UNIVERSITÉ DE SHERBROOKE

IFT814
Cryptographie

Devoir 4 : Protocole ‘’handshake” et Protocol ‘’record layer”

Etudiants :
Mamadou **Senghor** (**senm1912**)
Clément **Dumas** (**dumc4449**)
Rafael **Dhaene** (**dhar2976**)
Ricardo **Beaujour** (**bear4830**)

Enseignant :
Professeur Martin Fiset

25 janvier 2026

Table des matières

Résumé

Ce travail présente la conception et l'analyse d'un protocole de communication sécurisé, composé de deux sous-systèmes complémentaires inspirés des standards modernes tels que TLS 1.3 : (i) un protocole de *handshake* assurant l'authentification mutuelle et l'établissement d'une clé de session avec protection avancée, et (ii) un protocole de *record layer* garantissant la confidentialité, l'intégrité et la protection contre le rejeu des messages échangés après l'établissement de la session.

Dans la première partie, nous décrivons un protocole de *handshake* robuste intégrant le *Perfect Forward Secrecy* (PFS) grâce à un échange de clés éphémères (type ECDHE), associé à des certificats émis par une autorité de confiance (CA) afin d'assurer l'authentification des participants. Ce protocole met en œuvre plusieurs mécanismes de sécurité :

- une négociation authentifiée de la suite cryptographique, empêchant toute dégradation d'algorithme ;
- l'établissement d'un secret partagé confidentiel, dérivé à l'aide d'une fonction de dérivation de clés (HKDF) ;
- la vérification d'intégrité et d'authenticité des échanges par signatures numériques ;
- la prévention du rejeu via l'usage de nonces de session uniques.

La seconde partie introduit le protocole *record layer*, qui exploite la clé de session issue du handshake pour chiffrer et authentifier les messages applicatifs à l'aide d'un schéma de chiffrement authentifié (AEAD, tel qu'AES-GCM). Sa conception repose sur les principes suivants :

- l'utilisation d'un compteur de séquence strictement croissant pour contrer les rejoués et réorganisations ;
- un identifiant de session dérivé cryptographiquement, assurant l'isolation entre sessions ;
- la dérivation de clés directionnelles indépendantes (client $\rightarrow$ serveur et serveur $\rightarrow$ client).

Les propriétés de sécurité attendues — robustesse, confidentialité, authenticité, intégrité, *Perfect Forward Secrecy* et résistance aux attaques de rejeu ou d'homme-du-milieu — sont ensuite démontrées à travers une argumentation formelle fondée sur les définitions du cours.

Enfin, une analyse conclusive résume les garanties offertes par notre protocole et identifie les conditions nécessaires à sa mise en œuvre sécurisée dans un contexte pratique.

I. Protocole Handshake avec Protection Avancée

1. Description du protocole

Le protocole proposé permet à Alice et Bob de s'authentifier mutuellement et d'établir une clé de session privée k offrant une forte sécurité, même si des clés privées à long terme venaient à être compromises plus tard (Perfect Forward Secrecy).

Le handshake repose sur trois étapes principales :

- **Négociation sécurisée :** Alice envoie une liste de suites cryptographiques supportées ainsi qu'un nonce aléatoire. Bob vérifie son certificat, choisit une suite commune et renvoie ses propres paramètres. Les deux parties construisent un résumé cryptographique (transcript) signé par Bob, empêchant toute modification ou dégradation des algorithmes.
- **Échange de clés éphémères authentifié :** Alice et Bob génèrent chacun une paire de clés Diffie-Hellman éphémères. Ils s'échangent leurs clés publiques et signent l'ensemble des paramètres déjà négociés, ce qui prouve que les deux parties participent réellement à l'échange. Le secret partagé issu de Diffie-Hellman est ensuite calculé indépendamment par chacun.
- **Confirmation mutuelle de la clé :** À partir du secret partagé, une clé de session k est dérivée via HKDF. Un message *Finished*, protégé par chiffrement et MAC, est échangé dans les deux sens pour confirmer que les deux parties ont bien obtenu la même clé de session. Si une vérification échoue, le protocole est immédiatement interrompu.

Lorsque ces trois étapes sont terminées avec succès, Alice et Bob disposent d'une clé de session k authentifiée, confidentielle et robuste contre les attaques de rejeu et de compromission future des clés privées.

