Protocoles de sécurité et PQC

Bruno MARTIN,
Université Côte d'Azur

Plan

Services et mécanismes de sécurité

- Services de sécurité implémentent la politique de sécurité.
  Certains services inutiles pour une politique donnée.
  Exemple : confidentialité ne requiert pas la signature
- Chaque service traite un ensemble particulier de menaces
  Exemple : confidentialité prémunit contre l'accès non-autorisé à l'information.
- Services de sécurité implémentés par les mécanismes de sécurité
  Certains services utilisent le même mécanisme
  Exemple : hachage utilisé en identification et signature.

Services de sécurité

Définis dans la norme ISO 7498-2 comme :

1	service d'authentification d'entités
	service d'authentification de l'origine des données
2	service de contrôle d'accès
3	service de confidentialité avec/sans connexion
	service de confidentialité sélectif
	service de confidentialité du flux de trafic
4	service d'intégrité de connexion avec/sans récupération
	service d'intégrité sélectif
	service d'intégrité sans connexion (sélectif ou non)
5	non répudiation avec preuve d'origine
	non répudiation avec preuve de dépot

Mécanismes de sécurité

Implémentent les services de sécurité

- chiffrement
- signatures numériques
- mécanismes de contrôle d'accès
- mécanismes d'integrité des données
- mécanismes d'authentification
- empaquetage pour le trafic
- contrôle de routage
- tiers de confiance (notariat électronique)

- gestionnaire de sécurité (gestion des clés)
- audit
- détection d'intrusion

Standards Internet et l'IETF

l'Internet Engineering Task Force (IETF) : groupe technique chargé de l'ingénierie et de l'évolution d'Internet. Propose des standards via des groupes de travail, p.e. en sécurité :

- Open specification for PGP
- Authenticated Firewall Traversal
- Common Authentication Technology
- Domain Name Security
- IP Security Protocol (IPSEC)
- One time password authentication
- Public Key Infrastructure
- S/MIME Mail Security
- Secure Shell
- Simple Public Key Infrastructure
- Transport Layer Security
- Web Transaction Security

Que produit l'IETF ?

- Des **RFC** (Request for Comments) proposent des standards pour l'évolution d'internet.
- Des documents de travail, Internet Drafts sans statut formel de durée de vie de 6 mois.

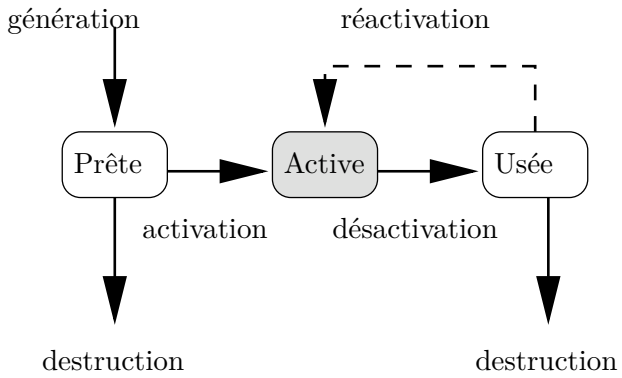
Les RFC recouvrent une grande variété de documents allant des documents informels jusqu'à la spécification complète de certains protocoles. Une fois publié comme RFC le document reçoit un numéro. Un document n'est ni revu ni ré-édité sous le même numéro. On les trouve à l'adresse `www.ietf.org/rfc.html`

Plan

Techniques de gestion de clés [2]

Reposent sur

- chiffrement
- utilisation des clés
- politique de sécurité



permettent d'éviter :

- la modification
- la destruction malintentionnée
- le rejeu
- la divulgation (disclosure)

Pendant toute la durée de vie des clés, il faut assurer :

la génération sûre	suppression	révocation	la certification
l'enregistrement	distribution	destruction	installation

Modèles pour l'établissement de clés

Procédé qui rend disponible une clé à une ou plusieurs entités.
Recouvre :

- transport de clés (publiques, secrètes)
- mise à jour des clés
- dérivation des clés
- mise en accord symétrique

Transport de clés

Besoin d'échanger des clés de façon sûre :

- évident dans le cas de la crypto à clé secrète
- contrer « MIM » pour la crypto à clé publique

Comment faire ?

- techniques cryptographiques :
  - chiffrement contre la divulgation et l'utilisation frauduleuse.
  - intégrité contre la modification abusive.
  - authentification contre la masquarade.

Premières solutions

1. Fixer un rendez-vous pour échanger les clés
2. Envoyer la clé par un courrier spécial
3. Utiliser une clé partagée pour chiffrer une nouvelle clé

Discussion

2 premiers cas : contact nécessaire pour échanger la clé. Pas toujours possible. P.e. dans un conflit.
Dans le dernier cas, on considère un réseau de communication entre N utilisateurs. Pour que chaque utilisateur puisse communiquer avec n'importe quel autre de façon sûre, il faut $\binom{N}{2} = O(N^2)$ clés partagées distribuées de manière sûre.

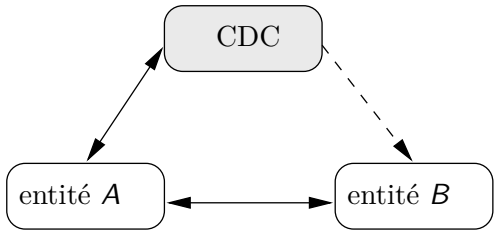
Etablissement de clés point à point

Mécanisme de base

- basé sur un échange symétrique : les deux entités partagent une clé secrète commune.
- basé sur un échange asymétrique : chaque entité a un couple (clé publique certifiée/clé privée) et la clé publique certifiée de l'autre partie
  - pour l'intégrité des données ou l'authentification de l'origine, le destinataire a besoin du certificat de la clé publique de l'expéditeur
  - pour la confidentialité, l'expéditeur a besoin du certificat de la clé publique du destinataire.

