*)
- **3DES (Triple DES)**
  - insensible à l'attaque *Meet in the Middle*
  - `56x3 = 168` bits (OK à l'époque en attendant AES)
  - aujourd'hui, encore très utilisé (phénomène de *legacy code*) mais **considéré non sécurisé**

AES (symétrique)

- *Advanced Encryption Standard* (**Rijndael**), 2001
- Vainqueur d’une compétition lancée fin 1990s
- Conçu pour taille de clé adaptable (128, 192, 256)
- **Aujourd’hui : AES-256 considéré comme très sécurisé**

AES - Algorithme

```
AES(byte in[4*Nb], byte out [4*Nb], word w[Nb*(Nr+1)])
begin
  byte state[4,Nb]
  state = in
  AddRoundKey(state, w)
  for round = 1 step 1 to Nr-1
    SubBytes(state)
    ShiftRows(state)
    MixColumns(state)
    AddRoundKey(state, w+round*Nb)
  end for
  SubBytes(state)
  ShiftRows(state)
  AddRoundKey(state, w+Nr*Nb)
  out = state
end
```

RSA (asymétrique)

- *Rivest Shamir Adleman*, 1977
- Standard pour le chiffrement asymétrique
- Non inquiété depuis sa création, et donc **considéré très sûr**
- Consommateur de ressources, ne s’adapte pas idéalement aux besoins d’aujourd’hui (informatique mobile, objets connectés…)

ECC (asymétrique)

- *Elliptic Curve Cryptography*, divers algos depuis 2004
- Concurrents de RSA, beaucoup moins gourmands
- L'adoption n'est cependant pas si rapide :
  - brevets actifs sur les algos de cette famille (⇒ frilosité, coût)
  - équivalent à RSA en terme de sécurité pure (⇒ pas de besoin *absolu* de remplacer RSA rapidement)

Rappels sur la complexité des algorithmes

- Algorithmes symétriques : rapides car correspondent à des problèmes faciles (permutations, substitutions...)
- Algorithmes asymétriques : lents car correspondent à des problèmes difficiles
  - factorisation en nombres premiers (RSA)
  - problème du logarithme discret (El Gamal)
  - Calcul de courbes elliptiques (ECC) (plus rapide cependant)

Chiffrement : équivalence de tailles des clés

sécurité	sym	asym (RSA/DH)	asym (ECC)
80		1024	160
112	3DES	2048	224
128	AES-128	3072	256
192	AES-192	7680	384
256	AES-256	15360	512

MD5 (hachage)

- *Message Digest 5*, 1991, 128 bits
- **Non sécurisé**
  - on a pu montrer que, pour certains messages donnés, on pouvait construire des collisions par calcul
  - on considère donc qu'on pourrait éventuellement réussir à le faire pour n'importe quel message
- **À éviter aujourd'hui** pour les contrôles d'intégrité importants

SHA-1 (hachage)

- *Secure Hash Algorithm*, 1995, 160 bits
- **Non sécurisé**
  - comme pour MD5, considéré cassable depuis 2005
  - 2019 : nouvelles avancées qui fragilisent encore l'algo
- **À éviter aujourd'hui** pour les contrôles d'intégrité importants

SHA-2 / SHA-3 (hachage)

- **SHA-2** : 2001
  - SHA-256 / SHA-384 / SHA-512
  - **Toujours considéré comme sûr**, mais les avancées de 2019 sur SHA-1 sont inquiétantes pour SHA-2 (basé sur le même principe)
- **SHA-3** : 2012, basé sur une toute autre technique
  - **Considéré comme sûr**, mais beaucoup moins utilisé aujourd'hui que SHA-2

Stéganographie

L’art de la dissimulation

Principe

- **Dissimulation de données**
- Le message confidentiel est « caché » dans un message quelconque (non confidentiel)
- Ex. : image, document Word, musique…

Application

- La **cryptographie** garantit la confidentialité *mais pas le secret*
  - les attaquants qui interceptent un message chiffré savent pertinemment qu'il est chiffré
  - **on sait que le message original est là, mais très difficile à lire**
- La **stéganographie** met l'accent sur le secret plutôt que sur la confidentialité forte
  - principe : *personne ne sait* qu'un message se cache dans le document, à part les entités concernées
  - le message est à la fois invisible et à la vue de tous

Association Crypto + Stégo

- On souhaite avoir la confidentialité *et* le secret ?
  - ⇒ on peut associer les deux techniques
- On va alors :
  - chiffrer le message (crypto)
  - et dissimuler ce message chiffré, par exemple dans une image (stégo)

Techniques de stéganographie

1. **Injection** : on *ajoute* au fichier
   - ex. : commentaires GIF, document Word
   - augmente la taille du fichier (suspect, mais pas de limite de taille)
2. **Substitution** : on *remplace* des parties peu significatives du document par les données à dissimuler
   - ex. : bits peu significatifs des pixels d'un image
   - ne change pas la taille du fichier (quantité limitée de données dissimulées)
3. **Génération** : l'algorithme *génère* un fichier à partir des données
   - taille quelconque
   - problème : peut paraître suspect

Stéganographie - Détection

- De (très) nombreuses méthodes de stégo existent
- Donc pas de méthode universelle pour détecter l'utilisation de la stégo
- En général on sait parce qu'on a aussi trouvé le fichier original ou le programme utilisé
- Si on soupçonne l'utilisation de stégo et qu'on a affaire à quelques fichiers, on peut trouver
- Si on soupçonne l'utilisation de stégo et qu'on a affaire à de grosses quantités de fichiers potentiels, ça devient très difficile