2. Pseudo-code du protocole

Les trois phases décrites précédemment sont maintenant présentées sous forme de pseudocode détaillé, selon les règles vues au cours. Chaque algorithme spécifie les messages échangés et les vérifications assurant la sécurité du protocole :

- **Phase 1 :** Négociation des paramètres cryptographiques et authentification du certificat initial.
- **Phase 2 :** Échange des clés Diffie-Hellman éphémères et authentification mutuelle des participants.
- **Phase 3 :** Dérivation de la clé de session finale et confirmation que les deux parties possèdent la même clé.

Phase 1 — Négociation sécurisée

Algorithm 1: Phase 1 — Négociation sécurisée

Alice : **begin**

- `// Génération du nonce et annonce des algorithmes supportés`
- $nonce_A \leftarrow$ Gen_Random(128 *bits*)
- $algos_A \leftarrow$ {AES-256-GCM, ChaCha20-Poly1305, SHA-384, ECDHE-P384}
- `// Envoi du message Client Hello`
- Envoyer($nonce_A$, $algos_A$, $cert_{C\to A}$)

Bob : **begin**

- Recevoir($nonce_A$, $algos_A$, $cert_A$)
- `// Vérification du certificat d'Alice via la clé publique de Charlie`
- Si NOT VrifierCertificat($cert_A$, pk_C) alors **Abort**
- `// Génération du nonce et sélection de la suite cryptographique`
- $nonce_B \leftarrow$ Gen_Random(128 *bits*)
- $algos_B \leftarrow$ {AES-256-GCM, ChaCha20-Poly1305, SHA-384, ECDHE-P384}
- $suiteID \leftarrow$ BestCommon($algos_A$, $algos_B$)
- Si $suiteID = NULL$ alors **Abort**
- `// Calcul du transcript cumulatif (Client Hello + Server Hello)`
- $th \leftarrow H(\text{"Client Hello"} \| nonce_A \| algos_A)$
- $th \leftarrow H(th \| \text{"Server Hello"} \| nonce_B \| algos_B \| suiteID)$
- `// Identification des entités et signature d'engagement`
- $id_A \leftarrow H(cert_A)$
- $cert_B \leftarrow cert_{C\to B}$
- $id_B \leftarrow H(cert_B)$
- $sig_B \leftarrow$ Signer(sk_B, ("commit"$\|th\|id_A\|id_B$))
- `// Envoi du message Server Hello signé`
- Envoyer($nonce_B$, $algos_B$, $suiteID$, $cert_B$, sig_B)

Alice : **begin**

- Recevoir($nonce_B$, $algos_B$, $suiteID$, $cert_B$, sig_B)
- `// Vérification du certificat et reconstruction du transcript`
- Si NOT VrifierCertificat($cert_B$, pk_C) alors **Abort**
- $th \leftarrow H(\text{"Client Hello"} \| nonce_A \| algos_A)$
- $th \leftarrow H(th \| \text{"Server Hello"} \| nonce_B \| algos_B \| suiteID)$
- `// Vérification de la signature d'engagement de Bob`
- $id_A \leftarrow H(cert_{C\to A})$
- $id_B \leftarrow H(cert_B)$
- Si NOT VrifierSignature(pk_B, ("commit"$\|th\|id_A\|id_B$), sig_B) alors **Abort**

Phase 2 — Échange éphémère et authentification mutuelle

Algorithm 2: Phase 2 — Échange éphémère et authentification mutuelle

Alice : **begin**
- `// Génération de la paire de clés éphémères (Perfect Forward Secrecy)`
- $(sk_{Ae}, pk_{Ae}) \leftarrow \Pi_A^{KE}$
- `// Mise à jour du transcript avec la clé publique éphémère`
- $th \leftarrow H(th \| pk_{Ae})$
- `// Signature authentifiant l'identité d'Alice et le transcript courant`
- $sig_A \leftarrow \text{Signer}(sk_A, (\text{"client\_auth"} \| th \| id_A \| id_B))$
- Envoyer(pk_{Ae}, sig_A)

Bob : **begin**
- Recevoir(pk_{Ae}, sig_A)
- `// Validation de la clé éphémère reçue`
- Si NOT EstClPubliqueValide(pk_{Ae}) alors **Abort**
- `// Mise à jour du transcript`
- $th \leftarrow H(th \| pk_{Ae})$
- `// Vérification de la signature d'Alice`
- Si NOT VrifierSignature$(pk_A, (\text{"client_auth"} \| th \| id_A \| id_B), sig_A)$ alors **Abort**
- `// Génération de la paire de clés éphémères de Bob`
- $(sk_{Be}, pk_{Be}) \leftarrow \Pi_B^{KE}$
- `// Calcul du secret partagé (ECDHE)`
- $shared_B \leftarrow KE(sk_{Be}, pk_{Ae})$
- Si $shared_B = \text{NULL}$ alors **Abort**
- `// Suppression immédiate de la clé privée éphémère (PFS)`
- Supprimer(sk_{Be})
- `// Mise à jour du transcript`
- $th \leftarrow H(th \| pk_{Be})$
- `// Signature d'authentification de Bob`
- $sig_{B2} \leftarrow \text{Signer}(sk_B, (\text{"server\_auth"} \| th \| id_A \| id_B))$
- Envoyer(pk_{Be}, sig_{B2})

