

Chiffrement de bout en bout

Cryptologie moderne – Projet

Thomas Peugnet – 2023

État de l'art	3
Introduction.....	3
L'Ère Pré-E2EE	3
Qu'est-ce que l'End-to-End Encryption?.....	4
E2EE dans les Applications de Messagerie	5
Futurs Tendances de l'E2EE	5
Cas pratique – Utilisation des sockets pour envoyer un message	6
Introduction.....	6
Première approche : Sans chiffrement de bout en bout.....	7
Seconde approche : Utilisation du chiffrement de bout en bout.....	10
Conclusion	12
Bibliographie	12
État de l'art	12
Cas pratique.....	12

État de l'art

Introduction

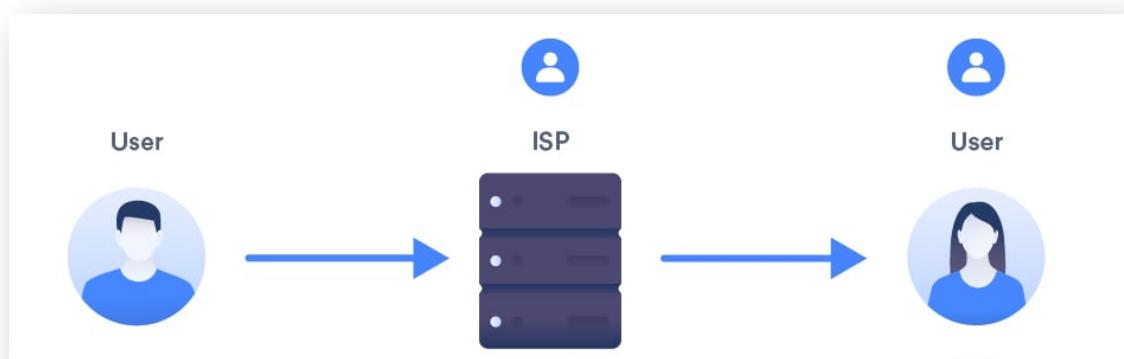
Dans un monde de plus en plus numérique, la sécurité des données et la protection de la vie privée sont devenues des préoccupations majeures. L'encryption de bout en bout (E2EE) est une technologie fondamentale qui a révolutionné la manière dont nous sécurisons nos communications et nos données en ligne. Ce texte a pour objectif de fournir un État de l'art de l'end-to-end encryption, en examinant son rôle dans l'évolution de la sécurité des données et de la confidentialité.

L'Ère Pré-E2EE

Avant l'adoption généralisée de l'E2EE, la sécurité des données en ligne laissait beaucoup à désirer. Les communications électroniques étaient souvent exposées aux interceptions malveillantes, mettant en danger la confidentialité des informations échangées.

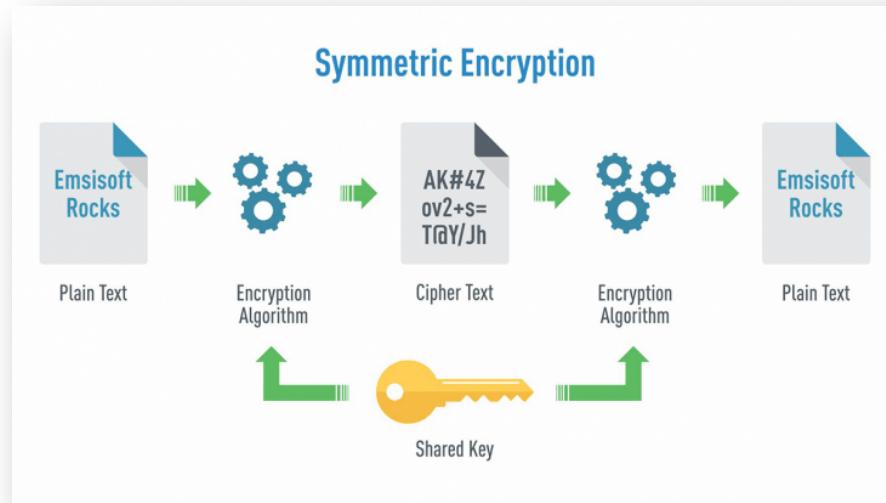
Pendant cette période, les données sensibles pouvaient être compromises lors de leur transmission sur Internet, et les fournisseurs de services de communication avaient un accès sans restriction aux données de leurs utilisateurs. Les pirates informatiques et les gouvernements pouvaient exploiter ces vulnérabilités pour accéder à des informations privées, soulevant ainsi d'énormes préoccupations en matière de sécurité et de confidentialité.

