Table of contents

Protocoles d'Établissement de Clés (KEP)

KEP Définition et Propriétés

Définitions et Classification

Protocole d'établissement de clés (KEP) : Mécanisme permettant aux entités de partager un secret pour leurs échanges cryptographiques.

Deux types :

- **Key Transport Protocol (KTP)** : Une entité crée et transmet la clé
- **Key Agreement Protocol (KAP)** : Les entités dérivent conjointement la clé

Classification temporelle :

- **Pré-distribution** : Clés déterminées à priori
- **Dynamic Key Establishment (DKE)** : Clés changeant à chaque exécution

Key Establishment Protocols

Key Agreement

Key Transport

symétrique + pré-dist.

symétrique + DKE

symétrique + DKE

asymétrique + pré-dist.

asymétrique + DKE

asymétrique + DKE

Texte original

Un **protocole d'établissement de clés** (key establishment protocol ou **KEP**) est celui qui met à disposition des entités impliquées un **secret partagé** (une clé) qui servira comme base pour des échanges cryptographiques ultérieurs.
Les deux variantes des KEP sont les **protocoles de transport de clé** (key transport protocol ou **KTP**) et les **protocoles de mise en accord** (key agreement protocol ou **KAP**).

- Un **key transport protocol (KTP)** est un mécanisme permettant à une entité de **créer une clé secrète et de la transférer** à son (ses) correspondant(s).
- Un **key agreement protocol (KAP)** est un mécanisme permettant à deux (ou plusieurs) entités de **dériver une clé** à partir d'informations propres à chaque entité.

Key pré-distribution schemes sont ceux où les clés utilisées sont entièrement **déterminées à priori** (p.ex. à partir des calculs initiaux).
Dynamic key establishment schemes (DKE) sont ceux où les clés **changent pour chaque exécution** du protocole.

Révision rapide

KEP : Protocoles pour établir un secret partagé.

- **KTP** : transport de clé
- **KAP** : accord mutuel sur la clé
- **Pré-distribution** vs **DKE** (dynamique)

Propriétés des KEP

Propriétés d'authentification :

- **Implicit key authentication** : Assurance que seul le correspondant peut accéder à la clé (sans preuve de possession)
- **Key confirmation** : Assurance que le correspondant possède effectivement la clé
- **Explicit key authentication** : Implicit + confirmation
- **Authenticated KEP** : KEP fournissant l'authentification de clé

Propriétés de sécurité temporelle :

- **Perfect Forward Secrecy (PFS)** : Compromission des clés long terme ne révèle pas les clés de sessions passées
- **Future Secrecy** : Clés futures protégées même si clés long terme compromises (par attaquant passif)
- **Deniability/Repudiability** : Impossibilité de prouver la participation à un tiers (comme Zero-Knowledge)

Types d'attaques :

- **Attaque passive** : Enregistrement et analyse des échanges
- **Attaque active** : Modification ou injection de messages
- **Known-key attack** : Exploitation d'une clé de session compromise pour attaquer les clés futures

Texte original

Propriétés des protocoles d'établissement de clés :

- **Implicit key authentication** (ou key authentication) : propriété par laquelle une entité est assurée que seul(s) son (ses) correspondant(s) peut (peuvent) accéder à une clé secrète. Cependant, ceci ne spécifie rien sur le fait de posséder effectivement la clé.
- **Key confirmation** : propriété permettant à une entité d'être sûre que ses correspondants sont en possession des clés de session générées.
- **Explicit key authentication** : = implicit key authentication + key confirmation.
- Un **authenticated KEP** est un KEP capable de fournir key authentication.

Attaques :

- Une **attaque passive** est celle qui essaye de démonter un système cryptographique en se limitant à l'**enregistrement et à l'analyse** des échanges.
- Une **attaque active** fait intervenir un adversaire qui **modifie ou injecte** des messages.
- Un protocole est dit vulnérable à un **known-key attack** si lorsqu'une clé de session antérieure est compromise, il devient possible : (a) de compromettre par une attaque passive des clés futures et/ou (b) de monter des attaques actives visant l'usurpation d'identité.

Propriétés modernes :

- **Perfect Forward Secrecy (PFS)** est une caractéristique qui garantit la **confidentialité des clés de sessions utilisées par le passé** même si les clés long terme (par exemple la clé privée du destinataire) est compromise.
- **Future Secrecy** : Le protocole garantit la **sécurité des échanges ultérieurs** (les clés des sessions futures sont protégées) même si les clés long terme sont compromises par un attaquant passif.
- **Deniability / Repudiability** (répudiabilité) : À l'image des protocoles d'authentification Zéro-Knowledge, permet aux entités de garantir l'authentification des échanges sans apporter des informations qui permettraient de prouver à un tiers leur participation dans l'échange cryptographique.

Révision rapide

Authentification :

- **Implicit** : seul le correspondant accède à la clé
- **Key confirmation** : preuve de possession
- **Explicit** = Implicit + confirmation

Sécurité :

- **PFS** : clés passées protégées si compromission
- **Future Secrecy** : clés futures protégées
- **Deniability** : participation non prouvable

KAP

Symétrique

avec Pré-distribution

Cas trivial :

Pour n utilisateurs avec un Key Distribution Center (KDC) :

1. KDC génère $\frac{n(n-1)}{2}$ clés différentes (une par paire d'utilisateurs)
2. KDC distribue $n-1$ clés à chaque utilisateur via canal confidentiel

Avantages :

- Inconditionnellement sûr contre complots d'utilisateurs (sécurité information-théorique)

Inconvénients :

- Complexité $O(n^2)$ en stockage pour le KDC
- Complexité $O(n)$ en clés par utilisateur
- Non scalable

Texte original

KAP Symétrique avec Pré-distribution - Cas Trivial
Soit un nombre n d'utilisateurs avec un **centre de distribution de clés** (key distribution center ou **KDC**).
On peut construire un KAP symétrique avec pré-distribution trivial de la façon suivante :

(1) KDC génère $n(n-1)/2$ clés différentes (une clé différente pour chaque couple d'utilisateurs).

(2) KDC distribue ensuite par un **canal confidentiel et authentique** les clés en donnant $n-1$ clés à chaque utilisateur.

Si KDC génère les clés de façon vraiment **aléatoire**, ce système est **inconditionnellement sûr contre des complots d'utilisateurs** (même en admettant que $n-2$ utilisateurs complotent, ils ne pourraient pas trouver la clé des deux autres) par construction du protocole.
Problème de ce protocole :

- $O(n^2)$ en **stockage de clés** par le KDC.
- $O(n)$ en **clés secrètes échangées** pour chaque entité.

Révision rapide

KAP symétrique trivial :

- $n(n-1)/2$ clés pour n utilisateurs
- Inconditionnellement sûr
- Problème : $O(n^2)$ en stockage

---

avec Dynamic Key Establishment (DKE)

Exemple Simple

Initialisation : A et B partagent une clé long terme S

Protocole :

1. $A \to B : r_a$ (nombre aléatoire)
2. $A \leftarrow B : r_b$ (nombre aléatoire)
3. Clé de session : $K := E_S(r_a \oplus r_b)$

Propriétés :

- Entity authentication
- Implicit key authentication

- Key confirmation
- Perfect Forward Secrecy

AKEP2 (Authenticated Key Exchange Protocol 2)

Initialisation : A et B partagent S (pour MAC) et S' (pour clé de session)

Protocole :

1. $A \to B : r_a$
2. $A \leftarrow B : T = (B, A, r_a, r_b), h_S(T)$
3. $A \to B : (A, r_b), h_S(A, r_b)$
4. Clé de session : $K := h'_{S'}(r_b)$

Propriétés :

- Entity authentication (mutuelle)

- Implicit key authentication
- Key confirmation
- Perfect Forward Secrecy

Note : Clé dépend uniquement de B et de la clé long terme S' !