Alice : **begin**
- Recevoir(pk_{Be}, sig_{B2})
- `// Validation de la clé éphémère de Bob`
- Si NOT EstClPubliqueValide(pk_{Be}) alors **Abort**
- `// Mise à jour du transcript`
- $th \leftarrow H(th \| pk_{Be})$
- `// Vérification de la signature de Bob`
- Si NOT VrifierSignature$(pk_B, (\text{"server_auth"} \| th \| id_A \| id_B), sig_{B2})$ alors **Abort**
- `// Calcul du secret partagé`
- $shared_A \leftarrow KE(sk_{Ae}, pk_{Be})$
- Si $shared_A = \text{NULL}$ alors **Abort**
- `// Suppression immédiate de la clé privée éphémère (PFS)`
- Supprimer(sk_{Ae})

Phase 3 — Dérivation et confirmation mutuelle

Algorithm 3: Phase 3 — Dérivation et confirmation mutuelle

Alice : **begin**
- `// Dérivation de la clé maîtresse et des clés directionnelles`
- $k \leftarrow \text{HKDF}(shared_A, th, (\text{"session\_key"} \| id_A \| id_B))$
- $(k_e^{A\to B}, k_m^{A\to B}, k_e^{B\to A}, k_m^{B\to A}) \leftarrow \text{HKDF-Expand}(k, \text{"keys\_split"})$
- `// Construction du message Finished d'Alice`
- $tag_A \leftarrow \text{MAC}(k_m^{A\to B}, th \| \text{"finished\_A"})$
- $fin_A \leftarrow \text{ECpriv}(k_e^{A\to B}, (\text{"finished"} \| tag_A))$
- Envoyer(fin_A)

Bob : **begin**
- Recevoir(fin_A)
- `// Dérivation des clés (Bob calcule` k `indépendamment)`
- $k \leftarrow \text{HKDF}(shared_B, th, (\text{"session\_key"} \| id_A \| id_B))$
- $(k_e^{A\to B}, k_m^{A\to B}, k_e^{B\to A}, k_m^{B\to A}) \leftarrow \text{HKDF-Expand}(k, \text{"keys\_split"})$
- `// Vérification du message Finished d'Alice`
- $(\text{"finished"} \| tag_A) \leftarrow \text{DCpriv}(k_e^{A\to B}, fin_A)$
- Si $\text{VerifMAC}(k_m^{A\to B}, th \| \text{"finished\_A"}, tag_A) =$ Faux alors **Abort**
- `// Construction du message Finished de Bob`
- $tag_B \leftarrow \text{MAC}(k_m^{B\to A}, th \| \text{"finished\_B"})$
- $fin_B \leftarrow \text{ECpriv}(k_e^{B\to A}, (\text{"finished"} \| tag_B))$
- Envoyer(fin_B)

Alice : **begin**
- Recevoir(fin_B)
- `// Vérification du message Finished de Bob`
- $(\text{"finished"} \| tag_B) \leftarrow \text{DCpriv}(k_e^{B\to A}, fin_B)$
- Si $\text{VerifMAC}(k_m^{B\to A}, th \| \text{"finished\_B"}, tag_B) =$ Faux alors **Abort**
- `// Génération de l'identifiant de session`
- $sessionID \leftarrow H(th \| \text{"sid"})$
- Retourner(k, $sessionID$, $suiteID$)

Remarque : La dérivation simultanée des quatre sous-clés directionnelles $(k_e^{A\to B}, k_m^{A\to B}, k_e^{B\to A}, k_m^{B\to A})$ garantit que chaque partie puisse à la fois authentifier son propre message *Finished* et vérifier celui de l'autre. Cette symétrie assure la cohérence et la sécurité mutuelle du protocole, tout en préservant la propriété de Perfect Forward Secrecy.