De façon plus schématique, on peut regarder le dessin suivant.



Qu'est-ce que l'End-to-End Encryption?

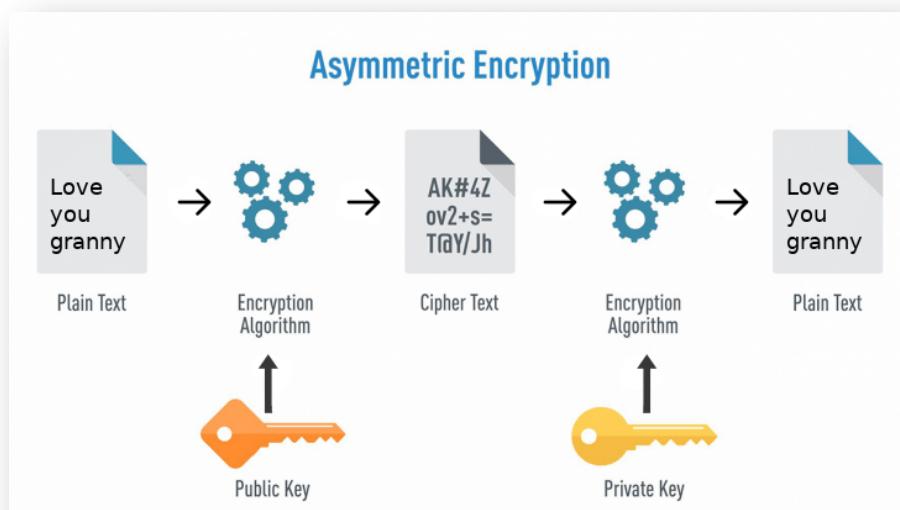
L'end-to-end encryption (E2EE), ou chiffrement de bout en bout, est une technologie de sécurité de pointe qui garantit que seuls l'expéditeur et le destinataire d'un message ou d'une donnée peuvent la lire. Contrairement aux méthodes de chiffrement traditionnelles, où les clés de déchiffrement étaient souvent détenues par un tiers, l'E2EE permet aux utilisateurs de conserver un contrôle total sur leurs données.



Le fonctionnement de l'E2EE repose sur l'utilisation de clés de chiffrement spécifiques à chaque utilisateur.

Lorsqu'un expéditeur envoie un message, il est chiffré avec la clé publique du destinataire, et seule la clé privée du destinataire peut le déchiffrer. Ainsi, même si un pirate informatique intercepte le message en transit, il ne peut pas le lire sans la clé privée du destinataire.

A noter que le schéma ci-dessus indique un chiffrement de bout en bout avec deux clés identiques partagées. Le schéma ci-dessous illustre davantage le fonctionnement avec un système de clés publiques/privées.



E2EE dans les Applications de Messagerie

L'E2EE a été largement adopté dans les applications de messagerie populaires telles que WhatsApp, Signal et Telegram. Cette adoption généralisée a considérablement amélioré la confidentialité des conversations en ligne. Les utilisateurs peuvent désormais échanger des messages, des photos et des vidéos en toute confiance, sachant que leurs communications sont sécurisées contre les regards indiscrets.

