ECC - suite

- Standard NIST
  - ECDH (échange de clés)
  - ECDSA (signature numérique)
  - Pas utilisé pour le chiffrement (clé symmétrique)
- 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 connaitre la clé*
        (à la S-box dans DES….)

Cryptographie post-quantique (PQC)

- 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
    - ➔ Impose 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)

POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

Cryptographie post-quantique (PQC)

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

Exemples de fonctions de hachage cryptographique

- 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 authentification 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

Intégrité avec clé secrete partagée

- 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 $h = h(K \,||\, x)$
  - 2. Alice transmet x par le canal principal
  - 3. Alice transmet h par le *même canal* que x
  - 4. Bob calcule h' = $h(K \,||\, x')$ 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 connaitre la clé K
  - Pas besoin d'un 2e canal sécurisé

Protocole standardisé (HMAC)

- Protocole standardisé (standard HMAC)
  1. Alice calcule HMAC (K, x) où
     - HMAC (K,x) = h(K' ⊕ *opad*) || *h*(K'⊕ *ipad*) || x)

       et
     - 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 HMAC (K, x') = HMAC (K, x)
- Proposé par Bellare, Canneti, 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 prétendu auteur.

    => Il ne doit pas être possible pour personne de falsifier la signature d'autrui.
  - Intégrité :
    - Pouvoir prouver que le document n'as 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-répudiabilité)
    - 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 …"
       $x' = T \| x$
    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. $y = (x')^d \bmod N$ 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

Échange de clés – Diffie-Hellman

- 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 $[1..p]$

$p, g, g^a \bmod p$

Choisi
- b aléatoire dans $[1..p]$

$g^b \bmod p$

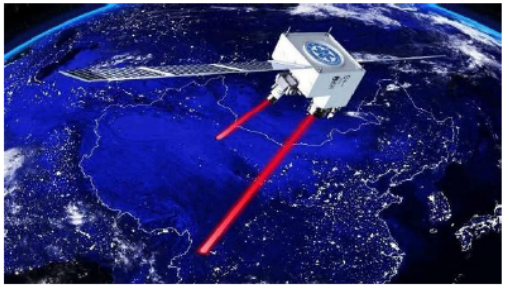
Calcule
- $K = (g^b)^a \bmod p$

Calcule
- $K = (g^a)^b \bmod p$

Données x chiffrées par $E_K(x)$

Cryptographie quantique

- Algorithme d'échange de clé
  - Proposé en 1984 par Brassard et Bennett
  - Principe de base
    - 1 bit codé sur un seul photon



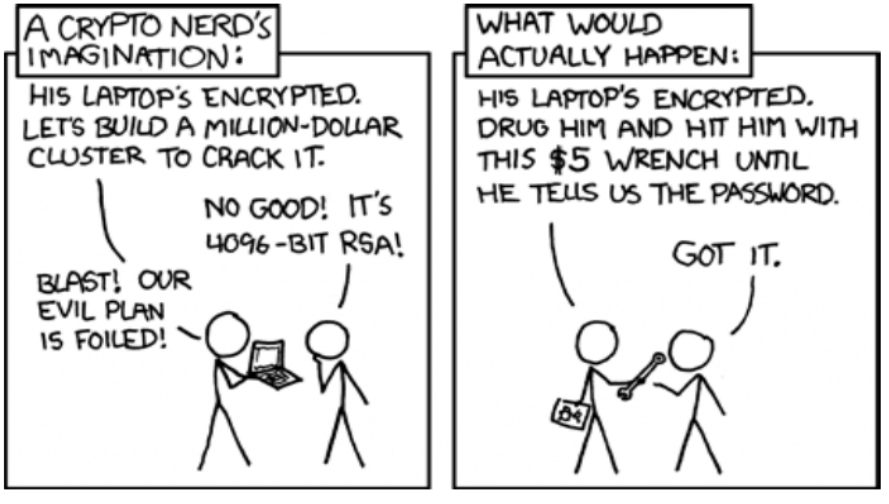
- Avantages
  - Sécurité basée sur la mécanique quantique
    - Les propriétés quantiques d'un photon ne peuvent pas être reproduites parfaitement
    - Toute observation du photon pour extraire de l'information le détruit
    - ➔ Est « post-quantique » de façon démontrable
- Désavantages
  - Nécessite 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 a 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 mauvais 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, surement 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 possibilité 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 :
    - $\mathbf{O(M) = O(2^{H(S)})}$ ou 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/El Gamal/ECC
    - Chiffrement/déchiffrement : $\mathbf{O(n^3)}$
    - Force brute : $\mathbf{O(2^n)}$
    - Attaque par dictionnaire : $\mathbf{O(2^{H(S)})}$
    - Algorithmes factorisation/log discret « classique »: $\mathbf{O(2^{n/3})}$
    - Attaque quantique (algorithmes de Shor): $\mathbf{O(n^3)}$
- 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…)