



POLYTECHNIQUE
MONTRÉAL

UNIVERSITÉ
D'INGÉNIERIE

INF4420a: Sécurité Informatique

Cryptographie III



Aperçu – Crypto III

- Cryptographie à clé publique
 - Algorithme RSA
 - Algorithme de El-Gamal
 - Chiffrement à courbe elliptique (ECC)
 - Algorithmes post-quantiques
- Autres primitives cryptographiques
 - Hachage cryptographique
 - Signature numérique et infrastructures à clé publique (PKI)
 - Échange de clés
 - Diffie-Hellman
 - Cryptographie quantique
- Méthodes cryptanalytiques (suite)
 - Type d'attaques
 - Attaques quantiques
 - Complexité
- Principes d'utilisation de la cryptographie
 - Gestion de clés (privés et publiques)
 - Risques résiduels liés à l'utilisation de la crypto

RSA : préliminaires

- Deux clefs:
 - clef publique : une paire (e, N)
 - clef privée : une paire (d, N)
- Première étape: choisir N
 - Choisir n , la taille de N en fonction de sécurité requise
 - Selon le niveau de sécurité désiré
 - Tailles typiques $n = (1024), 2048, 3072, 4096, 8192$
 - Trouver deux nombres premiers p et q de taille $n/2$
 - Calculer $N = p * q$
- Choisir un entier e de taille n
 - relativement premier à $\varphi(N) = (p-1)*(q-1)$ i.e. $\text{pgcd}(e, \varphi(N)) = 1$
- Choisir d tel que
 - $e * d \equiv 1 \pmod{(p-1)*(q-1)}$
- À la fin, on n'a plus besoin de conserver p et q



RSA : Chiffrement et déchiffrement

- Alphabet de codage
 - $T = \mathbb{Z}_N^*$
 - généralement représentés par des chaînes de n bits
- Fonction de chiffrement
 - $x \in \mathbb{Z}_N^* \rightarrow y = E_e(x) = x^e \bmod N$
- Fonction de déchiffrement
 - $y \in \mathbb{Z}_N^* \rightarrow x' = D_d(y) = y^d \bmod N$
- Est-ce que $x' = x$?
 - $D_d(E_e(x)) = (x^e)^d \bmod N = x^{ed} \bmod N = x \bmod N$
(parce que $e^*d = 1 \bmod \varphi(N)$ et théorème de Euler)



L'algorithme RSA : exemple

- On utilise des valeurs petites pour illustrer
 - Soit $p = 11$ et $q = 13$, alors
 - $N = p * q = 143$ et $\phi(N) = (p-1) * (q-1) = 120$
 - e doit être relativement premier à $(p-1) * (q-1)$:
 - Exemple 1 : soit $e = 11$
 - l'inverse de 11 mod 120 est aussi 11: $11 * 11 = 121 = 1 * 120 + 1$ (pas très astucieux ! Cas atypique ...)
 - Pour chiffrer un mot de code $x = 7 \rightarrow y = 7^{11} \text{ mod } 143 = 106$
 - Pour déchiffrer $y = 106 \rightarrow x = 106^{11} \text{ mod } 143 = 7$
 - La clef publique est $N = 143$, $e = 11$
 - Exemple 2 : soit $e = 17$
 - Alors $d = 113$, parce que $17 * 113 = 1921 = 16 * 120 + 1$
 - Pour chiffrer un mot de code $x = 7 \rightarrow y = 7^{17} \text{ mod } 143 = 50$
 - Pour déchiffrer $y = 50 \rightarrow x = 50^{113} \text{ mod } 143 = 7$
 - La clef publique est $N = 143$, $e = 17$



L'algorithme RSA : exemple

- Avec des petites valeurs, on peut retrouver d
 1. En factorisant N
 - 143 ne peut pas être autre chose que $11 * 13$
 - on peut le retrouver en 11 essais
 2. En calculant $\phi(N)$
 3. En retrouvant la clé privé d
 - en résolvant équation $e * d = 1 \text{ mod } \phi(N)$, avec l'algorithme d'Euclide