Cependant, l'implémentation de l'E2EE dans les applications de messagerie n'est pas sans défis. Les entreprises doivent résoudre des problèmes tels que la gestion des clés de chiffrement, l'interopérabilité entre différentes applications, la convivialité de l'interface utilisateur et la conformité réglementaire. De plus, bien que l'E2EE protège les messages en transit, elle ne garantit pas nécessairement la sécurité des appareils des utilisateurs, qui peuvent toujours être vulnérables aux attaques.

Enfin, le chiffrement de bout en bout garantit la confidentialité des données durant le transit, mais pas avant ce dernier : si le développeur de l'application de messagerie décide de récupérer les données durant le moment où elles sont tapées sur le clavier, avant que l'utilisateur clique sur [Envoyer](#), ces données ne sont pas nécessairement régies par du chiffrement de bout en bout.

Futurs Tendances de l'E2EE

L'E2EE continue d'évoluer pour répondre aux besoins croissants de sécurité et de confidentialité des utilisateurs. Les chercheurs et les développeurs travaillent sur des améliorations pour simplifier la gestion des clés de chiffrement, améliorer la résilience contre les attaques, et étendre son utilisation à d'autres domaines tels que la téléphonie et la vidéoconférence.

De plus, la montée en puissance de l'informatique quantique pose de nouveaux défis à la sécurité de l'E2EE. Les chercheurs se penchent sur des technologies de chiffrement post-quantique qui seront capables de résister aux attaques des ordinateurs quantiques puissants. Cette avancée est cruciale pour maintenir la sécurité de l'E2EE à long terme.

Cas pratique – Utilisation des sockets pour envoyer un message

Introduction

Pour illustrer l'intérêt du chiffrement de bout en bout, nous allons procéder à une petite expérience entre deux protagonistes, Bob et Alice. Ces derniers souhaitent pouvoir communiquer, certes sur la même machine (127.0.0.1) mais en utilisant la technologie des sockets.

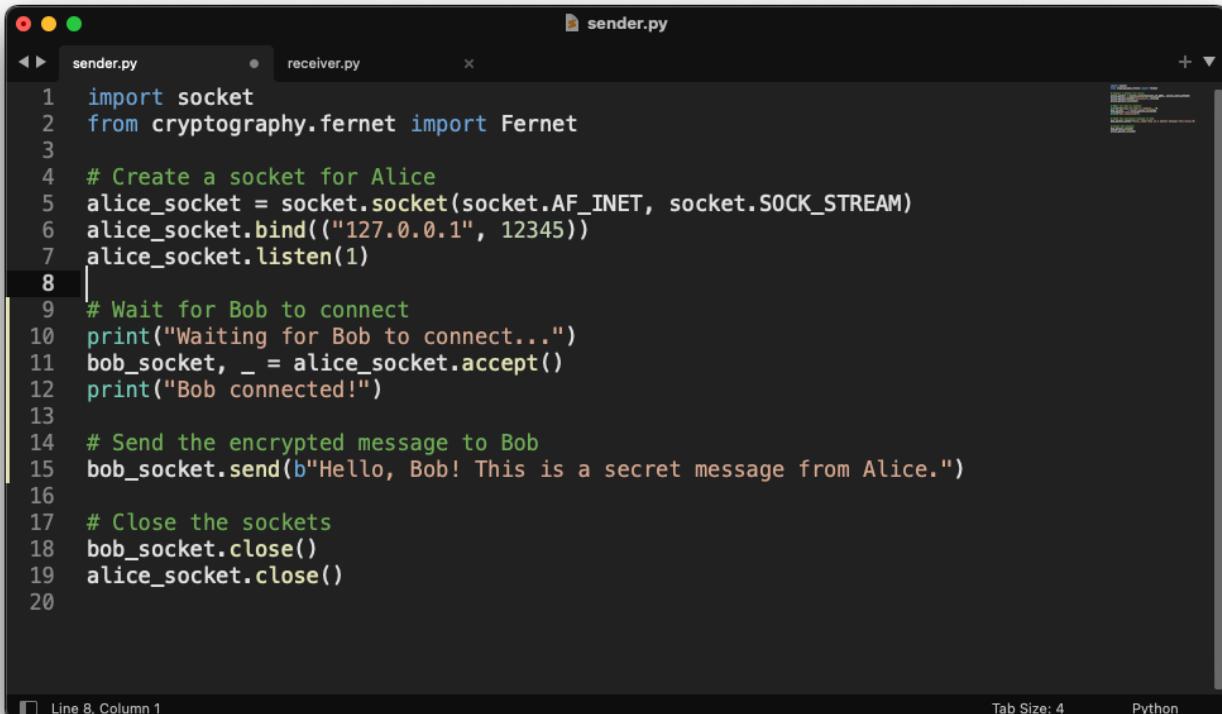
Pour cela, ils vont développer un programme (`sender.py` et `receiver.py`) afin de pouvoir discuter. `sender.py` fera hôte de serveur, et écoutera les connexions entrantes. `Receiver.py` se connectera et recevra le message envoyé à la connexion au serveur.