Texte original

KAP Symétriques avec Dynamic Key Establishment
Ces méthodes permettent aux entités impliquées de dériver des **clés de courte durée** (typiquement, des clés de session) à partir de **secrets de longue durée** qui, pour ces protocoles, sont des clés symétriques.
Exemple intuitif :
(Initialisation): A et B partagent une clé symétrique long terme S

(1) $A \to B : r_a$; A génère un nb. aléatoire et l'envoie à B

(2) $A \leftarrow B : r_b$; B fait de même

A et B calculent la clé de session: $K := E_S(r_a \oplus r_b)$
Propriétés : - **Entity authentication** : NON : par construction du protocole, les r_i peuvent être envoyés par une entité quelconque. - **Implicit key authentication** : OUI : seules les entités partageant la clé symétrique long terme S peuvent accéder à la clé de session K. - **Key confirmation** : NON : les r_i étant aléatoires, ils peuvent être modifiés par un adversaire et empêcher A et B de se mettre d'accord sur la clé de session K. Ceci ne serait pas détecté par le protocole. - **Perfect Forward Secrecy** : NON : si la clé long terme S est compromise, toutes les clés de session précédentes peuvent être facilement calculées par un adversaire qui aurait enregistré tous les échanges.
Authenticated Key Exchange Protocol 2 (AKEP2)
(Init.) : A et B partagent deux clés symétriques long terme S et S'. S est utilisé pour générer des MACs $h_S()$ (afin de garantir l'intégrité et l'authentification d'entités) et S' pour la génération de la clé de session K.

(1) $A \to B : r_a$; A génère un nb. aléatoire et l'envoie à B

(2) $A \leftarrow B : T = (B, A, r_a, r_b), h_S(T)$; B idem + identités + MAC de tout

(3) $A \to B : (A, r_b), h_S(A, r_b)$; A vérifie les identités et le r_a fournis par B ; ensuite, il envoie identité + r_b + MAC du tout.

La clé est calculée bilatéralement avec un MAC dédié $h'_{S'}()$: $K := h'_{S'}(r_b)$.
Propriétés : - **Entity authentication** : OUI mutuelle (fournie par les MACs). - **Implicit key authentication** : OUI. - **Key confirmation** : NON (pas d'évidence que la clé S' est connue du correspondant). - **Perfect forward secrecy** : NON (si la clé S' est compromise, les clés de session K précédentes aussi).

La clé dépend seulement de B (et de la clé long terme S') mais le protocole peut être facilement modifié pour que la clé dépende aussi de A et en faire un "vrai" KAP.

Révision rapide

KAP symétrique DKE :

- **Simple** : $K := E_S(r_a \oplus r_b)$ - pas de PFS
- **AKEP2** : utilise MACs pour authentification + clé dérivée $K := h'_{S'}(r_b)$
- Pas de PFS si S' compromise

Asymétrique avec pré-distribution

Diffie-Hellman

Initialisation : Premier p et générateur $\alpha \in \mathbb{Z}_p^*$ publics

Protocole :

1. $A \to B : \alpha^x \mod p$ (A choisit x secret)
2. $A \leftarrow B : \alpha^y \mod p$ (B choisit y secret)
3. Clé partagée : $K := \alpha^{xy} \mod p$

- A calcule $K := (\alpha^y)^x \mod p$
- B calcule $K := (\alpha^x)^y \mod p$

Sécurité :

- Basée sur le problème Diffie-Hellman (DHP) : impossible de calculer α^{xy} à partir de α^x et α^y.
- Résultat prouvé : DHP $\equiv$ DLP.

Attaque Man-in-the-Middle (MIM) :

Adversaire C intercepte et remplace :

- α^x par $\alpha^{x'}$ vers B
- α^y par $\alpha^{y'}$ vers A
- C établit deux clés : $\alpha^{xy'}$ avec A et $\alpha^{x'y}$ avec B

Propriétés (DH non authentifié) :

- Entity authentication
- Implicit key authentication (vulnérable MIM)
- Key confirmation

Génération de clés symétriques :

Les clés DH ne sont pas bit secure.

Solution : appliquer un MDC (SHA, MD5) à toute la clé K :

$$K_{sym} := \text{SHA-256}(K)$$

Résultat: KAP avec Dynamic Key Establishment

Texte original

KAP Asymétrique avec Pré-Distribution - Diffie-Hellman
Publié en 1976, il s'agit du **précurseur des protocoles asymétriques**.
Il permet à deux entités qui ne se sont jamais rencontrées de construire une **clé partagée** en échangeant des messages sur un **canal non confidentiel**.
Protocole :
Initialisation : Un nb. premier p est généré et un générateur α de $\mathbb{Z}_p^*$, t.q. $\alpha \in \mathbb{Z}_{p-1}$. Les deux nombres sont rendus publiques.

(1) $A \to B : \alpha^x \mod p$; A choisit un secret $x \in \mathbb{Z}_{p-1}$ et envoie la partie publique

(2) $A \leftarrow B : \alpha^y \mod p$; B choisit un secret $y \in \mathbb{Z}_{p-1}$ et envoie la partie publique

A calcule la clé secrète : $K := (\alpha^y)^x \mod p$ et B à son tour : $K := (\alpha^x)^y \mod p$
La **sécurité** de ce schéma réside dans l'impossibilité de trouver $\alpha^{xy} \mod p$ à partir de $\alpha^x \mod p$ et $\alpha^y \mod p$. (**Diffie-Hellman Problem** : DHP).
Résultat prouvé : DHP $\equiv$ DLP.
Diffie-Hellman est **sûr** (autant que DHP) contre des **attaques passives**. En d'autres mots, un adversaire qui se limite à voir passer des messages ne peut pas trouver la clé K.
Ceci n'est cependant plus vrai pour des **attaques actives** ; voyons ce que C peut faire en modifiant les messages :
C échange des clés secrètes avec A et B, respectivement : $\alpha^{xy'} \mod p$ et $\alpha^{x'y} \mod p$ (C contrôle x' et y'). Si C ré-encrypte chaque paquet qu'il reçoit avec la clé publique correspondante, l'attaque se fera de manière transparente pour A et B.
Cette attaque est appelée **Man in the Middle (MIM)** et s'applique à tous les protocoles asymétriques.
Elle est due au **manque d'authentification des clés publiques**, i.e. lorsque A "parle" à B, il doit utiliser la clé publique **authentique** de B.
Caractéristiques de Diffie-Hellman (non authentifié) :

- **Entity Authentication** : NON.
- **Implicit key authentication** : NON (par l'attaque MIM).
- **Key confirmation** : NON (dû au risque de MIM, A ne peut pas être sûr que B possède la clé secrète partagée).