Etablissement de clés au sein d'un même domaine

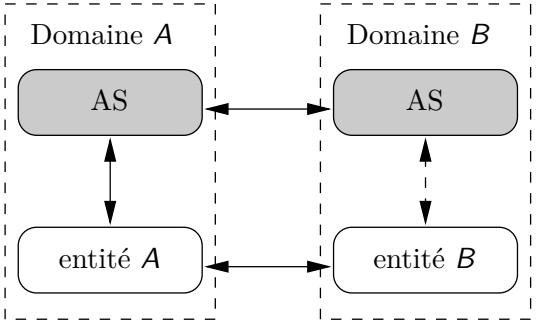
Solution possible : centre de distribution de clé (CDC) en qui tout le monde a confiance.
Chaque utilisateur partage une clé secrète avec le CDC. Quand A veut communiquer avec B, elle choisit une nouvelle clé de session et l'envoie (chiffrée) au CDC qui la retransmet à B chiffrée avec la clé commune entre le CDC et B.
Moyen convenable pour l'**échange au sein d'un même domaine**.



1. tout le monde a confiance en CDC !
2. Tous les utilisateurs partagent une clé avec le CDC
3. CDC a accès à tous les messages échangés

Etablissement de clés entre deux domaines

Hyp : $\exists$ CDC (ou **autorité de sécurité**) par domaine.



Si A et B ont une confiance mutuelle ou si chaque entité a confiance en l'AS de l'autre domaine, tout se passe comme s'il n'y en avait qu'un.
Deux réalisations : asymétriques ou symétriques.

Etablissement de clés entre deux domaines (asymétrique)

Si les entités n'ont pas de service de certification, chaque entité contacte sa propre AS pour obtenir le certificat de son correspondant. Les AS peuvent échanger les certificats des clés publiques des entités A et B et les redistribuer à A et B.
Autre approche : certification croisée, i.e. une AS certifie les clés publiques de l'autre.

Etablissement de clés entre deux domaines (symétrique)

1. Une des entités contacte son AS pour obtenir une clé de session.
2. Les deux AS établissent une clé secrète partagée utilisée par les deux entités.
3. clé distribuée par chaque AS qui utilise l'autre comme **centre de traduction de clé** [1] ou par l'intermédiaire d'une des entités qui sera responsable de la transmission vers la seconde entité.

Si les AS ne se font pas confiance, il faut ajouter une autorité de certification supplémentaire.

1. un CTC recoit une clé chiffrée d'une entité, la déchiffre et la rechiffre en utilisant une clé partagée avec l'entité de son domaine.

Mécanismes de transport de clés

Publiques

Procédé qui permet la mise à disposition de la clé publique d'une entité à d'autres entités.

Principale condition à assurer : authentification de la clé qui est souvent réalisée au moyen de tiers de confiance. Si on utilise une AC, l'authentification est faite par le certificat.

Secrètes

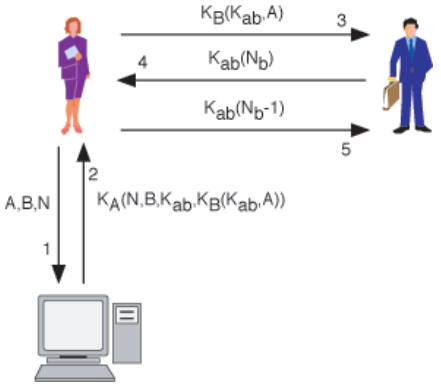
Procédé qui permet de transférer une clé secrète créée –ou obtenue s'il s'agit d'un tiers de confiance– par une entité à une autre entité.

Réalisation par de techniques de chiffrement, (a)symétriques.

18 mécanismes dans ISO/IEC 11770-2 et 3, dont 5 sont point à point, les autres utilisent des tiers de confiance comme CDC. Les différences résident surtout dans le nombre d'étapes de communication, les parties qui contrôlent les clés reçues, l'authentification. . .

Transport de clé secrète (symétrique avec tiers de confiance)

Exemple : Needham-Schroeder ; suppose l'existence d'un tiers de confiance qui sert de serveur d'authentification (AS). AS partage une clé secrète avec chaque acteur principal [3].

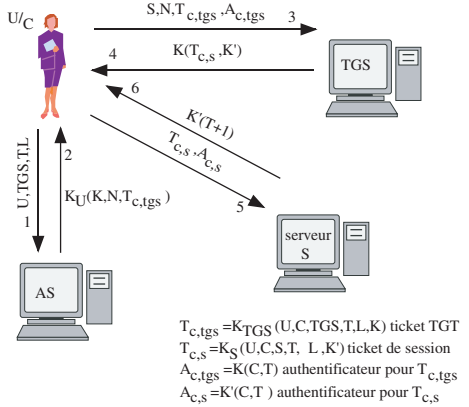


Utilité dans Kerberos

Permettre à l'utilisateur U du client C de prouver son identité à un serveur d'applications sans que ces données transitent par le réseau.
Requiert un tiers de confiance qui sert de **centre de distribution de clé** (KDC) pour le domaine ; il est composé de :

- serveur d'auth. (AS)
- distributeur de tickets (TGS)

qui sont sécurisés

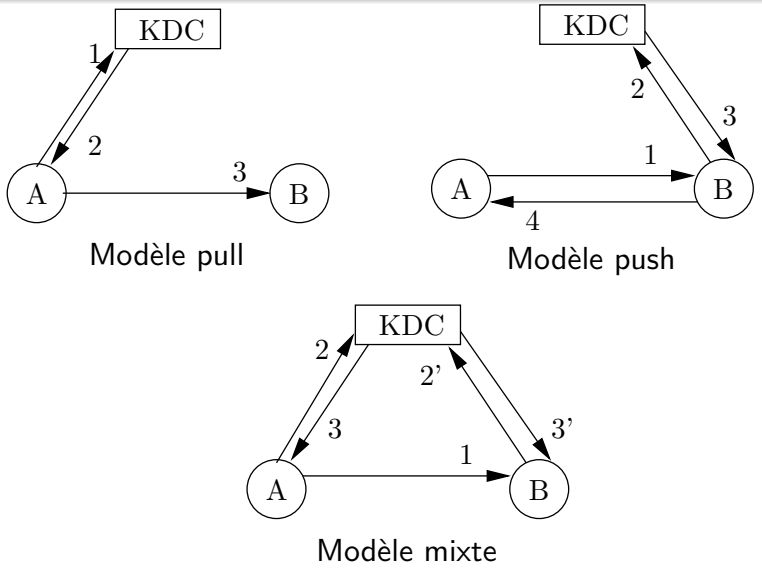


Systèmes analogues

Systèmes analogues

- Kryptoknight, par IBM qui utilise à la fois le modèle pull et le modèle push.
- Sesame, projet Européen qui ajoute un serveur de privilèges.