Rappels arithmétiques (en nombres entiers)

- Plus grand commun diviseur (pgcd)
 - soit deux nombres a et b ; le plus grand entier qui divise a et b sans reste est le pgcd de ces deux nombres
- Algorithme d'Euclides
 - Calcul du pgcd
 - Pour $\text{pgcd}(3615807, 2763323)$
 $3,615,807 = 1 * 2,763,323 + 852,484 \quad a = m * b + r$
 $a \leq b; \quad b \leq r$
 $2,763,323 = 3 * 852,484 + 205,871$
 $852,484 = 4 * 205,871 + 29,000$
 $205,871 = 7 * 29,000 + 2,871$
 $29,000 = 10 * 2,871 + 290$
 $2,871 = 9 * 290 + 261$
 $290 = 1 * 261 + 29$
 $261 = 9 * 29 + 0$
• $\text{pgcd}(3615807, 2763323) = 29$
 - Permet également de calculer les inverses multiplicatifs dans \mathbb{Z}_N^*
 - Complexité = $O(n^3)$

Exponentiation rapide

Permet de calculer $x^e \bmod N$ en temps $O(n^3)$



Comment trouver un nombre premier très grand

- L'algorithme standard - Crible d'Ératosthène
 - On examine tous les nombres en éliminant ceux qui sont un produit de deux facteurs différents de 1
- Test de primalité probabiliste
 - Tout nombre premier doit satisfaire deux conditions:
 - $\text{pgcd}(p, r) = 1$ et $J(r, p) \equiv r^{(p-1)/2} \pmod{p}$
 - r est un nombre plus petit que p
 - J est la fonction de Jacobi
 - | $1 \text{ si } r = 1$
 $J(r,p)= J(r/2)*(-1)^{(p*p-1)/8} \text{ si } r \text{ est pair}$
 $J(p \bmod r, r) (-1)^{(r-1)*(p-1)/4} \text{ si } r \text{ est impair et } \neq 1$
 - si c'est satisfait, la probabilité que p soit premier est 50%
 - si on répète le test k fois avec succès, la probabilité est $1-2^{-k}$



Problème du log discret

- Propriétés mathématiques de Z_p
 - Tous les éléments de Z_p ont des inverses multiplicatifs, sauf 0
 - Donc, $Z_p^* = Z_p - \{0\}$
 - Il existe des éléments g dit *générateur* ou *racine primitive* tel que :
 $Z_p^* = \{g^0, g^1, \dots, g^{p-1}\}$
 - Notes :
 - Il est possible de vérifier en temps polynomial si un élément g est un générateur.
 - Il existe un très grand nombre de générateurs dans Z_p^*
 - Définition : Le logarithme discret en base g de $a \in Z_p$ est l'entier x tel que $a = g^x \bmod p$
 - Hypothèse calculatoire : Il n'est pas possible de calculer le log discret en temps polynomial sans connaître la factorisation de $p-1$.



Algorithme de El-Gamal

- Génération de clé
 1. Trouver un grand entier premier p tel que $p-1$ a au moins un grand facteur premier (donc difficile à trouver)
 2. Choisir au hasard un générateur g de Z_p^* et un entier d
 3. Calculer la valeur $e = g^d \text{ mod } p$
 4. Clé publique = (p, g, e) , Clé privé = d
- Chiffrement/Déchiffrement – pour un message $x \in Z_p^*$
 - Choisir un entier $k \in [0..p-1]$ au hasard
 - $E(k, x) = (y_1, y_2) = (g^k \text{ mod } p, xe^k \text{ mod } p)$
 - $D(y_1, y_2) = y_2 / y_1^d \text{ mod } p$



Algorithme de El-Gamal

- Intuition
 - Le message x est "masqué" dans y_2 en le multipliant par e^k par
 - La partie y_1
fournit à qui connaît d , l'information nécessaire pour reconstruire x en "divisant" par y_1^d (en réalité, calculer son inverse et multiplier)