Génération de clés symétriques à partir d'une clé partagée Diffie-Hellman :
Les quantités manipulées dans DH (notamment K) sont de taille 512 - 1024 bits (suivant le nb. premier p utilisé).
Une approche intuitive pour générer des clés symétriques de petite taille (64 - 128 bits) serait de prendre un sous-ensemble de bits de la clé K.
Malheureusement, on peut prouver que les clés DH ne sont pas **bit secure** ce qui signifie que des sous ensembles de bits (notamment les Least Significant Bits) peuvent être calculés avec un effort non proportionnel à l'effort nécessaire à calculer la clé entière.
Pour générer des clés de manière sûre il est conseillé d'**appliquer un MDC** (comme SHA ou MD5) à toute la clé (év. enchaîner l'application des MDCs pour obtenir des clés symétriques successives).
Cette méthode permet d'obtenir un KAP avec Dynamic Key Establishment.

Révision rapide

Diffie-Hellman :

- $K := \alpha^{xy} \mod p$ calculée indépendamment par A et B
- Sûr contre attaques passives (DHP $\equiv$ DLP)
- Vulnérable MIM sans authentification
- Générer clés symétriques : $K_{sym} := \text{SHA}(K)$

Asymétrique avec DKE

Station to Station Protocol (STS)

Diffie-Hellman **authentifié** avec signatures numériques.

Initialisation : Nombre premier p, générateur α publics. A et B ont copies authentiques des clés publiques.

Protocole :

1. $A \to B : \alpha^x \mod p$
2. $A \leftarrow B : \alpha^y \mod p, E_k(S_B(\alpha^x, \alpha^y))$

- B calcule $k := (\alpha^x)^y \mod p$
- B signe et encrypte les parties publiques

3. $A \to B : E_k(S_A(\alpha^y, \alpha^x))$
   - A décrypte, vérifie signature de B
   - A signe et encrypte en inversant l'ordre

Propriétés :

- Entity authentication (mutuelle, par signatures)
- Implicit key authentication (DHP + signatures empêchent MIM)
- Key confirmation (encryption prouve possession de k)
- Explicit key authentication (authentication + key confirmation)
- **Perfect Forward Secrecy** (clé privée signature compromise ne révèle pas clés session passées)

Variante efficace : Remplacer $E_k(S_B(...))$ par $(sig, h_k(sig))$ avec MAC au lieu d'encryption symétrique.

ℹ Texte original

KAP Asymétrique avec DKE - Station to Station Protocol
(Notation) S_A : Signature avec la clé privée de A.
(Initialisation) : (a) On choisit un nb. premier p et un générateur α de $\mathbb{Z}_p^*$, t.q. $\alpha \in \mathbb{Z}_{p-1}$. Les deux nombres sont rendus publiques (et éventuellement associés aux clés publiques des intervenants).

(b) Les intervenants ont accès aux copies authentiques des clés publiques des correspondants. Des certificats peuvent être échangés si besoin dans (2) et (3).

(1) $A \to B : \alpha^x \mod p$; A génère un secret x et envoie la partie pub.

(2) $A \leftarrow B : \alpha^y \mod p, E_k(S_B(\alpha^x, \alpha^y))$; B génère un secret y et calcule la clé : $k := (\alpha^x)^y \mod p$ + signe et encrypte les p.pub.

(3) $A \to B : E_k(S_A(\alpha^y, \alpha^x))$; A décrypte en calculant $k := (\alpha^y)^x \mod p$, teste la signature de B et les parties publiques ; si OK, A signe + encrypte en inversant les parties publiques.

B décrypte et teste la signature de A sur les parties publiques. Si OK => FIN.
Caractéristiques :

- **Entity Authentication** : OUI mutuelle (fournie par les signatures).
- **Implicit key authentication** : OUI, les clés sont protégées par DHP. L'attaque MIM est rendue impossible par les signatures.

- **Key confirmation** : OUI, les deux entités prouvent la possession de la clé en encryptant des quantités avec.
- **Explicit key authentication** : OUI : implicit key authentication + key confirmation.
- **Perfect Forward Secrecy** : OUI. La seule clé à long terme est celle utilisée pour signature/vérification. Si cette clé est compromise, les clés de session antérieures sont protégées par le fait qu'elles ne sont pas explicitement échangées mais plutôt calculées par DH.

Évidemment, dès que la clé de signature est compromise (vol de clé privée), les propriétés énoncées ne sont plus vérifiées pour les échanges ultérieurs.
Le protocole fournit en plus l'**anonymat** car l'identité des parties est protégée par k.
Variante : Dans (2), calculer $sig := S_B(\alpha^x, \alpha^y)$, et envoyer : $(sig, h_k(sig))$ plutôt que $E_k(S_B(\alpha^x, \alpha^y))$. Pareil pour (3) en observant les asymétries du protocole.
Solution plus efficace car elle fait intervenir un **MAC** plutôt qu'un cryptage symétrique. Algorithme robuste et efficace choisi comme support de base pour la génération de clés dans **IPv6**.

Révision rapide

Station to Station (STS) :

- DH + signatures numériques
- PFS : clés session passées protégées
- Explicit key authentication
- Utilisé dans IPv6

Protocoles OTR et Signal

Off-The-Record (OTR)

Protocole (2004) pour messagerie instantanée avec **répudiabilité**.

Technique SIGMA (SIGn-and-MAC) :

- Signatures DH + authentification éphémère via MAC
- Key Derivation Function (KDF) génère deux clés : K_e (encryption AES-CTR) et K_m (MAC)
- Changement de clés à chaque conversation
- **Révélation des clés MAC** précédentes pour garantir répudiabilité

Protocole simplifié :

1. $A \rightarrow B : \alpha^x \mod p$
2. $A \leftarrow B : \alpha^y \mod p, S_B(\alpha^x, \alpha^y), \text{MAC}_{K_m}(B)$
   - B calcule $k := (\alpha^x)^y \mod p$
   - $(K_m, K_e) := \text{KDF}(k)$
3. $A \rightarrow B : S_A(\alpha^y, \alpha^x), \text{MAC}_{K_m}(A)$

Messages chiffrés avec K_e.

Signal Protocol

Évolution d'OTR pour réseaux sociaux (WhatsApp, Facebook Messenger).

Caractéristiques :

- Clés asymétriques et symétriques éphémères
- DH sur courbes elliptiques
- PFS
- Future Secrecy
- Repudiability

> Texte original
>
> **Protocole Off-The-Record (OTR)**
> Protocole conçu en 2004 dans le but d'offrir des services d'authentification et de confidentialité dans les échanges des messages (instant messaging) en préservant le caractère "**répudiable**" d'une conversation "off the record".
> Le protocole satisfait également les propriétés de **PFS** et **Future Secrecy** en cas de compromis des clés long terme.
> Il reprend les mêmes principes que le protocole Station-to-Station en rajoutant aux signatures des paramètres DH une **authentification éphémère via un MAC**. Cette technique double est appelée **SIGMA** (SIGn-and-MAC).
> Il utilise une **fonction de dérivation de clés** (Key Derivation Function ou KDF) pour générer une clé d'encryption (K_e) préservant la confidentialité des messages avec AES CTR-mode et une clé MAC (K_m) garantissant l'authenticité d'origine de ceux-ci.
> Chaque conversation implique un **changement de clés** (nouvel échange de paramètres DH) avec en plus un **échange en clair des clés MAC** (K_m) utilisées dans l'échange précédent pour garantir la répudiabilité !
> **Échanges schématiques du protocole OTR :**
>
> (1) $A \rightarrow B : \alpha^x \mod p$; A génère un secret x et envoie la partie pub.