Notations et primitives cryptographiques

- $H(\cdot)$: Fonction de hachage cryptographique (ex. : SHA-384) utilisée pour assurer l'intégrité et résumer les messages importants dans le transcript th.
- nonce : Valeur aléatoire utilisée une seule fois pour empêcher les attaques par rejeu.
- `certA`, `certB` : Certificats signés par l'autorité de confiance Charlie.
- `pkX` / `skX` : Clé publique et clé privée du participant X.
- Signer($sk, data$) : Produit une signature numérique authentifiant les données.
- VérifierSignature($pk, data, sig$) : Vérifie qu'une signature est valide.
- ECDHE.Generate() : Génère une paire de clés éphémères pour l'échange de clé.
- ECDHE(sk_e, pk_{peer}) : Permet de calculer un secret partagé grâce au Diffie-Hellman elliptique.

- HKDF$(shared, info)$: Dérive la clé de session à partir du secret partagé.
- Abort : Arrêt immédiat du protocole en cas d'erreur de sécurité détectée.
- SuiteID : Suite cryptographique choisie par les deux parties pour la session.
- Transcript th : Données importantes du handshake résumées avec $H(\cdot)$.
- `"commit"` :Constante de domaine (domain separator) marquant l'engagement du participant sur les paramètres négociés.
- `"client_auth"` : Constate utilisée dans la signature d'Alice pour assurer son rôle.
- `"server_auth"` : Constate utilisée dans la signature de Bob pour vérifier sa participation.
- k : Clé de session principale dérivée du secret partagé éphémère via HKDF. Elle sert de base à toutes les clés de chiffrement et authentification de la communication dans le Record Layer, et assure la propriété de Perfect Forward Secrecy.

3. Sécurité du protocole handshake

Robustesse et intégrité

Chaque message du handshake est inclus dans le transcript cryptographique th, protégeant l'intégralité de la négociation. Les signatures numériques (Π_{Sign}) garantissent qu'aucune modification, suppression ou interversion ne peut passer inaperçue. Les messages *Finished*, protégés par Π_{CPriv} et Π_{MAC}, valident que les deux entités dérivent la même clé k.

Authenticité des entités

Les certificats $cert_{C\text{ß}A}$ et $cert_{C\text{ß}B}$, vérifiés via la clé publique de Charlie (pk_C), assurent l'authenticité des participants. Les signatures sur les clés éphémères lient cryptographiquement les identités à la session en cours.

Prévention des attaques Unknown Key-Share (UKS)

Les identifiants $id_A = H(cert_{C\text{ß}A})$ et $id_B = H(cert_{C\text{ß}B})$ sont inclus dans chaque signature. Ainsi, une clé de session ne peut jamais être attribuée à une autre identité qu'à son véritable propriétaire.

Confidentialité et Perfect Forward Secrecy (PFS)

La clé finale k est dérivée uniquement du secret Diffie-Hellman éphémère obtenu via Π_{KE} (ECDHE). Les secrets éphémères sk_{Ae} et sk_{Be} sont supprimés après leur utilisation, empêchant toute récupération ultérieure du secret partagé.

Même si les clés de signature à long terme (sk_A ou sk_B) sont compromises après la fin du protocole, elles ne permettent pas de retrouver les secrets éphémères et donc pas la clé de session k. Les signatures prouvent l'authenticité des échanges, mais ne protègent jamais la matière secrète utilisée pour dériver k.

Protection contre les attaques par rejeu

Les nonces aléatoires $nonce_A$ et $nonce_B$ sont incorporés dans le transcript et les signatures, ce qui rend toute réutilisation d'anciens messages immédiatement detectable.

Protection contre la dégradation d'algorithme

La suite cryptographique choisie ($suiteID$) est incluse dans les données signées. Un attaquant

ne peut donc pas imposer un algorithme plus faible sans entraîner l'échec des vérifications.

Isolation stricte des sessions
Chaque session possède un identifiant unique $sessionID = H(th \| \text{"}sid\text{"})$, assurant qu'aucun message ne puisse être réutilisé dans une autre session.

Conclusion
Le protocole satisfait l'ensemble des propriétés de sécurité attendues d'un protocole moderne d'établissement de clé authentifiée : robustesse, intégrité, authenticité, confidentialité, PFS, résistance au rejeu, protection contre les attaques de dégradation et isolation des sessions. La clé finale k sert ensuite dans le *Record Layer* pour sécuriser tous les échanges de données.

II. Protocole “record layer” avec gestion de sessions multiples

1. Description du protocole

Une fois le *handshake* complété, Alice et Bob partagent une clé de session k authentifiée et secrète. Le rôle du protocole **Record Layer** est d'utiliser cette clé pour assurer la confidentialité, l'intégrité et l'authenticité de tous les messages m_x échangés, tout en gérant plusieurs sessions concurrentes.

