Titre du projet final

Concevoir et analyser un mini-système de communication sécurisée

Filière : 4ème année en Ingénierie Informatique & IA

Présenté par :

DIAWANE Ramatoulaye

Date : 23-01-2026

Partie1 - Confidentialité des échanges (WhatsApp – chiffrement bout en bout)

1. Expliquer le principe du chiffrement symétrique:

Le chiffrement symétrique est un mécanisme cryptographique dans lequel l'émetteur et le récepteur utilisent une clé secrète commune pour chiffrer et déchiffrer les données.

Principe mathématique (simple):

- C=EK(M)
- M=DK(C)

2. Chiffrer un message confidentiel (fichier ou message texte):

```
cisco@labvm:~$ nano secret.txt
cisco@labvm:~$ openssl enc -aes-256-cbc -salt -pbkdf2 -in secret.txt -out secret.txt.enc
enter AES-256-CBC encryption password:
Verifying - enter AES-256-CBC encryption password:
cisco@labvm:~$ openssl enc -d -aes-256-cbc -pbkdf2 -in secret.txt.enc -out secret_dechiffre.txt
enter AES-256-CBC decryption password:
bad decrypt
40C79B548C7F0000:error:1C800064:Provider routines:ossl_cipher_unpadblock:bad decrypt:../providers/implementations/ciphers/ciphercommon_block.c
:129:
cisco@labvm:~$
```

Le message est chiffré à l'aide de l'algorithme AES avec une clé secrète partagée. Le texte chiffré obtenu est illisible sans la clé.

3. Montrer l'impact de la longueur du mot de passe sur la sécurité:

```
cisco@labvm:~$ zip -e try3.zip pass*
Enter password:
Verify password:
  adding: pass10.txt (deflated 50%)
  adding: pass14.txt (deflated 33%)
  adding: pass6.txt (deflated 17%)
cisco@labvm:~$ fcrackzip -vul 1-12 try3.zip
found file 'pass10.txt', (size cp/uc     17/    10, flags 9, chk 45a1)
found file 'pass14.txt', (size cp/uc     22/    15, flags 9, chk 4590)
found file 'pass6.txt', (size cp/uc     17/     6, flags 9, chk 4575)
checking pw ucUM~
```

Une attaque par force brute a été lancée à l'aide de l'outil fcrackzip afin de casser le mot de passe de l'archive ZIP.

L'outil a commencé à tester automatiquement des combinaisons de mots de passe de longueur comprise entre 1 et 12 caractères.

Cependant, aucune correspondance n'a été trouvée avant l'interruption de l'attaque,

ce qui indique que le mot de passe est soit suffisamment long et complexe, rendant l'attaque inefficace dans le temps imparti. D'ou l'importance de mettre un mot de pass long avec plusieurs caractères.

4. Répondre aux questions suivantes :

- Les mots de passe courts offrent un faible niveau de sécurité et peuvent être devinés par des attaques automatisées. WhatsApp utilise donc des clés cryptographiques longues et générées automatiquement, bien plus robustes qu'un mot de passe choisi par un utilisateur.
- WhatsApp utilise une clé dérivée automatiquement pour chaque message afin de garantir la confidentialité persistante. Ainsi, même si une clé est compromise, les messages passés et futurs restent protégés.

Conclusion:

WhatsApp repose sur un chiffrement de bout en bout utilisant l'algorithme AES. Une clé unique est générée par session et dérivée automatiquement pour chaque message. Les clés ne sont jamais stockées sur les serveurs, ce qui garantit que seuls les utilisateurs communicants peuvent accéder au contenu des messages.

Partie 2 – Intégrité et authentification (Signature numérique – messages WhatsApp)

1. Générer une paire de clés (publique / privée):

```
cisco@labvm:~$ openssl genrsa -out private_key.pem 2048
cisco@labvm:~$ openssl rsa -in private_key.pem -pubout -out public_key.pem
writing RSA key
```

La clé publique est dérivée de la clé privée et peut être partagée sans risque.Une paire de clés asymétriques est composée d'une clé privée, utilisée pour signer les messages, et d'une clé publique, utilisée pour vérifier la signature.