Algorithme de El-Gamal – Notes importantes

- Méthode de chiffrement dite "probabiliste"
 - il n'existe pas de chiffrement unique pour un même x .
- Choix de k
 - Ce n'est pas une clé *strictu sensu*
 - Il n'est pas nécessaire que Bob connaisse k en avance pour déchiffrer x
 - C'est plutôt un *masque*
 - Qui ne doit pas être dévoilée (!)
 - Très important de choisir un k différent à chaque fois
 - Un mauvais choix de k (toujours pareil, basse entropie) rend possible des attaques à texte clair choisi (dangereuses si entropie après codage basse)

Notion de groupe - rappel

- Notion de groupe $(G, *)$
 - Un ensemble abstrait G sur lequel on a défini une opération abstraite " $*$ " avec certaines propriétés :
 - Il existe un élément identité 1 : $a * 1 = a$
 - Associativité : $a * (b * c) = (a * b) * c$
 - Tout éléments à un inverse : $a * a^{-1} = 1$
 - (Commutatif): $a * b = b * a$, on dit que le groupe est "abélien" ou "commutatif"
 - Définition (exponentiation):
 - $a^n = a * a * \dots * a$, n fois où n est un entier et $(G, *)$ est un groupe abélien
 - Note: On peut définir le problème de log discret sur n'importe quel groupe
- Exemples
 - Z_p^* , Corps fini (corps de Galois), Courbe elliptique

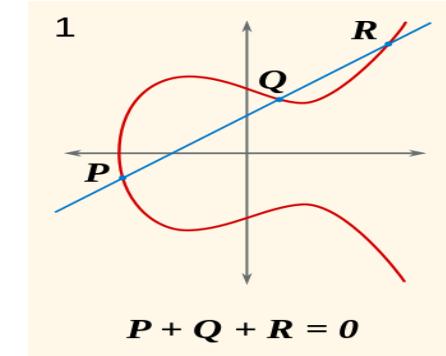
El-Gamal sur Corps de Galois

- $\text{GF}(2^n)$
 - Corps de Galois (corps fini) de taille 2^n
 - Se base sur l'arithmétique modulaire avec des polynômes de degré n
 - Toutes les coefficients des polynômes sont binaires
 - toute l'arithmétique est binaire
- El-Gamal sur $\text{GF}(2^n)$
 - Chiffrement et déchiffrement très efficaces
 - Surtout utilisée sur des plateformes matériel
 - Peu utilisé aujourd'hui



Cryptographie à courbes elliptiques (ECC)

- Courbe elliptique
 - Se base sur l'opération de "somme" de points sur une courbe elliptique
 - Combiné avec des opérations sur des corps de Galois
- ECC
 - Permet un niveau équivalent de sécurité avec des tailles de clés entre 6-12 fois plus petites (p.ex. 256 bit EC \cong 3072 bit RSA)
 - ➔ meilleure performance de chiffrement et déchiffrement
 - ➔ taille réduite des signatures numériques pour applications avec peu de bande passante





ECC - suite

- Standard NIST
 - ECDH (échange de clés)
 - ECDSA (signature numérique)
 - Pas utilisé pour le chiffrement (clé symétrique)
- Paramètres recommandés
 - Taille de clé 256 (SECRET) ou 384 bit (TOP SECRET)
 - Courbes elliptiques
 - En principe, n'importe quelle courbe peut être utilisée
 - Certaines courbes seraient moins “robustes” que d'autres
 - Standard NIST
 - Le choix de courbes possible est limité
 - “Théorie de la conspiration”
 - » Les courbes auraient été choisies par la NSA/NIST avec des “trappes” permettant de déchiffrer sans connaître la clé (à la S-box dans DES....)