Chaque session est identifiée par un `sessionID` unique dérivé du transcript du handshake. À partir de k, on dérive des sous-clés indépendantes par direction (Alice $\rightarrow$ Bob et Bob $\rightarrow$ Alice) à l'aide d'un HKDF :

$$(k_e^{A\to B}, k_m^{A\to B}, k_e^{B\to A}, k_m^{B\to A}) \leftarrow \text{HKDF-Expand}(k,\ sessionID \,\|\, \text{"traffic\_keys"}).$$

- $k_e^{\cdot}$ est utilisée pour le chiffrement symétrique via Π_{CPriv} ;
- $k_m^{\cdot}$ est utilisée pour l'authentification des messages via Π_{MAC} ;
- chaque direction maintient son propre compteur de séquence (seq_{snd} et seq_{rcv}).

Ce mécanisme permet :

- de garantir la **robustesse** et l'**intégrité** des échanges ;
- d'assurer la **confidentialité** du contenu ;
- de prévenir les attaques par **rejeu** et **réorganisation** ;
- de gérer simultanément plusieurs **sessions isolées**.

Ainsi, le *Record Layer* agit comme un canal sécurisé reposant sur la clé k, similaire à la couche TLS record, mais adapté au contexte du protocole défini dans la première partie.

2. Présentation du pseudo-code

Le pseudo-code suivant illustre les principales fonctions du protocole :

- **Initialisation de session** : dérivation des clés directionnelles et initialisation des compteurs de séquence ;

- **Envoi de message** : chiffrement du message, ajout d'un numéro de séquence et d'un tag MAC, puis transmission avec l'identifiant de session ;
- **Réception de message** : déchiffrement, vérification du MAC et du numéro de séquence pour détecter les rejets ou les réorganisations.

Ces algorithmes assurent que chaque message est correctement chiffré, authentifié, ordonné et associé à la bonne session.

Algorithme 1 — Initialisation de session

Algorithm 4: Initialisation locale d'une session sécurisée à partir de la clé k

Côté $X \in \{A, B\}$:
Input: Clé de session k, identifiant de session $sessionID$
Output: État de session initialisé pour X
begin
  `// 1. Dérivation des sous-clés directionnelles`
  $(k_e^{A\to B}, k_m^{A\to B}, k_e^{B\to A}, k_m^{B\to A}) \leftarrow HKDF\text{-}Expand(k,\ sessionID \| \text{"}traffic\_keys\text{"},\ 128)$
  `// 2. Assignation selon le rôle du participant`
  if $X = A$ **then**
    $k_e^{send} \leftarrow k_e^{A\to B}$
    $k_m^{send} \leftarrow k_m^{A\to B}$
    $k_e^{recv} \leftarrow k_e^{B\to A}$
    $k_m^{recv} \leftarrow k_m^{B\to A}$
  else
    $k_e^{send} \leftarrow k_e^{B\to A}$
    $k_m^{send} \leftarrow k_m^{B\to A}$
    $k_e^{recv} \leftarrow k_e^{A\to B}$
    $k_m^{recv} \leftarrow k_m^{A\to B}$
  `// 3. Initialisation des compteurs de séquence`
  $seq_{snd} \leftarrow 0$ `// Prochain numéro à envoyer`
  $seq_{rcv} \leftarrow 0$ `// Dernier numéro reçu valide`
  `// 4. Stockage de l'état de la session`
  $Sessions[sessionID] \leftarrow \{\ k_e^{send}, k_m^{send}, k_e^{recv}, k_m^{recv},\ seq_{snd}, seq_{rcv},\ active : True\ \}$
  return $sessionID$

Algorithme 2 — Envoi de message (Send_A2B)

Algorithm 5: Algorithme **Send_A2B** — Envoi chiffré et authentifié d'un message

Côté A (vers B) :
Input: $sessionID$, message clair m
Output: Message $record$ chiffré et authentifié envoyé avec succès
begin
    `// 1. Validation et récupération de la session`
    if $sessionID \notin Sessions$ **then**
        Lever Exception("Session invalide")
    $session \leftarrow Sessions[sessionID]$
    if $session.active = False$ **then**
        Lever Exception("Session inactive")
    `// 2. Capture et incrémentation du numéro de séquence`
    $seq \leftarrow session.seq_{snd}$
    $session.seq_{snd} \leftarrow session.seq_{snd} + 1$
    `// 3. Construction de l'en-tête authentifié`
    $header \leftarrow sessionID \| \text{"}A2B\text{"} \| Encoder\_Uint64(seq)$
    `// 4. Chiffrement authentifié (AEAD - AES-GCM recommandé)`
    $(c, tag) \leftarrow AES\text{-}GCM.Encrypt($
        $key = session.k_e^{send}$,
        $plaintext = m$,
        $associated\_data = header$)
    `// 5. Formation et envoi du message`
    $message\_record \leftarrow header \| c \| tag$
    EnvoyerSurRéseau($message\_record$)
    return *True* `// Succès de l'envoi`