(2) $A \leftarrow B : \alpha^y \mod p, S_B(\alpha^x, \alpha^y), \text{MAC}_{K_m}(B)$

B génère un secret y, calcule la clé de session $k := (\alpha^x)^y \mod p$ et signe les parties publiques DH. Il génère ensuite les clés K_e et K_m via la KDF : $(K_m, K_e) := \text{KDF}(k)$

(3) $A \rightarrow B : S_A(\alpha^y, \alpha^x), \text{MAC}_{K_m}(A)$: A fait de même

Les messages sont ensuite chiffrés avec la clé K_e
Il existe de nombreuses évolutions du protocole original OTR ayant permis d'adresser des vulnérabilités et de rendre le protocole plus efficace.
Le protocole Signal
Le protocole Signal est une évolution du protocole OTR qui cible la protection des échanges des messages dans les **réseaux sociaux**. Il utilise également des clés asymétriques et symétriques éphémères pour assurer la **PFS**, la **Future Secrecy** et la **repudiability** avec des calculs DH sur des courbes elliptiques.
Signal est utilisé pour protéger les plateformes de messagerie telles que **Whatsapp** et **Facebook Messenger** entre autres.

Révision rapide

OTR/Signal :

- SIGMA : signature + MAC
- KDF : génère K_e (chiffrement) et K_m (MAC)
- Révèle anciennes clés MAC $\rightarrow$ répudiabilité
- PFS, Future Secrecy
- Utilisé : WhatsApp, Facebook Messenger

Secure Remote Password (SRP)

Protocole KAP asymétrique basé sur **mot de passe**, résistant aux attaques dictionnaire.

Initialisation :

- $m := 2p + 1$ (safe prime), α générateur de $\mathbb{Z}_p^*$
- P : password de A, $x := H(P)$ avec H une CRHF
- B stocke le **vérificateur** : $v := \alpha^x \mod m$ (pas le password!)

Protocole :

1. $A \rightarrow B : \gamma := \alpha^r \mod m$ (A génère r secret)

2. $A \leftarrow B : \delta := (v + \alpha^t) \mod m, u$ (B génère t, u aléatoires)
3. A calcule $k := (\delta - v)^{r+ux} \mod m$
4. B calcule $k := (\gamma v^u)^t \mod m$
5. Key confirmation

Propriétés :

- Protège passwords des attaques dictionnaire
- **Verifier-based** : B ne stocke pas passwords
- Toutes propriétés KEP
- Inclus dans SSL/TLS, EAP

Texte original

KAP Asymétrique avec DKE - Secure Remote Password protocol

(a) Soit m un safe prime avec $m := 2p + 1$ et p premier
(b) Soit α un générateur de $\mathbb{Z}_p^*$, t.q. $\alpha \in \mathbb{Z}_{p-1}$
(c) Soit P le password de A et $x := H(P)$ avec H une CRHF.
(d) B garde dans sa base des mots de passe le **vérificateur** $v := \alpha^x \mod m$.

(1) $A \rightarrow B : \gamma := \alpha^r \mod m$; A génère un nombre aléatoire secret r

(2) $A \leftarrow B : \delta := (v + \alpha^t) \mod m, u$; B génère un nombre aléatoire secret t et un deuxième nombre aléatoire u

A calcule la clé symétrique : $k := (\delta - v)^{r+ux} \mod m$
B calcule la clé symétrique : $k := (\gamma v^u)^t \mod m$
A et B prouvent la connaissance de k (key confirmation) lors d'un échange ultérieur.

- SRP **protège les mots de passe** des attaques dictionnaire.
- B ne stocke pas les passwords mais des **valeurs de vérification** (verifier-based).
- SRP satisfait également **toutes les propriétés propres aux KEP** et est inclus dans des nombreux standards (SSL/TLS, EAP, etc.).

Révision rapide

SRP :

- KAP basé mot de passe
- B stocke vérificateur $v := \alpha^x$ (pas password)
- Résiste attaques dictionnaire
- Toutes propriétés KEP

Attaques sur DH et PFS

Attaque Logjam (2015) :

Attaque active permettant :

1. **Downgrade** : Man-in-the-Middle force utilisation de groupe DH 512 bits
2. **Calcul de logarithmes discrets** avec Number Field Sieve :
   - Pré-calcul d'une semaine pour un premier p fixé
   - Calcul individuel en ~1 minute après pré-calcul
3. **Réutilisation du pré-calcul** : Beaucoup de serveurs utilisent le même p

Conséquence :

Acteurs avec ressources étatiques peuvent compromettre PFS sur groupes 1024 bits répandus.

Solutions :

- Utiliser groupes $\geq$ 2048 bits
- Diversifier les premiers p utilisés

Texte original

Attaques Récentes sur Diffie-Hellman et la PFS
En 2015 un groupe de chercheurs a publié une série d'attaques sur le protocole TLS/SSL permettant de :

- Effectuer un **downgrade** via une attaque active appelée **Logjam** moyennant laquelle un man-in-the-middle réussit à diminuer à **512 bits** la taille du groupe Diffie-Hellman sur lequel s'effectue l'établissement de la clé secrète partagée.
- Calculer ensuite les **logarithmes discrets** de $\alpha^x \mod p$ et de $\alpha^y \mod p$ avec la technique **Number Field Sieve**.
- À partir d'un groupe basé sur un nombre premier p fixé, ils effectuent une **phase de pré-calcul** d'une durée approximative d'**une semaine**.
- Une fois cette phase initiale terminée, les calculs des logarithmes individuels ne prennent qu'**une minute** !
- Une constatation statistique montre qu'un pourcentage significatif des serveurs se basent sur le **même groupe** (même premier p) ce qui permet d'utiliser la même phase de pré-calcul pour compromettre plusieurs serveurs.

- Une des conclusions de cette recherche est que des **acteurs majeurs avec des ressources étatiques** seraient capables à ce jour de démonter la PFS lorsque celle-ci est basée sur des groupes (très répandus à ce jour…) de **1024 bits**.

Révision rapide

Logjam (2015) :

- Downgrade → DH 512 bits
- Pré-calcul (1 semaine) + calcul individuel (1 min)
- Réutilisation si même p
- États peuvent casser PFS sur 1024 bits

KTP

Symétrique

Cas Trivial

Initialisation : A et B partagent clé long terme S

Protocole :

1. $A \to B : E_S(r_a)$
2. Clé de session : $K := r_a$

Propriétés :

- Entity authentication
- Implicit key authentication
- Key confirmation (amélioration : $E_S(B, r_a)$)
- Perfect Forward Secrecy

Variante avec timestamp : $A \to B : E_S(B, t_a, r_a)$ (nécessite horloges synchronisées)

Shamir's No-key Protocol

Équivalent de DH en transport de clé.