2. Signer un message:

```
cisco@labvm:~$ echo "Bonjour Bob, message authentique" > message.txt
cisco@labvm:~$ openssl dgst -sha256 -sign private_key.pem -out signature.bin message.txt
```

Le message est d'abord haché avec SHA-256, puis l'empreinte est signée à l'aide de la clé privée de l'émeteur.

3. Vérifier la signature:

```
cisco@labvm:~$ openssl dgst -sha256 -verify public_key.pem -signature signature.bin message.txt
Verified OK
cisco@labvm:~$
```

La vérification réussit, ce qui prouve que le message provient bien du détenteur de la clé privée et qu'il n'a pas été modifié.

4. Modifier le message et montrer que la vérification échoue:

```
cisco@labvm:~$ echo "Bonjour Bob, message modifié" > message.txt
cisco@labvm:~$ openssl dgst -sha256 -verify public_key.pem -signature signature.bin message.txt
Verification failure
4077B5EE2B7F0000:error:02000068:rsa routines:ossl_rsa_verify:bad signature:../crypto/rsa/rsa_sign.c:430:
4077B5EE2B7F0000:error:1C880004:Provider routines:rsa_verify:RSA lib:../providers/implementations/signature/rsa_sig.c:774:
cisco@labvm:~$
```

Toute modification du message entraîne l'échec de la vérification, ce qui garantit l'intégrité des données.

Lien clair avec WhatsApp :

Dans WhatsApp, chaque utilisateur possède une clé privée stockée sur son appareil. Les messages sont signés avant l'envoi afin de garantir l'authenticité de l'expéditeur et l'intégrité du message. Toute modification entraîne l'échec de la vérification, empêchant ainsi l'usurpation d'identité.

Réponses aux questions de réflexion:

- **Pourquoi la signature est indispensable même si le message est chiffré ?**

La signature numérique reste indispensable même si le message est chiffré, car le chiffrement seul ne garantit ni l'identité de l'expéditeur ni l'intégrité du message. De plus le chiffrement assure surtout la confidentialité: seul quelqu'un qui a la bonne clé peut lire le contenu. Mais un attaquant peut toujours remplacer le message chiffré par un autre, ou se faire passer pour l'expéditeur légitime.

La signatute numérique apporte en plus :

- ✓ l'authenticité : le destinataire sait qui a envoyé le message (liée à la clé privée de l'expéditeur)
- ✓ l'intégrité : toute modification du message (même d'un bit) fait échouer la vérification de signature .
- ✓ la non-répudiation : l'expéditeur ne peut pas nier avoir signé ce message

Donc : un message peut être parfaitement chiffré (personne ne peut le lire) mais modifié ou

forgé sans qu'on le remarque ; la signature est ce qui permet de vérifier qu'il vient bien du bon auteur et qu'il n'a pas été altéré.

➢ **Différence entre confidentialité et intégrité ?**

- Confidentialité :

Objectif : empêcher la lecture par des personnes non autorisées.

Exemple de mécanisme : chiffrement symétrique/asymétrique (AES, RSA, etc.).

- Intégrité :

Objectif : garantir que les données n'ont pas été modifiées (ni supprimées ni ajoutées) de manière non autorisée, pendant le stockage ou le transport.

Exemple de mécanisme : fonctions de hachage (SHA- 256), MAC, HMAC, signatures numériques

En résumé : la confidentialité protège le secret du contenu, l'intégrité protège la fidélité du contenu ; un système sécurisé doit avoir les deux (plus l'authentification et la non- répudiation).