Algorithme 3 — Réception d’un message

Algorithm 6: Algorithme **Recv_A2B** — Réception et vérification d’un message via AES-GCM

Côté B (réception depuis A) :
Input: Enregistrement reçu $message\_record$
Output: Message clair m livré si authentifié avec succès
begin
  // 1. Extraction des champs du message
  $sessionID \leftarrow message\_record[0:32]$
  $seq \leftarrow Decoder\_Uint64(message\_record[32:40])$
  $c \leftarrow message\_record[40 : len(message\_record) - 16]$
  $tag \leftarrow message\_record[len(message\_record) - 16 :]$
  // 2. Vérification de la session
  if $sessionID \notin Sessions$ **then**
    Rejeter("Session inconnue")
  $session \leftarrow Sessions[sessionID]$
  if $session.active = False$ **then**
    Rejeter("Session inactive")
  // 3. Protection contre le rejeu
  if $seq < session.seq_{rcv} + 1$ **then**
    Rejeter("Rejeu détecté")
  // 4. Protection contre la réorganisation
  if $seq > session.seq_{rcv} + 1$ **then**
    Rejeter("Message hors ordre")
  // 5. Validation du numéro attendu
  if $seq \neq session.seq_{rcv} + 1$ **then**
    Rejeter("Numéro de séquence invalide")
  // 6. Construction des données authentifiées (AAD)
  $AAD \leftarrow sessionID \| \text{"}A2B\text{"} \| Encoder\_Uint64(seq)$
  // 7. Déchiffrement et vérification simultanés (AES-GCM)
  $m \leftarrow AES\text{-}GCM.Decrypt($
    $key = session.k_e^{recv}$,
    $ciphertext = c$,
    $tag = tag$,
    $associated\_data = AAD$)
  // 8. Vérification du résultat
  if $m = \acute{E}CHEC$ **then**
    Rejeter("Authentification échouée")
  // 9. Mise à jour du compteur après succès
  $session.seq_{rcv} \leftarrow seq$
  // 10. Livraison du message
  Délivrer(m)
  return m

3. Justification des propriétés de sécurité

Confidentialité

Chaque message m est chiffré via Π_{CPriv} (par exemple AES-GCM) avec une clé directionnelle $k_e^{\cdot}$ dérivée de la clé maître k établie lors du handshake authentifié. Seuls Alice et Bob possèdent k, garantissant qu’eux seuls peuvent dériver les clés de chiffrement $k_e^{A\to B}$ et $k_e^{B\to A}$.

Un adversaire passif ne peut donc pas déchiffrer les messages interceptés. L'usage de clés directionnelles distinctes renforce la séparation cryptographique et prévient les attaques de réflexion où un message intercepté serait renvoyé à son émetteur.

Intégrité et authentification

Chaque message est authentifié grâce à un chiffrement authentifié (AES-GCM) ou à Π_{MAC} dans une construction *Encrypt-then-MAC*. Le tag d'authentification est calculé sur $AAD \| c$, où AAD contient le $sessionID$, la direction et le numéro de séquence. La vérification du tag est effectuée **avant** tout déchiffrement, ce qui empêche toute fuite d'information en cas de message falsifié. Ainsi, toute modification du ciphertext, du numéro de séquence ou du $sessionID$ entraîne l'échec immédiat de la vérification du MAC.

L'identité de l'émetteur est implicite dans les clés directionnelles ($k_m^{A \to B} \neq k_m^{B \to A}$), dérivées du handshake authentifié. Le label directionnel (“A2B” ou “B2A”) inclus dans l'AAD empêche qu'un message légitime soit accepté dans la mauvaise direction.

Protection contre le rejeu

Chaque direction maintient un compteur de séquence indépendant (seq_{snd} et seq_{rcv}). Tout message reçu avec $seq \leq seq_{rcv}$ est immédiatement rejeté comme tentative de rejeu. Le numéro de séquence est inclus dans l'AAD et protégé par le tag d'authentification, ce qui empêche un adversaire de le modifier. Ainsi, même si un message chiffré est capturé puis rejoué, il sera refusé car son numéro est déjà dépassé dans l'état de la session.