Autres modèles de distribution de clés



Mise à jour des clés

La clé une fois obtenue, la faire évoluer d'une session à l'autre par **mise à jour des clés** : procédé pour partager des clés construites au préalable en les faisant évoluer via des paramètres de session.
On définit une nouvelle clé de session K à partir :

- d'une clé partagée K_{AB}
- d'un paramètre F (nombre aléa, estampille, numéro de séquence)
- d'une fonction de calcul de clé f

Fonctionnement en deux étapes :

1. l'initiateur A choisit le paramètre de dérivation F et le transmet à B
2. A et B calculent K en utilisant la fonction de calcul f t.q.

$$K = f(K_{AB}, F)$$

Exemple de fonction f : utiliser une fonction de hachage cryptographique H à la concaténation de données : $K = H(K_{AB}; F)$

Un exemple de dérivation de clés PBKDF2

Fonction de dérivation de clé (PKCS#5 v2.0) qui calcule l'empreinte d'un mot de passe ou engendre des clés secrètes.

PBKDF2 applique une fonction (hachage, chiffrement, HMAC) à une entrée avec un sel et répète cette opération un grand nombre de fois (> 1000) afin de générer une clé.
`PBKDF2(PRF, secret, sel, iter, long)` où

- `PRF` est la fonction à utiliser
- `secret` le secret initial
- `sel` une valeur qui modifie le fonctionnement de PRF
- `c` le nombre d'itérations
- `long` la taille de la clé en sortie

WPA2 : `PBKDF2(HMAC-SHA1, passphrase, ssid, 4096, 256)`

Protocole de mise en accord (Diffie Hellman)

Procédé qui permet d'établir une clé partagée entre plusieurs entités de telle sorte qu'aucune d'entre elle ne puisse établir sa valeur par avance.

On cherche donc une solution qui permette à deux entités :

- qui ne se sont jamais rencontrés
- qui ne possèdent pas d'information partagée

de construire une clé secrète commune

- connue d'eux seuls
- inconnue de quiconque, même d'un indiscret qui écouterait leurs communications.

L'idée

Imaginer une solution facile à calculer pour les utilisateurs légaux et difficile pour un indiscret : une fonction à sens unique.

Un bon candidat est le **logarithme discret**.

Mise en accord par Diffie Hellman [4]

- **Etape préliminaire**
  - On choisit p un grand premier
  - On choisit α, $1 < \alpha < p$
- **Les clés :** Chaque utilisateur U :
  - choisit une valeur aléatoire r_U, $1 < r_U < p$ conservée secrète ;
  - publie $Y_U = \alpha^{r_U} \mod p$

A et B construisent une clé commune avec : Y_A et Y_B.

- A calcule $K = Y_B^{r_A} \mod p$
- B calcule $K = Y_A^{r_B} \mod p$

A et B ont alors une clé (secrète) commune K :

$$Y_B^{r_A} \equiv (\alpha^{r_B})^{r_A} \equiv (\alpha^{r_B})^{r_A} \equiv Y_A^{r_B} \mod p$$

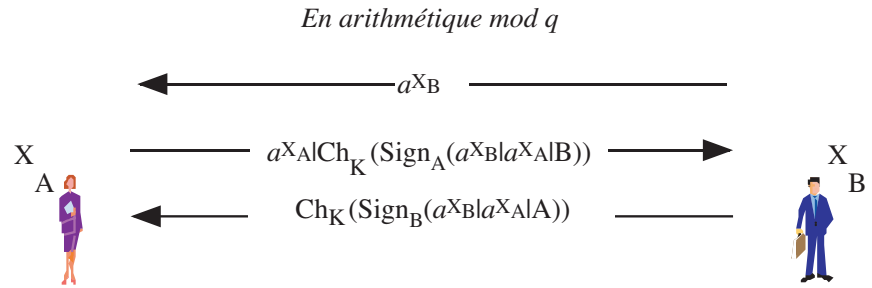
Clés ephémères et forward-secrecy

Différentes versions du protocole de Diffie-Hellman :

- **Anonymous Diffie-Hellman** DH without authentication ; the keys used in the exchange are not authenticated.
- **Fixed Diffie-Hellman** embeds the server's public parameter in the certificate, and the CA then signs the certificate. That is, the certificate contains the DH public-key parameters that never change.
- **Ephemeral Diffie-Hellman** uses temporary, public keys. Each run of the protocol uses a different public key. The authenticity of the server's temporary key can be verified by checking the signature on the key. Because the public keys are temporary, a compromise of the server's long term signing key does not jeopardize the privacy of past sessions. Assure le **forward-secrecy** : les clés précédentes ne peuvent pas être retrouvées même si les clés à long terme sont corrompues

Remède pour éviter l'attaque de l'homme du milieu

Protocole STS **Station To Station**. Variante de Diffie Hellman auquel on a jouté une authentification. Une autre variante est utilisée dans le protocole IPsec, intégré dans IPv6.