Partie 3 – HTTPS et TLS (Sécurisation client ↔ serveur)

1. Expliquer comment HTTPS ajoute une couche TLS au-dessus de HTTP:

- **HTTP** est le protocole qui permet de transférer des données entre un client (navigateur) et un serveur (site web), mais il n'est pas sécurisé : les données sont envoyées en clair et peuvent être interceptées.
- **HTTPS** (HTTP Secure) ajoute TLS (Transport Layer Security) au-dessus de HTTP.
- **TLS** chiffre les données échangées, assure leur intégrité et authentifie le serveur.

- Schéma simplifié :

Client ↔ TLS (chiffrement) ↔ HTTP ↔ Serveur

2. Décrire le handshake TLS (sans code):

Le handshake TLS est le processus par lequel le client et le serveur établissent une connexion sécurisée :

1-ClientHello : le client propose les versions de TLS et les suites cryptographiques qu'il supporte.

2-ServerHello : le serveur choisit la version TLS et la suite cryptographique.

3-Certificat du serveur : le serveur envoie son certificat (identité + clé publique) pour que le client puisse vérifier son authenticité.

4-Échange de clés : le client et le serveur se mettent d'accord sur une clé de session symétrique qui servira à chiffrer les données.

5-Finished : le client et le serveur confirment que le handshake est terminé, la communication chiffrée peut commencer.

3. Analyser un certificat HTTPS :

Pour analyser le certificat HTTPS de Google, nous avons utilisé la commande openssl s_client pour établir une connexion TLS sur le port 443 et récupérer le certificat du serveur, que nous avons ensuite exporté au format PEM dans le fichier google-cert.pem. En examinant ce certificat avec openssl x509, nous avons pu identifier l'émetteur (issuer) comme étant « Google Trust Services, CN = WR2 ». Nous avons également extrait la clé publique, qui est une clé elliptique de 256 bits utilisant la courbe P-256, et consulté son algorithme et ses

détails techniques. Enfin, nous avons généré l'empreinte SHA-256 (fingerprint) du certificat, qui permet de vérifier son intégrité et de s'assurer qu'il n'a pas été modifié. Cette analyse montre comment HTTPS, via TLS, permet d'authentifier le serveur et de sécuriser la communication entre le client et le serveur.

3. Expliquer comment un proxy HTTPS peut intercepter le trafic:

- Un proxy HTTPS agit comme intermédiaire entre le client et le serveur.

- Normalement, TLS empêche toute interception, mais un proxy peut utiliser un certificat intermédiaire :

  Le client croit communiquer avec le serveur réel, mais en réalité le proxy déchiffre le trafic, puis le chiffre de nouveau vers le serveur.

  Ceci fonctionne uniquement si le proxy est installé et reconnu par le client (certificat de confiance).

- C'est la raison pour laquelle HTTPS protège :

  les mots de passe

  les paiements

  les messages web

- Note spécifique sur WhatsApp : TLS est utilisé uniquement pour la connexion initiale, ensuite la communication est chiffrée de bout en bout (E2EE), ce qui empêche même WhatsApp d'accéder au contenu des messages.

Conclusion générale:

Ce projet a permis de concevoir un mini-système de communication sécurisée en combinant trois briques essentielles : le **chiffrement** pour protéger la confidentialité, les **signatures numériques** pour garantir l'intégrité et l'authenticité, et les **certificats TLS/HTTPS** pour établir un modèle de confiance entre le client et le serveur. L'analyse pratique d'un certificat HTTPS a illustré concrètement ces mécanismes.

En parallèle, le lien avec **WhatsApp** a permis de comprendre la complémentarité entre TLS pour la connexion initiale et le **chiffrement de bout en bout (E2EE)** pour sécuriser les messages eux-mêmes. Ce projet montre que l'intégration de ces mécanismes constitue une base solide pour sécuriser toute communication moderne, et reflète les principes appliqués dans des systèmes réels comme HTTPS et WhatsApp.