Protection contre la réorganisation

Le protocole n'accepte qu'un seul ordre de livraison : le message dont $seq = seq_{rcv} + 1$. Tout message reçu hors de cet ordre est rejeté. Cela assure que l'ordre de réception correspond strictement à l'ordre d'envoi. Une perte ou un décalage dans les numéros de séquence est immédiatement détecté. Cette approche, similaire au fonctionnement de TLS sur TCP, garantit une cohérence stricte au prix d'une tolérance nulle aux pertes réseau.

Robustesse

Le protocole assure la livraison correcte des messages non altérés grâce à la vérification d'intégrité avant tout traitement du contenu. Tout message corrompu — qu'il soit dû à une erreur réseau ou à une modification malveillante — est rejeté sans impact sur l'état interne de la session. Le compteur seq_{rcv} n'est incrémenté que si *toutes* les vérifications (authenticité, intégrité, ordre) sont réussies, ce qui maintient la cohérence de l'état même face à des attaques actives.

Gestion de sessions multiples et isolation

Le $sessionID$ est dérivé cryptographiquement du transcript du handshake :

$$sessionID = H(\text{transcript}),$$

garantissant son unicité. Il remplit deux rôles essentiels :

1. **Dérivation de clés** :

$$(k_e^{A\to B}, k_m^{A\to B}, k_e^{B\to A}, k_m^{B\to A}) \leftarrow HKDF\text{-}Expand(k,\, sessionID \| \text{"}traffic\_keys\text{"}),$$

assurant que deux sessions différentes produisent des ensembles de clés indépendants même à partir de la même clé maître k.

2. **Authentification du contexte** : le $sessionID$ est inclus dans l'AAD de chaque message, liant cryptographiquement le message à sa session d'origine.

Cette double inclusion assure une **isolation complète** entre sessions :

- un message de la session S_1 injecté dans S_2 est rejeté, car son $sessionID$ ne correspond pas ;
- le MAC calculé avec les clés de S_1 échoue avec celles de S_2 ;
- les compteurs de séquence étant indépendants, aucune interférence entre sessions n'est possible.

Exemple d'attaque empêchée. Un adversaire capturant un message M de la session S_1 et tentant de l'injecter dans S_2 échouera à trois niveaux : (1) $sessionID$ incorrect, (2) MAC invalide, (3) échec du déchiffrement.

Synthèse

Le protocole *Record Layer* satisfait toutes les propriétés de sécurité requises :

- **Confidentialité et intégrité** via Π_{CPriv} (AES-GCM) ;
- **Authenticité** grâce à la dérivation de clés directionnelles du handshake ;
- **Protection contre le rejeu et la réorganisation** par numéros de séquence stricts ;
- **Isolation cryptographique** des sessions multiples via le $sessionID$.

Cette conception, inspirée du Record Layer de TLS 1.3, offre un canal de communication fiable, robuste et sécurisé, assurant confidentialité, authenticité et synchronisation parfaite entre les parties.

III. Analyse des attaques

Cette section évalue la résistance du protocole proposé face à plusieurs attaques classiques : rejeu de handshake, dégradation algorithmique, rejeu intersession et compromission de clés de longue durée.

1. Attaque par rejeu sur le handshake

Un adversaire enregistre une exécution complète et valide du protocole entre Alice et Bob, puis tente de la rejouer ultérieurement pour forcer la réutilisation d'une ancienne clé.

Prévention. Les signatures numériques (σ_A, σ_B) authentifient non seulement les identités, mais aussi les nonces $(nonce_A, nonce_B)$ et les clés éphémères (pk_{Ae}, pk_{Be}) spécifiques à chaque session. Lors d'une nouvelle exécution, chaque partie génère de nouveaux nonces aléatoires, ce qui entraîne un transcript th entièrement différent. Ainsi, toute tentative de rejouer un ancien message (ex. `ServerHello` avec un ancien $nonce_B$) échoue, car la vérification de la signature de Bob ne correspond plus au transcript actuel.

De plus, Bob peut conserver un petit historique des identifiants de session récents pour détecter toute répétition. La probabilité qu'un couple $(nonce_A, nonce_B)$ soit régénéré identiquement est cryptographiquement négligeable.

2. Attaque par dégradation (downgrade attack)

L'attaquant tente de modifier les messages échangés lors de la négociation afin de forcer l'utilisation d'un algorithme plus faible (par exemple, DES au lieu de AES-256).