Cependant, un méchant pirate va tenter de d'intercepter leur message (oui, toujours sur la même machine, on fait avec les moyens matériels du bord.. !). Ce pirate se nomme BadBob.

BadBob, par l'intermédiaire du logiciel Wireshark, va tenter de récupérer les paquets transmis entre Bob et Alice.

Première approche : Sans chiffrement de bout en bout

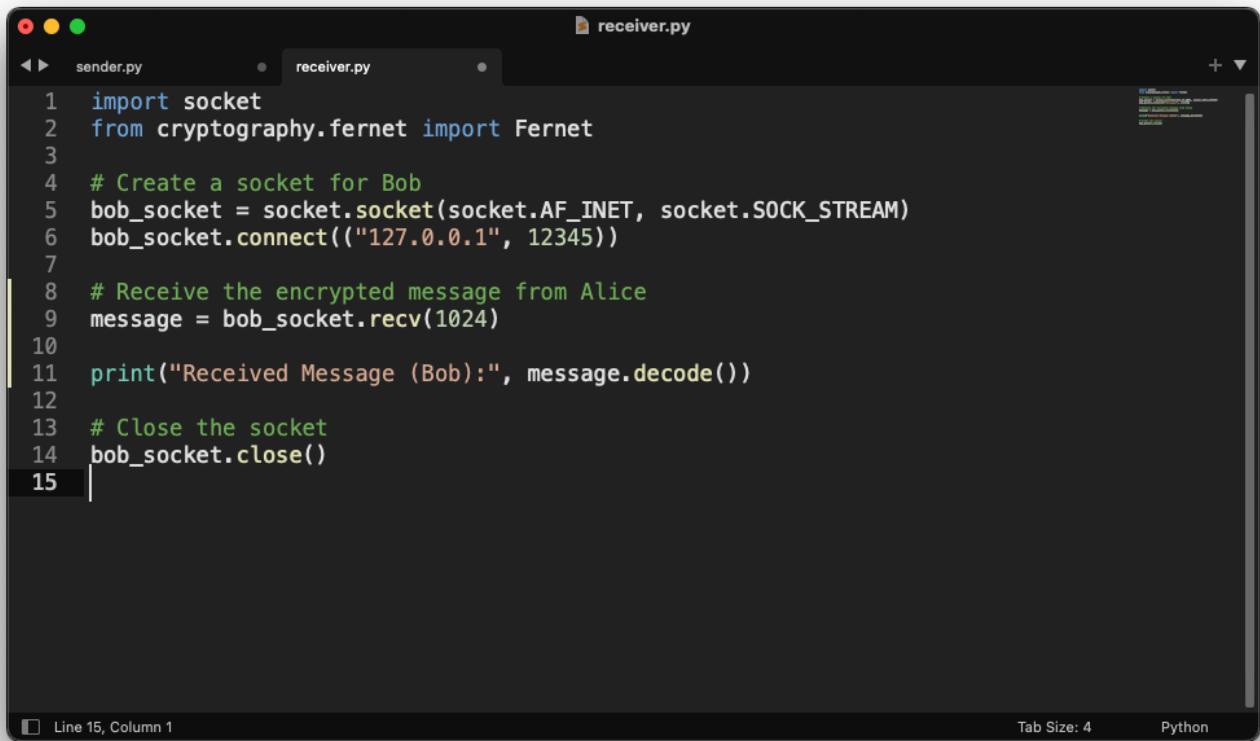
Nous avons donc rédigé deux programmes, `sender.py` et `receiver.py`, possédant le code ci-dessous :