- Caractéristiques
 - Une généralisation de l'algorithme de Shor (Mosca, Ekert 1998) permet de retrouver un groupe “caché” sous-jacent (“hidden subgroup”) en temps polynomial quantique
 - ➔ Imposse une restriction sur les algorithmes post-quantiques
 - ➔ Élimine d'emblée plusieurs algorithmes à clé publique
- Type (selon l'origine)
 - Algorithmes « pré-quantique »
 - Anciens algorithmes à clé publique (années 80)
 - Avaient été mis de côté car moins pratiques
 - Maintenant sorti de l'armoire et « recyclés » (p.ex. McEliece)
 - Nouveaux algorithmes
 - Invention plus récente (années 90 et 2000)
 - Souvent (mais pas tout le temps) motivée par la menace quantique (p.ex. basés sur les réseaux)



- Types (selon le problème calculatoire sous-jacent)
 - Basé sur les réseaux euclidiens (« lattice-based »)
 - Basé sur les polynômes multivariés
 - Basé sur codes correcteur d'erreurs
 - Basé sur les fonctions de hachage....
- NIST Post-Quantum Project
 - Concours lancé en 2016 pour trouver un standard de PQC
 - Round 1 – 60+ candidats proposés
 - Round 3 (juil 2020) –
 - Chiffrement : 4 finalistes choisis
 - McEliece, CRYSTALS-KYBER, NTRU, SABER
 - Signature numérique : 3 finalistes choisis
 - CRYSTALS-DILITHIUM, FALCON, Rainbow
- Tests en utilisation réelle
 - Test de performance (2016) réalisé par Google sur Google Chrome
 - Implémentait TLS avec la proposition de PQC CECPQ1, développé par Google
 - Résultats encourageants

Cryptographie post-quantique (PQC)

- Problème principaux
 - Algorithmes récents ou peu utilisés
 - ➔ Peu de scrutin de la résistance aux attaques classiques ou quantiques
 - Taille de clés plus élevées
 - Certificats et signatures plus grands
 - Peu pratiques pour des applications avec bande passante limitée

Algorithme	Type	Clé publique
<i>256-bit Elliptic Curve</i>	Classique	32 B
<i>3072-bit Discrete Log</i>	Classique	384 B
<i>SIDH</i>	Isogénie	751 B
<i>Streamlined NTRU Prime</i>	Réseau	1232 B
<i>Quasi-cyclic MDPC-based McEliece</i>	Code correcteur	1232 B
<i>New Hope</i>	Réseau (RLWE)	2 KB
<i>NTRU Encrypt</i>	Réseau	6130 B
<i>Random Linear Code based encryption</i>	RLCE	115 KB
<i>Rainbow</i>	Polynôme multivar.	124 KB
<i>Goppa-based McEliece</i>	Code correcteur	1 MB



Hachage cryptographique

- Objectif : Intégrité
 - S'assurer qu'un message n'a pas été modifié de façon non autorisée une fois qu'il a été terminé par son auteur légitime
- Fonctions de hachage cryptographique h
 - Une fonction $h()$ est dite de hachage cryptographique si à partir d'un message x elle produit un *haché* (ou *hachage*) $h(x)$, avec ces propriétés
 1. **(à sens unique)** : il est très difficile de trouver un x à partir d'un h donné, tel que $h = h(x)$
 2. **(absence de collision)** : il est très difficile de trouver deux messages x et y , tel que $h(x) = h(y)$
 3. **(effet d'avalanche)** : il est très difficile de trouver à partir d'un x donné de trouver un autre x' « similaire » tel que $h(x) = h(x')$
 - Notes
 - 3 implique 2 (trivial), 3 implique 1 (pas trivial)
 - En anglais, $h(x)$ est appelé "hash", MAC (pour *Message Authentication Code*), "message digest" ou simplement "digest"
 - Ne pas confondre avec les fonctions de hachage universelles, utilisées par exemple dans la construction de compilateur, les structures de données et algorithmes aléatoires, etc.