Détection et prévention. Alice envoie sa liste d'algorithmes supportés ($ALGS_A$) ; Bob choisit un algorithme commun et signe un résumé cryptographique (le transcript) incluant la version exacte de $ALGS_A$ reçue. Lorsqu'Alice reçoit la réponse signée, elle vérifie la signature à partir de sa propre version de $ALGS_A$. Si un attaquant a modifié la liste en transit, les deux valeurs de transcript diffèrent et la signature devient invalide.

Enfin, Alice applique une *politique de refus explicite* : elle rejette toute négociation qui aboutit à un algorithme plus faible que le meilleur commun disponible. Cette politique élimine toute tentative d'abaissement volontaire du niveau de sécurité.

3. Attaque par rejeu intersessions

Un adversaire capture un message chiffré c d'une session précédente et tente de le rejouer dans une autre session.

Prévention. Chaque message inclut :

$$(sessionID,\ seq,\ tag)$$

et est chiffré et authentifié avec des clés dérivées du $sessionID$. Ainsi, les clés de chiffrement et d'authentification diffèrent pour chaque session.

Si l'adversaire modifie le $sessionID$, la vérification du tag échoue immédiatement. S'il le conserve, le récepteur détecte que le message n'appartient pas à la session courante et le rejette. De plus, les compteurs de séquence (seq_{snd}, seq_{rcv}) sont indépendants pour chaque session et pour chaque direction, ce qui rend impossible l'acceptation d'un message d'une autre session.

4. Attaque par compromission de clé à long terme

L'adversaire obtient la clé privée de signature d'une des parties après la fin de la session et tente de déchiffrer les communications passées.

Analyse. Même avec la clé de signature compromise, l'adversaire ne peut pas retrouver les clés de session passées, car celles-ci sont dérivées d'un secret éphémère issu de l'échange de clés $(ephA_{priv}, ephB_{priv})$, détruit après usage. Les clés de signature servent uniquement à authentifier le handshake, et non à dériver le secret partagé. Ainsi, la **Perfect Forward Secrecy (PFS)** est assurée : la compromission d'une clé à long terme ne permet pas de déchiffrer les échanges antérieurs.

Seule une mauvaise implémentation — par exemple, une réutilisation ou une conservation des clés éphémères — compromettrait cette propriété.

5. Synthèse

- **Rejeu du handshake** : empêché par les nonces aléatoires et la signature du transcript.

- **Dégradation algorithmique** : empêchée par la signature du transcript incluant $ALGS_A$ et $suiteID$.
- **Rejeu intersession** : empêché par l'inclusion du $sessionID$ et du numéro de séquence dans l'AAD.
- **Compromission de clé à long terme** : neutralisée par l'échange éphémère ECDHE et la suppression des clés.

En conclusion, le protocole proposé résiste aux attaques classiques visant les canaux sécurisés modernes et offre les garanties de sécurité exigées dans un contexte équivalent à TLS 1.3.

6. Limitations et considérations pratiques

Tolérance aux pertes : Le protocole Record Layer impose un ordre strict (seq = seq_rcv + 1), adapté à TCP mais inadapté à UDP/DTLS où des pertes sont attendues.

Performance : Le chiffrement authentifié AES-GCM offre d'excellentes performances grâce à l'accélération matérielle (AES-NI), mais nécessite une gestion soigneuse des nonces pour éviter les réutilisations catastrophiques.

Gestion de l'état : Le maintien de compteurs de séquence par session nécessite une synchronisation stricte. Une perte de synchronisation force la renégociation complète.

Références

1. E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, IETF, Aug. 2018. *Disponible à :* `https://doi.org/10.17487/RFC8446`
2. T. Perrin, "The Noise Protocol Framework," Revision 34, 2018. *Disponible à :* `https://noiseprotocol.org/noise.pdf`
3. M. Marlinspike and T. Perrin, "The X3DH Key Agreement Protocol," Signal Foundation, 2016. *Disponible à :* `https://signal.org/docs/specifications/x3dh/`
4. M. Marlinspike and T. Perrin, "The Double Ratchet Algorithm," Signal Foundation, 2016. *Disponible à :* `https://signal.org/docs/specifications/doubleratchet/`
5. M. Bellare and P. Rogaway, "Entity Authentication and Key Distribution," in *Advances in Cryptology — CRYPTO'94*, pp. 232–249, Springer, 1994. *Disponible à :* `https://cseweb.ucsd.edu/~mihir/papers/eakd.pdf`
6. H. Krawczyk and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)," RFC 5869, IETF, May 2010. *Disponible à :* `https://doi.org/10.17487/RFC5869`