A screenshot of a dark-themed code editor window titled "sender.py". The code is written in Python and uses the Fernet library for encryption. It creates a socket for Alice, waits for Bob to connect, sends a message to Bob, and then closes both sockets.

```
1 import socket
2 from cryptography.fernet import Fernet
3
4 # Create a socket for Alice
5 alice_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
6 alice_socket.bind(("127.0.0.1", 12345))
7 alice_socket.listen(1)
8
9 # Wait for Bob to connect
10 print("Waiting for Bob to connect...")
11 bob_socket, _ = alice_socket.accept()
12 print("Bob connected!")
13
14 # Send the encrypted message to Bob
15 bob_socket.send(b"Hello, Bob! This is a secret message from Alice.")
16
17 # Close the sockets
18 bob_socket.close()
19 alice_socket.close()
20
```

Line 8, Column 1 Tab Size: 4 Python

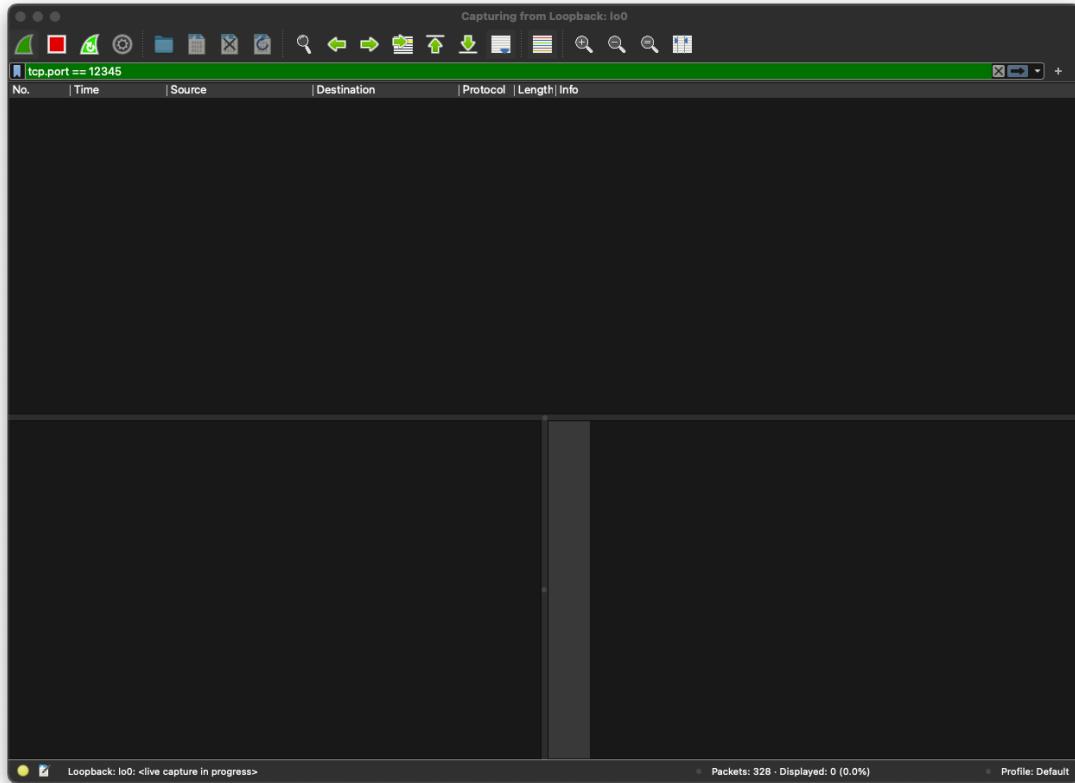