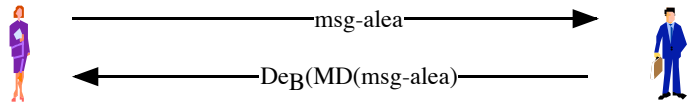


Identification & authentification

identification affirmation de l'identité d'une entité au moyen de son identifiant. (Je suis Bruno)
authentification procédé de vérification de l'identité d'une entité. (Je suis Bruno et en voici la preuve)

L'identification permet de connaitre l'identité d'une entité et l'authentification de vérifier cette identité.

Exemple d'authentification « asymétrique »

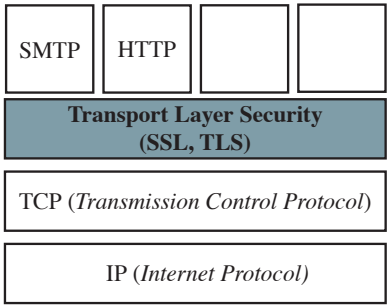


Si on n'utilise pas de calcul de l'empreinte du msg-aléa, ce protocole peut subir des attaques (msg-aléa/De_B(msg-aléa) connus).
Requiert qu'Alice connaisse la clé publique de Bob au préalable.

Sécurité de la couche de transport (TCP)

Protocoles pour sécuriser TCP :

- **Secure Socket Layer**
- ~~**Private Communication Technology**~~ (Microsoft)
- **Transport Layer Security** standardisé par l'IETF, évolution de SSL3



Protocole vs Implementation

Ne pas confondre le protocole avec son implémentation ! TLS (qui remplace maintenant SSL) est implémenté par de nombreuses libraires. Voir à ce sujet `https://en.wikipedia.org/wiki/Comparison_of_TLS_implementations`. Les plus classiques :

- OpenSSL
- LibreSSL
- BoringSSL
- GnuTLS

Et voir aussi les recommandations sur Crypto Best Practice

SSL & TLS

SSL fournit authentification, compression, intégrité, chiffrement. plusieurs mécanismes d'authentification et de chiffrement ; protège tout protocole au niveau applicatif (`http`,`smtp`)
TLS repose sur SSL3.0. Deux couches :

- **Rencontre** ou Handshake Protocol
- **Communication** ou Record Protocol

qui fournissent les services suivants :

- **confidentialité de la connexion** par AES, Chacha-20
- **intégrité de la connexion** par un MAC utilisant une clé en valeur initiale (SHA384 ou Poly1305). Voir les différents SHA

Openssl ciphers (2018)

1. Shell

```
yuzu:~ bmartin$ openssl ciphers|grep DES
ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-E
CDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA:ECDHE-ECDSA-AES256-SHA:SRP-DSS-AES-256-CBC-SHA:SR
P-RSA-AES-256-CBC-SHA:SRP-AES-256-CBC-SHA:DH-DSS-AES256-GCM-SHA384:DHE-DSS-AES256-GCM-SHA
384:DH-RSA-AES256-GCM-SHA384:DHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES256-SHA256:DHE-DSS-AES2
56-SHA256:DH-RSA-AES256-SHA256:DH-DSS-AES256-SHA256:DHE-RSA-AES256-SHA:DHE-DSS-AES256-SHA
:DH-RSA-AES256-SHA:DH-DSS-AES256-SHA:DHE-RSA-CAMELLIA256-SHA:DHE-DSS-CAMELLIA256-SHA:DH-R
SA-CAMELLIA256-SHA:DH-DSS-CAMELLIA256-SHA:ECDH-RSA-AES256-GCM-SHA384:ECDH-ECDSA-AES256-GC
M-SHA384:ECDH-RSA-AES256-SHA384:ECDH-ECDSA-AES256-SHA384:ECDH-RSA-AES256-SHA:ECDH-ECDSA-A
ES256-SHA:AES256-GCM-SHA384:AES256-SHA256:AES256-SHA:CAMELLIA256-SHA:PSK-AES256-CBC-SHA:E
CDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-EC
DSA-AES128-SHA256:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES128-SHA:SRP-DSS-AES-128-CBC-SHA:SRP
-RSA-AES-128-CBC-SHA:SRP-AES-128-CBC-SHA:DH-DSS-AES128-GCM-SHA256:DHE-DSS-AES128-GCM-SHA2
56:DH-RSA-AES128-GCM-SHA256:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES128-SHA256:DHE-DSS-AES12
8-SHA256:DH-RSA-AES128-SHA256:DH-DSS-AES128-SHA256:DHE-RSA-AES128-SHA:DHE-DSS-AES128-SHA:
DH-RSA-AES128-SHA:DH-DSS-AES128-SHA:DHE-RSA-SEED-SHA:DHE-DSS-SEED-SHA:DH-RSA-SEED-SHA:DH-
DSS-SEED-SHA:DHE-RSA-CAMELLIA128-SHA:DHE-DSS-CAMELLIA128-SHA:DH-RSA-CAMELLIA128-SHA:DH-DS
S-CAMELLIA128-SHA:ECDH-RSA-AES128-GCM-SHA256:ECDH-ECDSA-AES128-GCM-SHA256:ECDH-RSA-AES128
-SHA256:ECDH-ECDSA-AES128-SHA256:ECDH-RSA-AES128-SHA:ECDH-ECDSA-AES128-SHA:AES128-GCM-SHA
256:AES128-SHA256:AES128-SHA:SEED-SHA:CAMELLIA128-SHA:IDEA-CBC-SHA:PSK-AES128-CBC-SHA:ECD
HE-RSA-RC4-SHA:ECDHE-ECDSA-RC4-SHA:ECDH-RSA-RC4-SHA:ECDH-ECDSA-RC4-SHA:RC4-SHA:RC4-MD5:PS
K-RC4-SHA:ECDHE-RSA-DES-CBC3-SHA:ECDHE-ECDSA-DES-CBC3-SHA:SRP-DSS-3DES-EDE-CBC-SHA:SRP-RS
A-3DES-EDE-CBC-SHA:SRP-3DES-EDE-CBC-SHA:EDH-RSA-DES-CBC3-SHA:EDH-DSS-DES-CBC3-SHA:DH-RSA-
DES-CBC3-SHA:DH-DSS-DES-CBC3-SHA:ECDH-RSA-DES-CBC3-SHA:ECDH-ECDSA-DES-CBC3-SHA:DES-CBC3-S
HA:PSK-3DES-EDE-CBC-SHA
yuzu:~ bmartin$
```