Initialisation : Nombre premier p public, A et B génèrent secrets $a, b \in \mathbb{Z}_{p-1}$ avec $\gcd(a, p-1) = 1$ et $\gcd(b, p-1) = 1$

Protocole :

1. $A \to B : K^a \mod p$ (A choisit clé K et cache avec ^{a})
2. $A \leftarrow B : (K^a)^b \mod p$ (B exponentie avec b)
3. $A \to B : (K^{ab})^{a^{-1}} \mod p = K^b \mod p$ (A défait ^{a})
4. B calcule K en exponenti ant avec $b^{-1} \mod (p-1)$

Problème : Vulnérable Man-in-the-Middle (comme DH)

Texte original

Key Transport Protocol Symétrique - Cas trivial
(Init.) A et B partagent une clé symétrique long terme S

(1) $A \to B : E_S(r_a)$; A génère un nb. aléatoire et l'encrypte avec k

La clé de session utilisée par les deux entités est $K := r_a$.
Propriétés :

- **Entity Authentication** : NON.
- **Implicit Key Authentication** : OUI (seul A et B ont accès à la clé).
- **Key Confirmation** : NON. B ne peut pas être sûr que A possède la clé car r_a est un nombre aléatoire. En rajoutant de la redondance (p.ex. l'identité de B), B peut obtenir key confirmation unilatérale (et donc, explicit key authentication) :

(1)' : $A \to B : E_s(B, r_a)$

- **Perfect Forward Secrecy** : NON.

Si, de plus, B ne peut pas juger l'actualité (freshness) de (1) à partir du seul r_a, il peut demander à A d'inclure un timestamp à condition d'avoir des horloges synchronisés :
(1)'' : $A \to B : E_s(B, t_a, r_a)$
KTP Symétrique : Shamir's No-key Protocol
Rappel Théorie des nombres : Si p premier et $r \equiv t \mod p-1$ alors $a^r \equiv a^t \mod p$ $\forall a \in \mathbb{Z}$ et donc : $r \cdot r^{-1} \equiv 1 \mod p-1$ implique $a^{r \cdot r^{-1}} \equiv a \mod p$.
(Init.) (a) Choisir et publier un nb. premier p pour lequel il est difficile (par DLP) de calculer les logarithmes discrets dans $\mathbb{Z}_p$.

(b) A (resp. B) génère un nombre secret a (resp. b), t.q $\{a, b\} \in \mathbb{Z}_{p-1}$ et $\gcd(a, p-1) = 1$ et $\gcd(b, p-1) = 1$ (pour que les inverses existent).

(c) Pour la suite, A pré-calcule $a^{-1} \mod p-1$ et B pré-calcule $b^{-1} \mod p-1$

(1) : $A \to B : K^a \mod p$; A choisit une clé $K \in \mathbb{Z}_p$ et la cache avec ^{a}

(2) : $A \leftarrow B : (K^a)^b \mod p$; B exponentie à son tour avec b

(3) : $A \to B : (K^{ab})^{a^{-1}} \mod p$; A défait l'exponentiation avec $a^{-1} \mod p-1$; mais la clé reste protégée par ^{b}

B n'a plus qu'à calculer K en exponentiant avec $b^{-1} \mod p-1$.
Ce protocole est l'équivalent de Diffie-Hellman en Key Transport (dans DH la clé n'est pas transportée mais calculée bilatéralement). Il souffre donc des mêmes problèmes (notamment Man in the Middle) que ce dernier.

Révision rapide

KTP symétrique :

- Trivial : $K := r_a$ avec $E_S(r_a)$
- Shamir : transport via exponentiations successives
- Pas de PFS

Asymétrique

Needham-Schroeder

Initialisation : A et B ont copies authentiques clés publiques mutuelles

Protocole :

1. $A \to B : E_{pub_B}(k_1, A)$
2. $A \leftarrow B : E_{pub_A}(k_1, k_2)$
3. $A \to B : E_{pub_B}(k_2)$
4. Clé de session : $K := H(k_1, k_2)$

Propriétés :

- Entity authentication + implicit key authentication + key confirmation
- Perfect Forward Secrecy (clés entièrement déterminées par quantités échangées)

Texte original

Key Transport Protocol Asymétrique - Needham-Schroeder Public Key Protocol
(Notation) : $E_{pub_E}(X)$ signifie encrypter avec la clé publique de l'entité E.
(Init) : A et B possèdent une copie authentique (év. un certificat) de la clé publique de l'autre.

(1) $A \to B : E_{pub_B}(k_1, A)$; A génère un nb. aléatoire k_1 + A + Encrypt

(2) $A \leftarrow B : E_{pub_A}(k_1, k_2)$; B idem pour k_2 + concat avec k_1 + Encrypt

(3) $A \to B : E_{pub_B}(k_2)$; A vérifie si k_1 coïncide, si oui, encrypt k_2 ; B vérifie si k_2 coïncide avec (2)

La clé est générée à l'aide d'une fonction de hachage cryptographique : $K := H(k_1, k_2)$
Caractéristiques :

- **Entity Authentication + implicit key authentication + key confirmation** : OUI.
- **Perfect forward secrecy** : NON : Les clés sont entièrement déterminées par les quantités échangées.

Un protocole semblable (seul (3) change) peut être utilisé pour l'authentification d'entités.

Révision rapide

Needham-Schroeder asymétrique :

- $K := H(k_1, k_2)$ avec échanges encryptés
- Authentification complète
- Pas de PFS

Mixte

Encrypted Key Exchange (EKE)

Protocole **mixte** (symétrique + asymétrique) résistant aux attaques dictionnaire.

Initialisation : A et B partagent password p

Protocole :

1. $A \rightarrow B : A, E_p(pub_A)$ (A génère paire clés, envoie publique encryptée)
2. $A \leftarrow B : E_p(E_{pub_A}(k))$ (B génère clé session k, double encryption)
3. $A \rightarrow B : E_k(r_a)$ (Key confirmation)
4. $A \leftarrow B : E_k(r_a, r_b)$
5. $A \rightarrow B : E_k(r_b)$

Avantages :

- Robuste même si password p faible
- Eve ne peut pas deviner sans "casser" aussi l'algorithme asymétrique

Propriétés :

- Entity authentication + implicit + confirmation
- Perfect Forward Secrecy **si** $pub_A/priv_A$ régénérée à chaque fois
- Pas de PFS si clés longue durée

Texte original

KTP mixte : Encrypted Key Exchange (EKE)
Ce protocole fait intervenir des schémas **symétriques et asymétriques** afin de minimiser le risque de cryptanalyse par attaque dictionnaire inhérents aux systèmes symétriques.
(Init.) : A et B partagent un secret symétrique p (password).

(1) $A \rightarrow B : A, E_p(pub_A)$; A génère une paire de clés pub/priv. et envoie la partie publique à B encrypté avec p.

(2) $A \leftarrow B : E_p(E_{pub_A}(k))$; B génère une clé de session k et l'envoie encryptée.

(3) $A \rightarrow B : E_k(r_a)$; A génère un nb. aléatoire et l'envoie encrypté avec k.

(4) $A \leftarrow B : E_k(r_a, r_b)$; B génère r_b et l'envoie avec r_a crypté avec k.

(5) $A \rightarrow B : E_k(r_b)$; Confirmation de la part de A. Si r_b = OK => FIN.

(6) et (2) sont responsables du **key transport** ; (3) à (5) du **key confirmation**.