A screenshot of a dark-themed code editor window titled "receiver.py". The code is written in Python and uses the Fernet library for decryption. It creates a socket for Bob, connects to Alice's socket, receives a message from Alice, prints it, and then closes the socket.

```
1 import socket
2 from cryptography.fernet import Fernet
3
4 # Create a socket for Bob
5 bob_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
6 bob_socket.connect(("127.0.0.1", 12345))
7
8 # Receive the encrypted message from Alice
9 message = bob_socket.recv(1024)
10
11 print("Received Message (Bob):", message.decode())
12
13 # Close the socket
14 bob_socket.close()
15
```

Line 15, Column 1 Tab Size: 4 Python

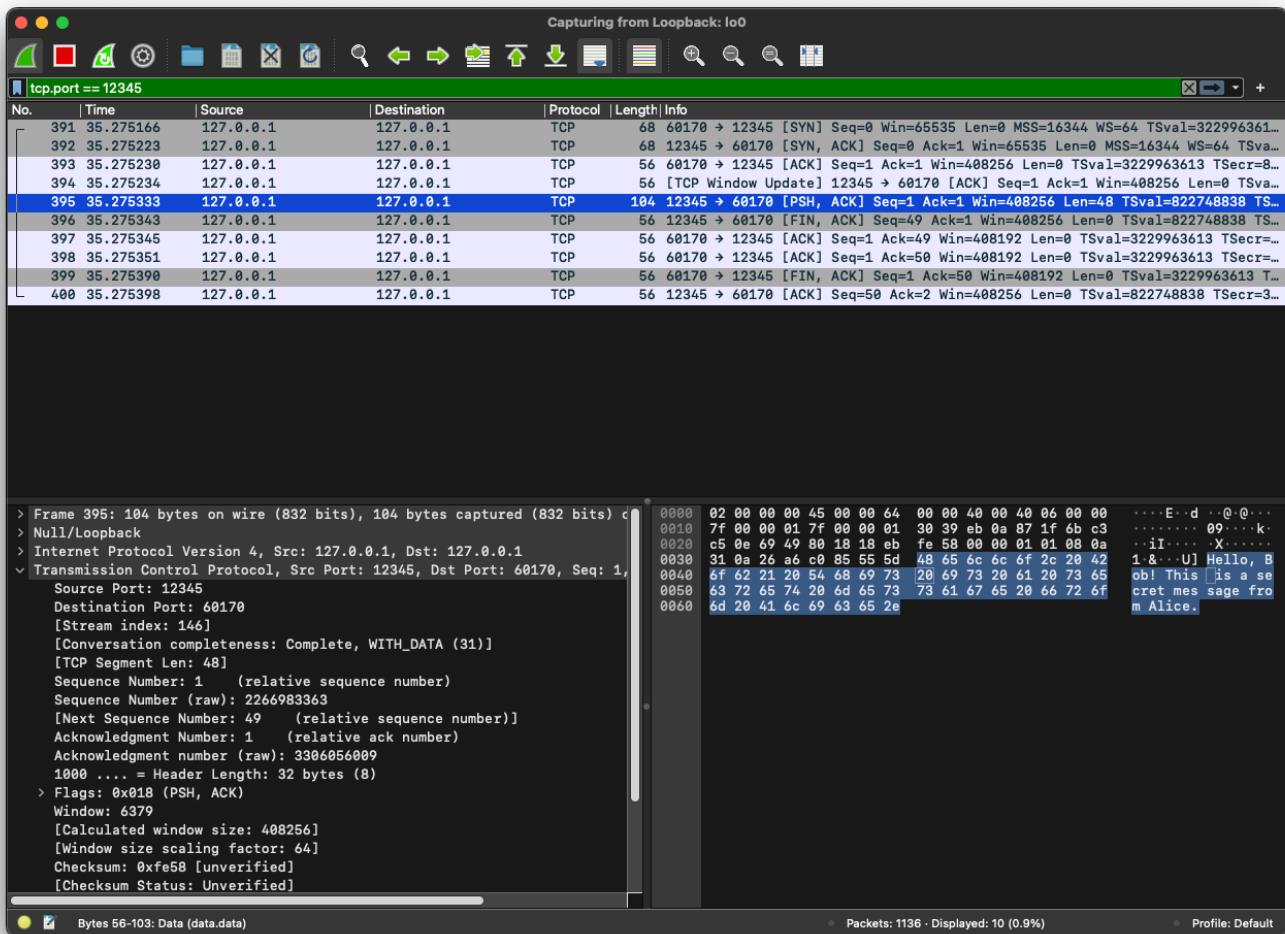
Puis, BadBob a démarré la capture wireshark. Par chance, il a réussi à savoir que le port d'écoute entre Bob et Alice était 12345.