Openssl ciphers (2024)

Default ⌥⌘1

```
satsuma:~ bmartin$ openssl ciphers
TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256:ECDHE
-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES256-GCM-SHA384:E
CDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:DHE-RSA-CHACHA20-POLY13
05:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES128-GCM-
SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES256-SHA256:E
CDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA256:ECDHE-ECD
SA-AES256-SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA:ECD
HE-RSA-AES128-SHA:DHE-RSA-AES128-SHA:RSA-PSK-AES256-GCM-SHA384:DHE-PSK-AES256-GC
M-SHA384:RSA-PSK-CHACHA20-POLY1305:DHE-PSK-CHACHA20-POLY1305:ECDHE-PSK-CHACHA20-
POLY1305:AES256-GCM-SHA384:PSK-AES256-GCM-SHA384:PSK-CHACHA20-POLY1305:RSA-PSK-A
ES128-GCM-SHA256:DHE-PSK-AES128-GCM-SHA256:AES128-GCM-SHA256:PSK-AES128-GCM-SHA2
56:AES256-SHA256:AES128-SHA256:ECDHE-PSK-AES256-CBC-SHA384:ECDHE-PSK-AES256-CBC-
SHA:SRP-RSA-AES-256-CBC-SHA:SRP-AES-256-CBC-SHA:RSA-PSK-AES256-CBC-SHA384:DHE-PS
K-AES256-CBC-SHA384:RSA-PSK-AES256-CBC-SHA:DHE-PSK-AES256-CBC-SHA:AES256-SHA:PSK
-AES256-CBC-SHA384:PSK-AES256-CBC-SHA:ECDHE-PSK-AES128-CBC-SHA256:ECDHE-PSK-AES1
28-CBC-SHA:SRP-RSA-AES-128-CBC-SHA:SRP-AES-128-CBC-SHA:RSA-PSK-AES128-CBC-SHA256
:DHE-PSK-AES128-CBC-SHA256:RSA-PSK-AES128-CBC-SHA:DHE-PSK-AES128-CBC-SHA:AES128-
SHA:PSK-AES128-CBC-SHA256:PSK-AES128-CBC-SHA
satsuma:~ bmartin$
```


Identification–handshake

C'est le moyen pour Alice de vérifier l'identité de Bob.
pk_B la clé publique de Bob et sk_B sa clé privée.

$$\begin{array}{l|l} A \to B & r = \text{un message aléatoire} \\ B \to A & c = \{r\}_{sk_B} \end{array}$$

Signer un message aléatoire r fourni par un tiers et le réexpédier peut s'avérer dangereux.
On pourrait utiliser une fonction de hachage h afin que Bob signe $h(r)$. Mais le danger persiste.

Identification–handshake

Mieux vaut que Bob signe un message de son cru

$A \to B$	*"Bonjour, est-ce Bob ?"*
$B \to A$	$m =$ *"Alice, je suis bien Bob"*
	$c = \{h(m)\}_{sk_B}$

Authentification–handshake

Alice n'a pas forcément déjà connaissance de la clé publique de Bob. Comment informer sûrement quelqu'un de sa clé publique ?

$A \to B$	*"Bonjour"*
$B \to A$	*"Bonjour, je suis Bob. Voici ma clé publique"* pk_B
$A \to B$	*"Prouve-le."*
$B \to A$	$m =$ *"Alice, c'est bien Bob"*
	$c = \{h(m)\}_{sk_B}$

N'importe qui peut se faire passer pour Bob aux yeux d'Alice, par une attaque MIM.

Transmettre un certificat–handshake

Certificat garantit la relation entre une identité et clé publique.

$A \to B$	*"Bonjour"*
$B \to A$	*"Bonjour, je suis Bob. Voici mon certificat"* $cert_B$
$A \to B$	*"Prouve-le."*
$B \to A$	$m =$ *"Alice, c'est bien Bob"*
	$c = \{h(m)\}_{sk_B}$

Eve pourrait se substituer à Bob dans les premiers échanges, mais échouera au dernier.

Echanger un secret–handshake

La communication par clés publiques est coûteuse, une fois finie la phase d'authentification, on échange une clé symétrique.

$A \to B$	*"Bonjour"*
$B \to A$	*"Bonjour, je suis Bob. Voici mon certificat"* $cert_B$
$A \to B$	*"Prouve-le".*
$B \to A$	$m =$ *"Alice, c'est bien Bob"*
	$c = \{h(m)\}_{sk_B}$
$A \to B$	*"Ok Bob, voici notre secret :"*
	$s = \{secret\}_{pk_B}$
$B \to A$	$m' = \{\text{message de Bob}\}_{secret}$

Attaque MIM

L'homme du milieu Melchior peut s'interposer dans les 5 premiers échanges. Arrivé au sixième, il peut brouiller le message de Bob, quitte à ne pas envoyer un message très intelligible à Alice :

$$\begin{array}{ll} B \to M & m' = \{\text{message de Bob}\}_{secret} \\ M \to A & \text{altération de } m' \end{array}$$

Alice n'a aucune certitude quant à l'existence de Melchior, même si elle trouve suspect le dernier message de Bob.

SSL–handshake

Pour éviter cette incertitude, mieux vaut utiliser un MAC :

$$M = h(\text{un message de Bob}, secret)$$

$A \to B$	*"Bonjour"*
$B \to A$	*"Bonjour, je suis Bob. Voici mon certificat"* $cert_B$
$A \to B$	*"Prouve-le".*
$B \to A$	$m =$ *"Alice, c'est bien Bob"*
	$c = \{h(m)\}_{sk_B}$
$A \to B$	*"Ok Bob, voici notre secret :"*
	$s = \{secret\}_{pk_B}$
$B \to A$	$m' = \{\text{message de Bob}\}_{secret}$
	$M = h(\text{message de Bob}, secret)$

Melchior peut pertuber ce qu'il veut, M aura au moins l'avantage d'en avertir le destinataire.