Ce protocole est **robuste même si le password** p partagé entre A et B est de mauvaise qualité. En effet, Eve ne peut pas essayer de deviner sans "casser" aussi l'algorithme asymétrique.
Propriétés :

- **Entity Authentication + implicit key authentication + key confirmation** : OUI.
- **Perfect forward secrecy** : OUI si la paire $pub_A/priv_A$ est régénérée à chaque instance du protocole. NON si $pub_A/priv_A$ est une clé de longue durée.

Révision rapide

EKE (mixte) :

- Password + crypto asymétrique
- Robuste même si password faible
- PFS si clés régénérées chaque fois

Symétrique avec Key Distribution Center (KDC)

Needham-Schroeder Symétrique

Protocole avec **Key Distribution Center** (KDC).

Initialisation : A partage K_{AT} avec T (KDC), B partage K_{BT} avec T

Protocole :

1. $A \to T : A, B, r_a$
2. $A \leftarrow T : E_{K_{AT}}(r_a, B, k_{AB}, E_{K_{BT}}(k_{AB}, A))$
3. $A \to B : E_{K_{BT}}(k_{AB}, A)$
4. $A \leftarrow B : E_{k_{AB}}(r_b)$
5. $A \to B : E_{k_{AB}}(r_b - 1)$

Propriétés :

- Entity authentication A auprès de B
- Entity authentication B auprès de A (A n'a jamais vu r_b)
- Implicit key authentication
- Key Confirmation (seul B sait que A possède la clé)
- Perfect Forward Secrecy

Vulnérabilités :

- **Replay attacks** : A peut rejouer (3) sans contrôle de B
- **Known-key attack** : Si ancienne clé k compromise, adversaire peut la faire accepter par B

Solutions :

- **key confirmation et entity authentication mutuelles :**

  Remplacer 3. et 4. par :

3. $A \rightarrow B : E_{k_{AB}}(r'_a), E_{K_{BT}}(k_{AB}, A)$
4. $A \leftarrow B : E_{k_{AB}}(r'_a - 1, r_b)$

- **Actualité des échanges**

  Ajouter timestamp dans 3. : $E_{K_{BT}}(k_{AB}, A, t)$

Texte original

KTP symétrique avec Key Distribution Center - Needham-Schroeder Symétrique
(Notation) : On appelle T, le **Key Distribution Center**.
(Init.) : A et T partagent la clé symétrique K_{AT}. B et T partagent K_{BT}.

(1) $A \rightarrow T : A, B, r_a$; A génère un nb. aléatoire r_a et l'envoie à T avec les ident.

(2) $A \leftarrow T : E_{K_{AT}}(r_a, B, k_{AB}, E_{K_{BT}}(k_{AB}, A))$; T génère k_{AB} et l'envoie encryptée.

(3) $A \rightarrow B : E_{K_{BT}}(k_{AB}, A)$; A forwarde le paquet à B.

(4) $A \leftarrow B : E_{k_{AB}}(r_b)$; confirmation de B en utilisant k_{AB} et un nb. aléatoire r_b

(5) $A \rightarrow B : E_{k_{AB}}(r_b - 1)$; confirmation de A

Propriétés :

- **Entity Authentication** :
  - A auprès de B : OUI.
  - B auprès de A : NON : A n'a jamais vu r_b (il pourrait s'agir de $E_{k'}(r'_b)$).
- **Implicit Key Authentication** : OUI (les clés sont toujours protégées par K_{AT} et K_{BT}). Cependant, en cas de known-key attack, ceci n'est plus vérifié pour B.
- **Key Confirmation** : Seul B obtient l'assurance que A possède la clé à cause de la faille décrite dans entity authentication.
- **Perfect Forward Secrecy** : NON. Si une des deux clés K_{AT} ou K_{BT} est compromise, les clés de session k deviennent immédiatement visibles.

Solution pour obtenir key confirmation et entity authentication mutuelles :
Remplacer (3) et (4) par :
(3') $A \rightarrow B : E_{k_{AB}}(r'_a), E_{K_{BT}}(k_{AB}, A)$
(4') $A \leftarrow B : E_{k_{AB}}(r'_a - 1, r_b)$
Pour autant que les r_i soient soigneusement contrôlés par les intervenants.
Cependant : attention aux **reflection attacks** !
Problème : A peut rejouer (3) autant de fois qu'il le souhaite, sans aucun contrôle de la part de B. Ce problème s'aggrave si une vieille clé k est compromise :

Vulnérable au known-key attack : Si une clé de session k déjà utilisée est obtenue par un adversaire C, il peut sans difficulté la faire accepter par B en rejouant (3) et en calculant le challenge envoyé par B dans (5). Dans ce cas, les propriétés entity authentication, implicit key authentication et key confirmation de A auprès de B sont aussi compromises.
Solution : Rajouter un **timestamp** dans (3) témoignant de l'actualité des échanges : (3'') $A \to B : E_{K_{BT}}(k_{AB}, A, t)$ (c'est la solution adoptée par **Kerberos**)

Révision rapide

Needham-Schroeder symétrique :

- KDC génère et distribue k_{AB}
- Vulnérable replay et known-key attacks
- Solution : ajouter timestamp
- Base de Kerberos

Kerberos

Protocole d'**authentification et distribution de clés** basé sur Needham-Schroeder avec corrections.

Architecture :

- **Authentication Server (AS)** : Émet tickets pour TGS
- **Ticket Granting Server (TGS)** : Émet tickets pour services
- **Tickets** : Structures encryptées contenant clés de session

Protocole simplifié :

Phase 1 : Demande TGT (Ticket Granting Ticket)

1. $A \to AS : A, TGS, r_a$
2. $A \leftarrow AS : E_{K_A}(k_{AT}, r_a), Ticket_{AT} := E_{K_T}(A, TGS, t_1, t_2, k_{AT})$

Phase 2 : Demande ticket pour service B

3. $A \to TGS : Authenticator_{AT} := E_{k_{AT}}(A, t), Ticket_{AT}, B, r'_a$
4. $A \leftarrow TGS : E_{k_{AT}}(k_{AB}, r'_a), Ticket_{AB} := E_{K_B}(A, B, t_1, t_2, k_{AB})$

Phase 3 : Authentification auprès de B

5. $A \to B : Authenticator_{AB} := E_{k_{AB}}(A, t), Ticket_{AB}, r''_a, [request]$

6. $A \leftarrow B : E_{k_{AB}}(r''_a), [response]$

Propriétés :

- Entity authentication (toutes entités)
- Implicit key authentication
- Key confirmation partielle (pas entre A et AS)
- Perfect Forward Secrecy

Vulnérabilités :

- **Password guessing attacks** sur $E_{K_A}(k_{AT}, r_a)$ (Solution : pré-authentification)
- **Replay attacks** si r_a mal contrôlés
- Nécessite synchronisation d'horloges