Enfin, Bob et Alice ont démarré leur échange. Alice a donc créé le serveur, et a attendu que Bob s'y connecte. Par la suite, Bob s'est connecté et a reçu le message envoyé par Alice.

```
python3 sender.py
└─ thomas@Mac-mini-de-Thomas.local ~/Documents Serveur/Scolarité/EFREI
    └─ Cours/Cryptographie/Projet
        └─ python3 sender.py
            1 ↵
Waiting for Bob to connect...
└─ thomas@Mac-mini-de-Thomas.local ~/Documents Serveur/Scolarité/EFREI
    └─ Cours/Cryptographie/Projet
        └─ python3 sender.py
            130 ↵
Waiting for Bob to connect...
Bob connected!
└─ thomas@Mac-mini-de-Thomas.local ~/Documents Serveur/Scolarité/EFREI
    └─ Cours/Cryptographie/Projet
        └─ python3 receiver.py
            130 ↵
Received Message (Bob): Hello, Bob! This is a secret message from Alice.
└─ thomas@Mac-mini-de-Thomas.local ~/Documents Serveur/Scolarité/EFREI
    └─ Cours/Cryptographie/Projet
```

BadBob a pu capturer les paquets, et en voici la trace sur Wireshark :



Capturing from Loopback: lo0

tcp.port == 12345

No.	Time	Source	Destination	Protocol	Length	Info
391	35.275166	127.0.0.1	127.0.0.1	TCP	68	60170 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=16344 WS=64 TSval=322996361...
392	35.275223	127.0.0.1	127.0.0.1	TCP	68	12345 → 60170 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=16344 WS=64 TSval=3229963613 TSecr=8...
393	35.275230	127.0.0.1	127.0.0.1	TCP	56	60170 → 12345 [ACK] Seq=1 Ack=1 Win=408256 Len=0 TSval=3229963613 TSecr=8...
394	35.275234	127.0.0.1	127.0.0.1	TCP	56	[TCP Window Update] 12345 → 60170 [ACK] Seq=1 Ack=1 Win=408256 Len=0 TSval=3229963613 TSecr=8...
395	35.275333	127.0.0.1	127.0.0.1	TCP	104	12345 → 60170 [PSH, ACK] Seq=1 Ack=1 Win=408256 Len=48 TSval=822748838 TSecr=...
396	35.275343	127.0.0.1	127.0.0.1	TCP	56	12345 → 60170 [FIN, ACK] Seq=49 Ack=1 Win=408256 Len=0 TSval=822748838 TSecr=...
397	35.275345	127.0.0.1	127.0.0.1	TCP	56	60170 → 12345 [ACK] Seq=1 Ack=49 Win=408192 Len=0 TSval=3229963613 TSecr=...
398	35.275351	127.0.0.1	127.0.0.1	TCP	56	60170 → 12345 [ACK] Seq=1 Ack=50 Win=408192 Len=0 TSval=3229963613 TSecr=...
399	35.275398	127.0.0.1	127.0.0.1	TCP	56	60170 → 12345 [FIN, ACK] Seq=1 Ack=50 Win=408192 Len=0 TSval=3229963613 TSecr=...
400	35.275398	127.0.0.1	127.0.0.1	TCP	56	12345 → 60170 [ACK] Seq=50 Ack=2 Win=408256 Len=0 TSval=822748838 TSecr=3...

```
> Frame 395: 104 bytes on wire (832 bits), 104 bytes captured (832 bits)
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
└ Transmission Control Protocol, Src Port: 12345, Dst Port: 60170, Seq: 1,
    Source Port: 12345
    Destination Port: 60170
    [Stream index: 146]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 48]
    Sequence Number: 1      (relative sequence number)
    Sequence Number (raw): 2266983363
    [Next Sequence Number: 49      (relative sequence number)]
    Acknowledgment Number: 1      (relative ack number)
    Acknowledgment number (raw): 3306056009
    1000 .... = Header Length: 32 bytes (8)
    Flags: 0x018 (PSH, ACK)
    Window: 6379
    [Calculated window size: 408256]
    [Window size scaling factor: 64]
    Checksum: 0xfe58 [unverified]
    [Checksum Status: Unverified]
```