- Fonctions obsolètes
 - MD4 (128 bit)
 - Conçu par Rivest (de RSA)
 - Ressemble un peu à DES
 - Plusieurs rondes de coupage, transposition, permutation, et autre opérations binaires.
 - MD5 (128 bit)
 - Version amélioré de MD4
 - Usage très répandu
 - Utilisé par le programme linux md5sum
 - **Collisions en quelques heures!**
 - SHA-1 (160 bit)
 - Conçue par la NSA
 - Collisions possibles
 - Remplacé officiellement depuis 2011
- Fonctions recommandées
 - SHA-2
 - Famille de 5 fonctions introduites en 2001
 - Conçue par la NSA
 - Taille de haché : 224, 256, 384, 512
 - Similaire en structure à MD5 et SHA1
 - Aucune vulnérabilité connue (2020)
 - SHA-3
 - Compétition du NIST en 2006
 - Besoin d'avoir une famille de fonction différente de MD5/SHA1/SHA2 (au cas où...)
 - Algorithme KECCAK choisi (Bertoni, Daemen, Peeters, van Aasche)
 - Taille de haché : 224, 256, 384, 512



Intégrité de message avec haché (MAC)

- Modèle théorique
 - Modèle de Shannon révisé - version « intégrité »
 - Ève peut intercepter tout message et le lire
 - Ève peut modifier le message partiellement ou entièrement sans que Alice ou Bob s'en rende compte
 - Attaque *par personne interposée*
 - » En anglais « Man-in-the-middle » (MITM) ou « person-in-the-middle » (PITM)



- Objectif
 - Alice et Bob veulent s'assurer que toute modification d'un message original x soit détectable par Bob



Intégrité de message par MAC

- Protocole « canonique » (de base)
 1. Alice calcule le haché $h = h(x)$ de son message x
 - h est le « message authentication code » (ou MAC) pour x
 2. Alice transmet le message x par le canal de communication
 3. Alice transmet le haché $h(x)$ soit
 - a) En utilisant un canal de transmission alternatif qu'Ève ne peut pas modifier (« out-of-band » transmission)
 - b) Sur le canal principal, mais en utilisant une méthode d'authentification qu'Ève ne peut pas falsifier (p.ex. reconnaissance vocale, etc.)
 4. Bob reçoit un message x' (possiblement modifié)
 5. Bob calcule $h(x')$ et compare avec $h(x)$ reçu
 - ➔ Si $h(x') = h(x)$ alors avec très haute probabilité $x' = x$ et aucune manipulation n'a eu lieu

- Si Alice et Bob
 - Ne dispose pas d'un moyen d'authentification,
 - Ne dispose pas d'un canal alternatif sécurisé (non accessible à Ève),
 - Mais ils peuvent partager une clé secrète au préalable, alors
- Protocole MAC à clé partagée
 - 0. Alice et Bob partage une clé secrète K connue d'eux seulement
 - 1. Lorsqu'Alice veut transmettre x , elle calcule $h = h(K \parallel x)$
 - 2. Alice transmet x par le canal principal
 - 3. Alice transmet h par le même canal que x
 - 4. Bob calcule $h' = h(K \parallel x')$ à partir du message reçu x'
 - 5. Si $h = h'$ alors très probablement $x = x'$ et le message n'a pas été modifié
- Avantages
 - Ève ne peut pas « tromper » Alice et Bob sans connaître la clé K
 - Pas besoin d'un 2e canal sécurisé



- Protocole standardisé (standard HMAC)
 1. Alice calcule $\text{HMAC}(K, x)$ où
 - $\text{HMAC}(K, x) = h(K \oplus opad) \parallel h(K \oplus ipad) \parallel x$
 - K est la clé dérivée
 - $H(K)$, si K est plus grande que la taille du bloc (déterminée par la fonction de hachage)
 - K , dans le cas contraire
 - $opad$ est une constante de la taille du bloc répétant le byte $0x5C$
 - $ipad$ est une constante de la taille du bloc répétant le byte $0x36$
 2. Bob calcule et vérifie $\text{HMAC}(K, x') = \text{HMAC}(K, x)$
 - Proposé par Bellare, Cannaeti, Krawczyk en 1996
 - Adopté comme standard par le NIST en 2002
 - Plus sécuritaire que le protocole « simplifié » présenté antérieurement