Texte original

KTP symétrique avec Key Distribution Center - Kerberos
Kerberos est un protocole permettant l'**authentification d'entités** et la **distribution de clés** à l'intérieur dans un réseau d'utilisateurs.
À l'origine, Kerberos était conçu comme solution de remplacement pour remédier aux problèmes d'insécurité (authentification faible, transactions en clair, etc.) propres aux environnements UNIX.
Kerberos fut créé à MIT comme partie intégrante du projet ATHENA.
Il est basé sur le protocole de Needham-Schroeder symétrique avec notamment la **correction de quelques failles** du protocole et l'inclusion de **timestamps**.
Les trois premières versions étaient instables. La **version 4** a eu un succès considérable aussi bien dans les environnements industriel qu'académique et reste prédominante. La **version 5**, bien qu'étant plus sûre et mieux structurée, est plus complexe et moins performante, ce qui a ralenti son déploiement.
Kerberos définit également un mode de collaboration entre domaines appartenant à des autorités administratives distinctes (les **realms**). Ceci permet à des utilisateurs d'un domaine d'utiliser des ressources d'un autre domaine "sans sortir" de l'environnement sécurisé de Kerberos.
Pour des transactions inter-realm, la cryptographie symétrique constitue un obstacle significatif car nécessite des canaux confidentiels pour la pré-distribution des clés.
Kerberos Version 5
(Notation) : - A et B veulent établir une transaction sécurisée ; dans l'environnement Kerberos, il s'agit normalement d'un **client** et d'un **serveur** fournissant des services. - Le KDC de Kerberos est subdivisé en deux entités fonctionnelles : l'**Authentication Server (AS)** et le **Ticket Granting Server (TGS)**. Les deux accèdent à la BdD passwords. - Les $r_a^{(n)}$ sont des nbs. aléatoires, t est un timestamp, t_1 et t_2 indiquent une fenêtre de validité de temps.

(Initialisation) : A et B partagent une clé secrète avec AS, soient : K_A et K_B (pour les clients, il s'agit d'une OWF du password). TGS a également une clé secrète K_T.

(1) $A \rightarrow AS : A, TGS, r_a$

(2) $A \leftarrow AS : E_{K_A}(k_{AT}, r_a), Ticket_{AT} := E_{K_T}(A, TGS, t_1, t_2, k_{AT})$; AS génère k_{AT}

(3) $A \rightarrow TGS : Authenticator_{AT} := E_{k_{AT}}(A, t), Ticket_{AT}, B, r'_a$

(4) $A \leftarrow TGS : E_{k_{AT}}(k_{AB}, r'_a), Ticket_{AB} := E_{K_B}(A, B, t_1, t_2, k_{AB})$; TGS génère k_{AB}

(5) $A \rightarrow B : Authenticator_{AB} := E_{k_{AB}}(A, t), Ticket_{AB}, r''_a, [request]$

(6) $A \leftarrow B : E_{k_{AB}}(r''_a), [response]$; [request] et [response] év. cryptés avec k_{AB}

(7) • (2) : Demande de ticket pour TGS

(8) • (4) : Demande de ticket pour B

(9) • (6) : Authentification et établissement de clé entre A et B.

Caractéristiques de Kerberos
Propriétés :

- **Entity Authentication** : OUI, de toutes les entités impliquées.
- **Implicit key authentication** : OUI : toutes les clés générées sont protégées par des clés partagées entre le AS et tous les participants.
- **Key confirmation** :
  - Entre A et AS : NON : AS n'a pas de preuve que A possède la clé K_A.
  - Entre A et TGS : OUI pour k_{AT} (des quantités redondantes encryptées avec k_{AT} sont échangées entre A et TGS) ; NON pour k_{AB} (TGS n'a pas de preuve de la part de A)
  - Entre A et B : OUI : échange des quantités redondantes encryptées avec k_{AB}.
- **Perfect forward secrecy** : NON : Toutes les clés sont explicitement transférées.

Problèmes :

- Les clés initiales (comme K_A) dépendent (directement) des passwords choisis par les utilisateurs. Ceci rend le protocole vulnérable à des vols de password ou à des :
  - **Password guessing attacks** : $E_{K_A}(k_{AT}, r_a)$ dans (2) aide à casser le password de A. **Solution :** Pré-authentification dans (1) : $E_{K_A}(t)$ avec t = timestamp (optionnelle dans v5).
- La fenêtre de validité d'un ticket peut conduire à des **replay attacks** si les $r_a^{(n)}$ ne sont pas correctement contrôlés par les intervenants.
- La **synchronisation d'horloges** est nécessaire ! Ceci n'est pas toujours facile dans des environnements hétérogènes.

Révision rapide

Kerberos :

- AS émet TGT, TGS émet tickets service
- Tickets contiennent clés de session
- Authentification via authenticators
- Vulnérable : password guessing, replay
- Solution : pré-authentification, timestamps

SSL/TLS

SSL/TLS: Secure Socket Layer / Transport Layer Security

Protocole de sécurisation entre couche transport (TCP) et application.

Services fournis :

- Confidentialité, intégrité, authentification du flot
- Identification serveur (client optionnelle)

Algorithmes utilisés :

- **Cryptographie publique** (RSA, DH, DSA) : échange clés
- **MACs** : authentification flot
- **Cryptographie symétrique** (DES, AES, IDEA) : encryption flot

Propriétés :

- Entity authentication (serveur + client optionnel via certificats)
- Implicit key authentication
- Key confirmation
- Perfect Forward Secrecy : dépend du protocole d'échange (DH → oui, RSA → non)

Remarques

- Les clés secrètes TLS sont dérivées par hachage à partir de valeurs aléatoires et du *pre_master_secret*.
- SSL/TLS est le standard de facto de la sécurité web (HTTPS).
- La confiance repose sur des certificats racine intégrés dans les navigateurs.
- Les failles majeures proviennent de l'aléa, des implémentations et des fonctions de hachage.
- Attaques notables : renégociation (2009), Heartbleed (2014).

Architecture SSL/TLS

Trois composants :

1. **SSL Record Protocol** : Encapsulation au-dessus de TCP (fragmentation + compression + encryption)
2. **SSL Handshake Protocol** : Authentification + négociation paramètres
3. **SSL State Machine** : Variables d'état session et connexion

Texte original

Secure Socket Layer (SSL) / Transport Level Security (TLS)
Se situe entre la couche transport (TCP) et les protocoles de la couche application (non seulement HTTP mais également SMTP, FTP, etc. !)
Il s'agit d'un **Meta Protocole** d'établissement de clés hautement paramétrable permettant des nombreux modes de fonctionnement et des options de négociation.
Offre des services de **confidentialité**, **intégrité**, **authentification du flot de données**, et **identification du serveur** (et accessoirement du client)
Utilise les familles d'algorithmes suivants :

- **Cryptographie publique** (RSA, Diffie-Hellmann, DSA, etc.) pour l'échange de clés symétriques
- **MACs** pour l'authentification du flot de données
- **Cryptographie symétrique** (DES, IDEA, AES, etc.) pour l'encryption du flot de données

L'intervention des **CAs** pour certifier l'association entre entités et clés publiques est vivement recommandée… mais pas indispensable !
Propriétés : - **Entity authentication** par certificats (serveur et client optionnelle) - **Implicit Key Authentication** et **Key Confirmation** sont garanties - La **Perfect Forward Secrecy** dépend du protocole choisi pour l'échange de clés.
SSL/TLS Aperçu
SSL est une "mini-pile" de protocoles avec des fonctionnalités des couches session, présentation et application.
SSL est constitué de trois blocs fondamentaux :