La communication : record protocol

Ce protocole transmet un message de taille arbitraire. Il le découpe en blocs, le comprime éventuellement, ajoute un MAC, chiffre et transmet le résultat en ajoutant un numéro de séquence pour détecter s'il manque des messages ou si certains ont été altérés.

Il assure :

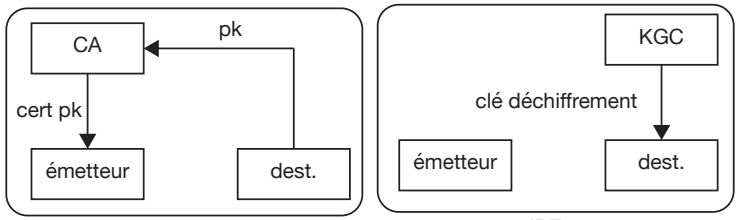
- confidentialité des données transmises par l'application
- intégrité des données
- authentification de l'origine

Une fois le record protocol achevé, les données chiffrées sont fournies à `TCP`.

Plan

Identity Based Encryption (IBE)

Génère la clé publique du destinataire via (`bob@ibe.com`).
`http://www.hackinsight.org/news,187.html`
Le destinataire s'adresse à un générateur de clé privée (KGC) pour obtenir la clé privée de déchiffrement.
Inconvénient : KGC peut calculer la clé de déchiffrement de tous et le destinataire doit avoir confiance dans son KGC.



Schémas de chiffrement sans certificat

Propriétés

1. le schéma est sûr sans que pk soit validée par un certificat
2. le schema reste sûr contre les attaques

2 biclés nécessaires :

- dest. génère une biclé traditionnelle :
  - sk : valeur secrète (éviter confusion avec sk complète)
  - pk : publique mais non certifiée
- une biclé d'identité (IB) :
  - pk : id. digital du dest.
  - sk : clé privée IB fournie par un KGC, clé privée partielle

Chiffrer : l'émetteur utilise l'id. digital du dest. et sa clé publique.
Déchiffrer : dest. utilise la valeur secrète et la clé privée partielle.
Beaucoup de modèles proposés.

Plan

Pourquoi le post-quantique ?

Store Now, Decrypt Later (NSA's Utah Data Center)

Tailles des clés recommandées

Paramètres de sécurité actuels (algorithmique et vitesse CPU)

Security (bits) (symmetric)	RSA	DLOG	EC
48	480	480	96
56	640	640	112
64	816	816	128
80	1248	1248	160
112	2432	2432	224
128	3248	3248	256
160	5312	5312	320
192	7936	7936	384
256	15424	15424	512

Source: ECRYPT II Yearly Key Size Report

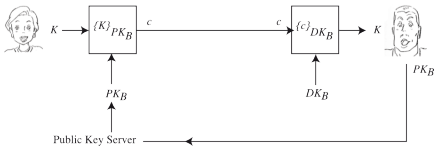
Short-term security (days to months)

Mid-term security (years to decades)

Long-term security (many years)

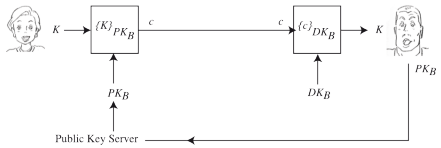
56153 a été factorisé par un ordinateur à 4 qubits en 2014...

Transport de clé

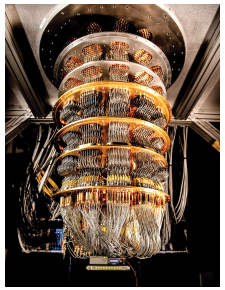


- PKC actuelle : RSA, DH, ECDH
- Basé sur théorie des nombres
- Importance ECC

Transport de clé



- PKC actuelle : RSA, DH, ECDH
- Basé sur théorie des nombres
- Importance ECC



- algo. Shor QP
- algo. Simon QP
- 2300 qubits cassent RSA-1024
- IBM Osprey : 433 qubits

Vers le Post-Quantique

Remplacer PKC traditionnelle

- Algos Shor et Simon solve résolvent dans QP :
  - factorisation. RSA est mort
  - DLP dans les corps finis [2]. DSA est mort
  - DLP sur les courbes elliptiques. ECDH est mort
- **Post-quantum crypto** doit résister à ces attaques
- Remplacer RSA, DSA, ECDH par de nouveaux standards
- Standards de 2022 reposent sur le problème **Learning With Errors** sur des réseaux arithmétiques.
  - CRYSTALS-Kyber pour chiffrer (keysizes : pk=1184, dk=2400, block=1088)
  - CRYSTALS-Dilithium pour signer
- Actuellement : OpenSSH, Cloudflare, AWS, IBM backup device

2. DLOG a besoin de moitié moins de qubits que pour la factorisation d'un entier de même taille

Approches de la PQC

- **Codes correcteurs d'erreurs**
- **Réseaux arithmétiques**
- **Fonctions de hachage**
- Formes quadratiques multivaluées
- Isogénies supersingulières

En remplacement des standards actuels de PKC.

Réseaux entiers

Definition

Un **réseau entier** est un sous-groupe de $\mathbb{Z}^n$, i.e. un ensemble de vecteurs à coordonnées entières qui vérifie :

- l'opposé de tout vecteur du réseau est dans le réseau ;
- la somme de 2 vecteurs du réseau est encore dans le réseau ;

Tout réseau a une base, i.e. une famille $b = (b_1, \ldots, b_r)$, tq le réseau L est l'ensemble des combinaisons linéaires entières de vecteurs de b.

$$L(b) = \left\{ \sum_{i=1}^{r} b_i x_i : x_1, \ldots, x_r \in \mathbb{Z} \right\}$$

Tout réseau possède non seulement une base, mais une infinité de bases qui ont toutes le même nombre de vecteurs et qui définissent la **dimension** du réseau.