Signature numérique

- Objectifs
 - Authenticité :
 - Pouvoir prouver qu'un document électronique a bel et bien composé et "signé" par son préputé auteur.
=> Il ne doit pas être possible pour personne de falsifier la signature d'autrui.
 - Intégrité :
 - Pouvoir prouver que le document n'a pas été modifié depuis qu'il a été signé par son auteur légitime.
=> Il ne doit pas être possible pour une autre personne que l'auteur de changer le document après sa signature sans violer la condition d'authenticité.
 - (Non-répudialité)
 - Empêcher qu'un auteur légitime puisse *a posteriori* nier qu'il est l'auteur et signataire d'un document qu'il a bel et bien signé
=> Il ne doit pas être possible de "répudier" une signature faite par soi-même



Signature numérique avec crypto à clé publique

- Signature
 - Pour signer un message x
 1. Ajouter au message un préambule T , p.ex.
"Le document qui suit a été signé par José M. Fernandez, en date du ..."
 $x' = T \parallel x$
 2. Utiliser la clé privé d pour produire la version signé y du document en utilisant la clé privé et l'algorithme de déchiffrement:
 $y = D(x', d)$ p.ex. $y = (x')^d \text{ mod } N$ avec RSA
 - Vérification
 - Pour vérifier un document y :
 1. Utiliser l'algorithme de chiffrement avec la clé publique e du présumé auteur pour obtenir $x' = E(y, e)$
 2. Vérifier si x' est bel et bien un message "légitime" (bien formaté, a un préambule, qui a du sens, etc.). Si oui, accepter la signature.
 - Notes
 - *Pourquoi un préambule?*
 - Parce qu'il est possible pour un malfaiteur de falsifier une signature sur un message aléatoire ("garbage"), mais il ne lui est pas possible de le faire sur un message déterminé de son choix (p.ex. ayant un préambule raisonnable en français)
 - *Authenticité de la clé publique ?*
 - Comment s'assurer que le vérificateur à la bonne clé publique e qui correspond vraiment à l'auteur ?



Signature numérique avec hachage

- Signature
 - Pour signer un message x
 1. Calculer le haché $h(x)$ du message avec une fonction de hachage cryptographique
 2. Utiliser la clé privé d pour $h(x)$ comme avant pour obtenir la signature
 - $\text{sig}(x) = D(h(x), d)$
 3. Le document signé contient : $(x, \text{sig}(x))$
- Vérification
 - Pour vérifier un document (y, s)
 1. Calculer le hachage $h(y)$ de y
 2. Obtenir la valeur h' en chiffrant la signature s avec la clé publique e ,
$$h' = E(s, e)$$
 3. Accepter la signature si $h' = h(y)$
- Avantages
 - Plus rapide
 - La signature est indépendante du message lui-même

- Objectifs
 - Alice et Bob n'ayant pas échanger de clés auparavant désirent établir un canal privé
- Conditions et préalable
 - Ils ont accès à un canal "public" (non sécurisé)
 - Ils peuvent s'authentifier mutuellement (p.ex. par la voix)
- Protocole de Diffie-Hellman
 - Se base sur la difficulté du log discret
 - Permet à Alice et Bob de générer une clé dans $[0..p-1]$ connue de personne d'autre
 - En l'absence d'authentification ...
 - ➔ Vulnérable aux attaques "man-in-the-middle"



Échange de clés - Diffie-Hellman (suite)



Alice



Bob

Choisi

- p premier
- g générateur de Z_p^*
- a aléatoire dans $[1..p]$

Calcule

- $K = (g^b)^a \text{ mod } p$

 $p, g, g^a \text{ mod } p$ $g^b \text{ mod } p$ Données x chiffrées par $E_K(x)$

Choisi

- b aléatoire dans $[1..p]$

Calcule

- $K = (g^a)^b \text{ mod } p$

Cryptographie quantique

- Algorithme d'échange de clé
 - Proposé en 1984 par Brassard et Bennett
 - Principe de base
 - 1 bit codé sur un seul photon
- Avantages
 - Sécurité basée sur la mécanique quantique
 - Les propriétés quantiques d'un photon ne peuvent pas être reproduites parfaitement
 - Toute observation du photon pour extraire de l'information le détruit
 - ➔ Est « post-quantique » de façon démontrable
- Désavantages
 - Nécessite une infrastructure dédiée spéciale
 - Réseau de fibre optique
 - Liens satellite par laser
 - Cher à implémenter
- Situation actuelle
 - Course aux armements : crypto quantique par satellite