- **SSL record protocol** permettant l'encapsulation des protocoles de plus haut niveau au-dessus de TCP (fragmentation + compression + encryption)
- **SSL handshake protocol** chargé de l'authentification des intervenants et de la négociation des paramètres d'encryption
- **SSL state machine**. Contrairement à HTTP, SSL est un protocole **à états** (stateful), il nécessite, donc, un ensemble de variables qui déterminent l'état d'une session et d'une connexion

SSL Handshake Protocol

Diagram

Phase 1 : Hello

- **Client Hello** : Version, random, session ID, algorithmes acceptés
- **Server Hello** : Version, random, session ID, algorithmes sélectionnés
- **Server Certificate** (optionnel) : Certificat serveur + chemin CA
- **Server Key Exchange** (optionnel) : Informations clé publique serveur
- **Certificate Request** (optionnel) : Demande certificat client

Phase 2 : Authentification client et échange de clé

- **Client Certificate** (optionnel) : Certificat client + chemin CA
- **Client Key Exchange** : Génère `pre_master_secret`, envoie encrypté avec clé publique serveur
- **Certificate Verify** (optionnel) : Vérification explicite certificat client

Phase 3 : Finalisation

- **Finish** (client) : Premier message protégé avec paramètres négociés
- **Finish** (serveur) : Idem côté serveur

Phase 4 : Application

- Données protégées avec clés dérivées

Génération des Clés SSL/TLS

Dérivation en cascade :

$$master\_secret = MD5(pre\_master\_secret + SHA('A' + pre\_master\_secret + ClientRandom + ServerRando$$
$$+ MD5(pre\_master\_secret + SHA('BB' + ...)) + ...$$

$$key\_block = MD5(master\_secret + SHA('A' + master\_secret + ServerRandom + ClientRandom)) + ...$$

Partition du key_block :

- client_write_MAC_secret[hash_size]
- server_write_MAC_secret[hash_size]
- client_write_key[key_material]
- server_write_key[key_material]
- client_write_IV[IV_size]
- server_write_IV[IV_size]

Texte original

SSL/TLS Handshake Protocol Simplifié
[Diagramme du handshake avec 4 phases : Hello, Key Exchange, Finish, Application Data]
SSL/TLS : Generation de clés

```
master_secret =
 MD5(pre_master_secret + SHA('A' + pre_master_secret +
 ClientHello.random + ServerHello.random)) +
 MD5(pre_master_secret + SHA('BB' + pre_master_secret +
 ClientHello.random + ServerHello.random)) +
 MD5(pre_master_secret + SHA('CCC' + pre_master_secret +
 ClientHello.random + ServerHello.random));

key_block =
 MD5(master_secret + SHA('A' + master_secret +
 ServerHello.random +
 ClientHello.random)) +
 MD5(master_secret + SHA('BB' + master_secret +
 ServerHello.random +
 ClientHello.random)) +
 MD5(master_secret + SHA('CCC' + master_secret +
 ServerHello.random +
 ClientHello.random)) + [...];

 until enough output has been generated. Then the key_block is
 partitioned as follows:

 client_write_MAC_secret[CipherSpec.hash_size]
 server_write_MAC_secret[CipherSpec.hash_size]
 client_write_key[CipherSpec.key_material]
 server_write_key[CipherSpec.key_material]
 client_write_IV[CipherSpec.IV_size] /* non-export ciphers */
 server_write_IV[CipherSpec.IV_size] /* non-export ciphers */
```

SSL/TLS : Remarques Finales

- Les clés secrètes sont le résultat de l'application de fonctions de hachage (MD5, SHA) sur les random numbers des enregistrement Hello et le `pre_master_secret`
- TLS/SSL est devenu le **standard de facto** pour la sécurité sur le web (à la base de **https**)
- Les clients SSL (Explorer, Firefox, Opera, Chrome, etc.) contiennent "hard-coded" des certificats correspondant à quelques entités de certification racine (Verisign, Thawte, Microsoft, RSA, etc.) permettant de vérifier les certificats présentés par certains serveurs mais SSL est conçu pour s'appuyer sur un **réseau global de certification** pour le moment inexistant.
- Les **failles de sécurité** les plus courantes de SSL concernent la génération aléatoire des clés ainsi que les défauts d'implantation les plus courants : buffer overflows, sql injection, etc. La faiblesse des fonctions de hachage (MD5, SHA) est aussi un facteur à risque.
- En Novembre 2009, on a découvert une attaque permettant à un Man in The Middle d'injecter du contenu (chosen plaintext) dans un flot authentique suite à une **renégociation** des paramètres prévue dans le protocole. Il s'agit d'une faille dans le protocole qui a nécessité un patch dans toutes les implantation.
- La faille **heartbleed** basée sur un buffer overflow a sérieusement troublé la communauté Internet lors de sa découverte en Avril 2014.

Révision rapide

SSL/TLS :

- Meta-protocole entre TCP et application
- Handshake : négociation + authentification
- Clés dérivées : `master_secret` → `key_block`
- Standard HTTPS
- Failles : génération aléatoire, heartbleed, renégociation

Remarques Finales sur les KEP

Avant de choisir un KEP :

1. **Définir objectifs** : confidentialité, authentification, non-répudiation
2. **Définir niveau de sécurité** : key confirmation, PFS, future secrecy
3. **Établir contraintes** : utilisateurs, machines, réseau, attaquants

Bonnes pratiques :

- Choisir solution prouvée et robuste
- Éviter d'inventer "from scratch"
- Vérifier propriétés satisfaites

Vérification des protocoles :

Deux approches complémentaires :

- **Analyse pratique** : "Sur papier" et "sur machine"
  - Contrôle nombres aléatoires (reflection attacks)
  - Redondance quantités encryptées/signées
  - Pièges classiques
- **Analyse formelle** : Logiques dédiées (BAN logic, etc.)

Texte original

Key Establishment Protocols : Remarques Finales
Les protocoles d'établissement de clés constituent une **pierre angulaire** de toute solution de sécurité. Avant de choisir (concevoir) un KEP, il est, donc, indispensable de :

- **Définir les objectifs** (confidentialité, authentification d'entités/données, non-répudiation, etc.)
- **Définir le niveau de sécurité souhaité** en fonction des propriétés étudiées (key confirmation, perfect forward secrecy, etc.)
- **Établir une liste des contraintes** liées à l'environnement (utilisateurs, machines, réseau, attaquants potentiels, etc.)

En fonction de ces critères nous pouvons :

- **Choisir une solution prouvée et robuste** (mieux qu'en inventer une from scratch !).
- **Vérifier que les objectifs sont atteints** et les propriétés satisfaites.

La **vérification des protocoles** est un processus complexe et délicat, de plus, les solutions publiées ne sont pas toujours correctes. Deux approches sont possibles (et nécessaires) :

- **L'analyse pratique**. Analyser les failles du protocole "sur papier" et "sur machine" en tenant compte des pièges classiques : contrôle des nbs. aléatoires pour éviter des reflection attacks, redondance des quantités encryptées/signées, etc.

- **L'analyse formelle** avec des logiques spécialement conçues à cet effet (comme la **logique BAN**)

Révision rapide

KEP - Bonnes pratiques :

1. Définir objectifs et contraintes
2. Choisir solution prouvée
3. Vérifier propriétés (pratique + formelle)
4. Éviter pièges : reflection, redondance, contrôle aléas