Invariants d'un réseau

Un **invariant** d'un réseau L est une quantité définie dans un réseau qui ne dépend que du réseau et non du choix de la base (p.e., la dimension). On introduit un autre invariant :

- les **minima successifs** du réseau. On appelle premier minimum d'un réseau la longueur du plus court vecteur non nul. Le i^{e} minimum d'un réseau L est le plus petit réel positif λ pour lequel il existe dans L, i vecteurs linéairement indépendants de normes $|| b_j ||^2 \leq \lambda$. On note ce réel $\Lambda_i(L)$.

En dimension 2 : Gauss

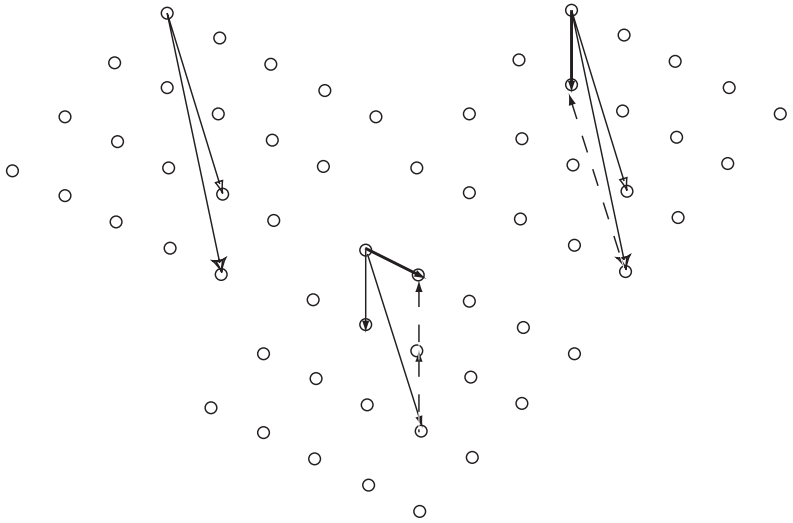
Base $b = (u, v)$ est **réduite** si elle réalise les minimas $\Lambda_1(L), \Lambda_2(L)$. Une base réduite est formée de vecteurs aussi courts que possible. L'algorithme d'Euclide calcule cette base réduite :

Algorithme

si $|| u || < || v ||$ ***alors*** $v \leftrightarrow u$;
répéter
  choisir k *t.q.* $-\frac{1}{2}\langle v, v\rangle < \langle u - kv, v\rangle \leq \frac{1}{2}\langle v, v\rangle (k \leftarrow \left\lfloor \frac{\langle u,v\rangle}{\langle v,v\rangle} \right\rceil)$
  $u \leftarrow u - k.v$; $v \leftrightarrow u$
jusqu'à $|| u || \leq || v ||$

A chaque étape, on diminue autant que possible la longueur du plus long vecteur de b par translation parallèlement à l'autre. L'algorithme termine quand le vecteur ainsi obtenu est plus long que le vecteur resté fixe.

Exemple



Réduction de réseaux quand $n > 2$

Quand $n > 2$, chercher une base réduite devient plus compliqué. L'algorithme LLL de réduction de réseaux trouve une base réduite sur la donnée d'une base d'entrée en temps polynomial (mais coûteux $\tilde{O}(n.\sharp b^5)$).

Deux problèmes difficiles :

- SVP : **Shortest Vector Problem** qui doit trouver le plus court vecteur d'un réseau (NP-difficile)
- CVP : **Closest Vector Problem** pour lequel, étant données les coordonnées d'un point hors du réseau, il faut trouver le point du réseau le plus proche (aussi difficile que SVP).

Problème LWE (vulgarisé)

Introduit en 2005 par O. Regev, utile à la conception de chiffres. Combinaisons linéaires entières de s_0 et s_1 (N), faciles à résoudre :

$$N = \begin{cases} 5s_0 + 2s_1 = 27 \\ 2s_0 + 0s_1 = 6 \end{cases} \qquad P = \begin{cases} 5s_0 + 2s_1 = 28 \\ 2s_0 + 0s_1 = 5 \end{cases}$$

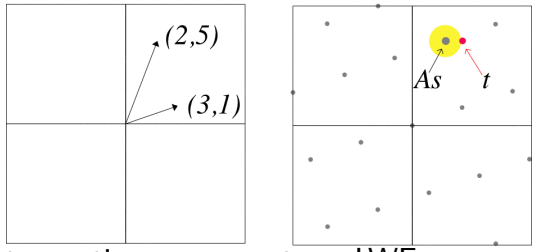
Mais si on perturbe le système, le problème (P) devient difficile : D'autant plus quand on augmente le nombre de paramètres.
LWE : s est un secret modulo un grand entier q. Etant données un nombre arbitraire de vecteurs aléatoires a_i et les valeurs des calculs $\langle a_i, s\rangle + e_i$ pour e_i une petite erreur aléatoire, trouver s.

Pour $a_0 = (5, 2)$, $a_1 = (2, 0)$ et la valeur perturbée $(28, 5)$, trouver $s = (3, 6)$ devient difficile (surtout avec $\mathbb{Z}_{4093}^{640\times256}$).

LWE et réseaux

$$\underset{t}{\begin{pmatrix}3\\5\end{pmatrix}} = \underset{A}{\begin{pmatrix}3 & 2\\1 & 5\end{pmatrix}} \underset{s}{\begin{pmatrix}0\\1\end{pmatrix}} + \underset{e}{\begin{pmatrix}1\\0\end{pmatrix}}$$

LWE est une construction reposant sur un problème de réseaux. On peut réduire CVP vers LWE : on sait résoudre CVP ssi on sait résoudre LWE



Les nouveau standards post-quantiques reposent sur LWE, vainqueurs d'une compétition du NIST en juillet 2022 : CRYSTALS-Kyber (chiffrement) et CRYSTALS-Dilithium (signature) (ainsi que FALCON et SPHINCS+) [1].