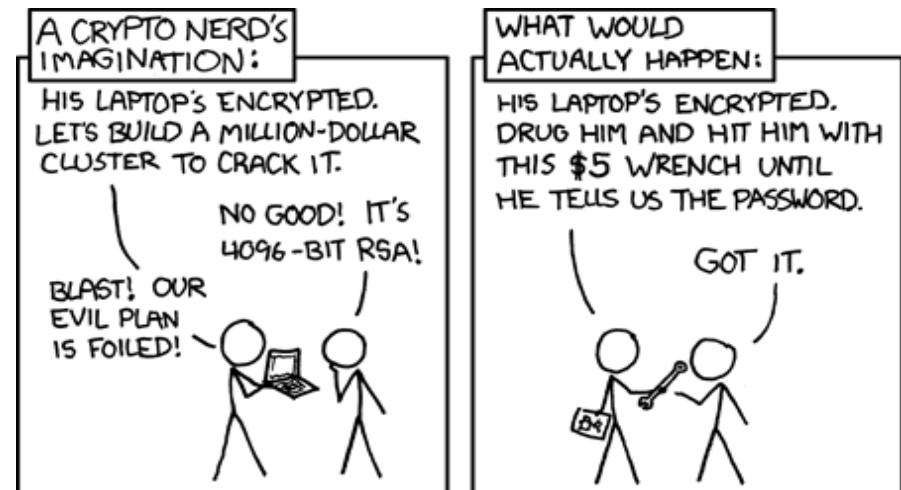
Attaques cryptographiques

- Types d'attaques
(en fonction des données disponibles pour l'attaquant) :
 - Texte chiffré seulement (*known ciphertext*)
 - uniquement accès au texte chiffré sans pouvoir chiffrer des messages (p.ex. document chiffré intercepté sur le réseau)
 - Texte connu (*known plaintext*)
 - accès au texte en clair en plus du texte chiffré (p.ex. interception d'un document chiffré qui sera diffusé publiquement plus tard comme un discours). On vise à trouver la clé pour déchiffrer les futurs messages
 - Texte choisi (*chosen plaintext*)
 - capacité de choisir le texte en clair en plus du texte chiffré (p.ex. on a accès à un clavier qui permet d'écrire des messages qu'on intercepte sous forme chiffré)
 - Texte chiffré choisi (*chosen ciphertext*)
 - Accès à un texte chiffré, et l'algorithme de chiffrement (p.ex. hachage de mot de passe sur un PC). On vise à chiffrer du texte connu et à obtenir le texte chiffré
- Typiquement, texte chiffré seulement



Attaques cryptographiques

- Plusieurs attaques possibles :
 - Attaque contre l'algorithme
 - Attaque contre l'implémentation
 - Attaque contre l'opération
- En pratique, on voit plus souvent les attaques contre les implémentations et les opérations
- Une mauvaise utilisation de la cryptographie peut introduire la possibilité d'un attaque contre l'algorithme



<http://xkcd.com/538/>



Attaques cryptographiques

- Force brute (*rappel*)
 - Attaque de type « texte chiffré seulement »
 - Procédure
 - On déchiffre le texte chiffré avec la clé n
 - On effectue un test sur le message déchiffré pour voir s'il s'agit d'un vrai message (ex. on calcule l'entropie du texte déchiffré, si elle est très faible, sûrement un message en clair)
 - Si le test indique qu'on a déchiffré le message, la clé recherchée est la clé n
 - Sinon, on recommence avec la clé $n+1$
 - En bref, on essaie toutes les possibilités de clés
 - Temps pour réaliser l'attaque dépend de la taille de clé et de la capacité de traitement
 - Lorsqu'on calcule le temps pour casser la clé, on calcule le pire cas (on essaie toutes les clés), si la distribution des clés est uniforme sur l'espace des clés, le temps moyen sera $t_{\max}/2$



Attaques cryptographiques

- Attaque dictionnaire
 - Attaque de type « texte clair choisi »
 - On construit un dictionnaire de symboles qui sont susceptibles d'être émis par la source
 - On essaie chacun des mots du dictionnaire jusqu'à ce qu'on trouve (ou non) une correspondance avec le mot chiffré
 - Selon Turing, on peut échanger de l'espace mémoire pour du temps et bâtir des tables de correspondance
 - Le calcul des tailles de tables de correspondance similaires au calcul de force brute
 - Efficace contre les chiffrement par bloc
 - Divers mécanismes pour diminuer la vulnérabilité aux dictionnaires
 - Grand espace de possibilité de message « uniformément couvert »
 - Codage avec compression et bourrage aléatoire
 - Utilisation de sel (salt)



Complexité de calcul des attaques

- Chiffrement symétrique
 - Chiffrement/déchiffrement: $O(n)$ ou $O(n^2)$
 - Force brute : $O(2^n)$
 - Attaque par dictionnaire :
 - $O(M) = O(2^{H(S)})$ où M est la taille du dictionnaire
 - Attaque quantique (algorithme de Grover) : $O(2^{n/2})$
 - Il faut « juste » doubler la taille de clé pour obtenir le même niveau de sécurité « post quantique »
- Chiffrement à clé publique
 - RSA/EI Gamal/ECC
 - Chiffrement/déchiffrement : $O(n^3)$
 - Force brute : $O(2^n)$
 - Attaque par dictionnaire : $O(2^{H(S)})$
 - Algorithmes factorisation/log discret « classique »: $O(2^{n/3})$
 - Attaque quantique (algorithmes de Shor): $O(n^3)$
- Normalement on doit intégrer la loi de Moore dans les calculs
 - Chaque 18 mois on double la puissance de calcul
(mais on arrive peut-être à la fin...)

Attaques cryptographiques

- Attaques sur l'implémentation souvent causées par la présence d'une des fautes mortelles en crypto :
 - Cryptographie propriétaire ou « maison »
 - Mauvais codage
 - Mauvaise gestion des clés en mémoire ou persistance des clés en mémoire (ex. attaque « cold boot »)
 - Mauvais génération de clé (ex. Debian OpenSSL)
 - Mauvaise source d'entropie dans le système
 - El-Gamal
 - Vecteur d'initialisation pour les algorithmes de flux
 - Challenge-response (à voir dans la section d'authentification)

Principe de gestion de clés

- Générations de clés
 - Nécessité de source de bit parfaitement aléatoire
 - Méthode matériel vs. logiciel vs. "manuel"
 - "Souveraineté" et contrôle sur la génération des clés
 - Difficulté technique pour certains algorithmes
 - RSA : p et q premier, etc.
 - El-Gamal : p t.q. $p-1$ a un grand facteur, etc.
- Gestion des clés et réduction de risque
 - Possibilité de révocation
 - Distribution au préalable
 - Contrôle positif (déttection de perte ou vol)
- Mécanisme de protection
 - Contrôle d'accès
 - Chiffrement des clés par mot de passe ou phrase de passe
- Principe de segmentation
 - Clés de réseaux vs. clés point-à-point
 - Durée de vie limitée des clés
- Distribution de clés
 - Nécessite de canaux privés dédiés
 - Distribution physique
 - Utilisation de KEK (key-encryption keys) ou équivalent

Réductions de risque et principes de bases (« cheatsheet »)

- Souveraineté de clé
 - Entropie
 - Contrôle d'accès
- Principe de Kerchoff
 - Pas de « sécurité par obscurité »
- Professionalisme
 - Algorithmes
 - Protocoles
 - Implémentations
- Gestion de clés
 - Segmentation
 - Temps,
 - Systèmes/réseaux,
 - Niveau de classification
 - compartiment “verticaux”
 - Mécanismes de confiance (PKI)
 - Hiérarchique
 - Décentralisé
 - Révocation