Kyber.CPAPKE

Génération de clés : $\mathbf{s}, \mathbf{e} \leftarrow \chi$, $sk = \mathbf{s}$, $pk = \mathbf{t} = \mathbf{As} + \mathbf{e}$
Chiffrement (de m étant donné pk dit "noisy El Gamal") :

- $\mathbf{r}, \mathbf{e_1}, e_2 \leftarrow \chi$ (aléatoires et $<< q/2$, q un grand premier)
- $\mathbf{u} = \mathbf{rA} + \mathbf{e_1}$ (clé privée éphémère)
- $v = \langle \mathbf{r}, \mathbf{t} \rangle + e_2 + \text{Enc}(m)$
- $c = (\mathbf{u}, v)$

Déchiffrement (de c étant donné sk) :

- $m = \text{Dec}(v - \langle \mathbf{u}, \mathbf{s} \rangle)$

Enc et Dec sont des fonctions de codage et décodage d'un bit. Enc peut être pris comme l'entier le plus proche de $q/2$ et Dec l'opération inverse. En réalité, c'est plus compliqué ! On utilise des structures algébriques plus complexes avec des hypothèses sur les distributions des bruits.

Exemple pour $q = 47$

$A = \begin{pmatrix} 3 & 2 \\ 1 & 5 \end{pmatrix}$. $s = (0,1)$, $e = (1,0)$ aléatoires. $\mathbf{t} = \mathbf{As} + \mathbf{e} = (3,5)$.

$$pk = (\mathbf{A}, \mathbf{t}, q) \text{ et } sk = (\mathbf{s}, q)$$

Le chiffré du bit $b = 1$ est $(\mathbf{u}, v)$: $\begin{cases} \mathbf{u} = \mathbf{r}.\mathbf{A} + \mathbf{e_1} \\ v = \langle \mathbf{r}, \mathbf{t} \rangle + e_2 + b.\lceil q/2 \rceil \end{cases}$
pour $\mathbf{r} = (9,8), \mathbf{e_1} = (1,2)$ aléatoires, $\mathbf{u} = (36,68)$ et $v = 97$ pour $e_2 = 7 < q/4$ aléatoire. On transmet $((36,68),97)$.

Déchiffrement : $v - \langle \mathbf{u}, \mathbf{s} \rangle = 29 > q/2$ et on décode le chiffré en 1.

Python

```
A,q = Matrix([[3,2],[1,5]]),47
s,e = Matrix([0,1]),Matrix([1,0])
t = A*s+e
pk, sk = (A,t,q), (s,q)
def chiffre(bit,pk):
    A,t,q = pk[0],pk[1],pk[2]
    r = Matrix([randint(1,q//4) for i in range(2)]).T
    e1 = Matrix([randint(1,q//4) for i in range(2)]).T
    u = r*A+e1
    e2 = randint(1,q//4)
    v = (r.dot(t)+e2+(bit*(q//2)))
    return u,v
def dechiffre(c,sk):
    s,q = sk[0],sk[1]
    u,v = c[0],c[1]
    res = (v-u.dot(s))
    return 1-int((res<(q//2))==true)
```


Jeux de paramètres

Kyber propose différents jeux de clés avec des tailles différentes qui indiquent le niveau de sécurité (en octets ; ct taille du chiffré) :

- Kyber-512 : sk : 1632, $pk = 800$, $ct = 768 \approx$ AES-128
- Kyber-768 : sk : 2400, $pk = 1184$, $ct = 1088 \approx$ AES-192
- Kyber-1024 : sk : 3168, $pk = 1568$, $ct = 1568 \approx$ AES-256

Recommandations d'utilisation :

- Kyber hybride en le combinant p.e. avec ECDH
- préférer Kyber-768 qui atteint une sécurité supérieure à 128 bits contre les attaques classiques et quantiques

Déjà implémenté :

- Cloudflare
- AWS key management service
- IBM pour un support de sauvegarde

OpenSSH et PQC

A partir de la version 9.0 `OpenSSH` intègre un chiffrement asymétrique hybride qui combine un chiffre post quantique NTRU, basé sur les réseaux arithmétiques (et normalisé ANSI X9.98), en conjonction avec des courbes elliptiques sur la courbe X25519.

L'intégration de NTRU a eu lieu avant la publication du standard. Le but était de prévenir les attaques de type "store-now, decrypt-later", dans lesquelles un attaquant récolte des données chiffrées afin de pouvoir les pirater ultérieurement à l'aide d'un ordinateur quantique.

Plus d'informations sur
`https://medium.com/cambridge-quantum-computing/openssh-bravely-addresses-the-quantum-threat-86b03e38c2ba`

R. Avanzi, J. Bos, E. Kiltz, T. Lepoint, V. Lyubashevsky, J.M. Schanck, P. Schwabe, G. Seiler, and D. Stehle.
Crystals Kyber.
https://pq-crystals.org/index.shtml, 2017.

W. Fumy.
Key management techniques.
In State of the art in applied cryptography, number 1528 in LNCS, pages 209–223. Springer Verlag, 1997.

R. Oppliger.
Internet and intranet security.
Artech House, 1998.

D. Stinson.
Cryptographie, théorie et pratique.
International Thomson Publishing, 1995.

Et les chiffres symétriques ?

Le chiffrement symétrique (AES-256), est supposé robuste aux attaques quantiques. Un ordinateur quantique ne devrait pas être en mesure de réduire suffisamment le temps d'attaque pour être efficace si la taille des clés est assez grande.

L'algorithme de Grover peut réduire le temps d'attaque par force brute à sa racine carrée. Ainsi, pour AES-128, le temps d'attaque se réduit à 2^{64} (pas très sûr), tandis que AES-256 se réduit à 2^{128}, ce qui est encore considéré comme sûr.

Si vous utilisez AES, privilégiez AES-256-CTR par rapport à AES-128 pour la résistance quantique.