Bytes 56-103: Data (data.data)

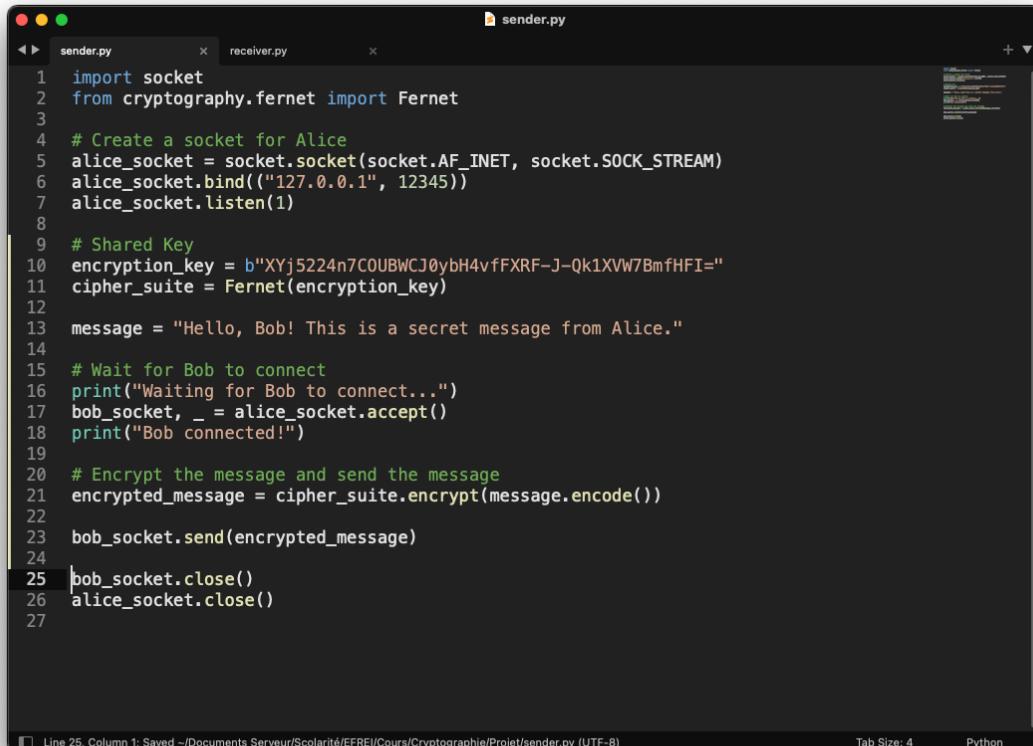
Packets: 1136 - Displayed: 10 (0.9%)

Profile: Default

Nous pouvons remarquer que, sur la capture ci-dessus, le paquet sélectionné contient le message envoyé par Alice. BadBob a donc réussi à intercepter le message.

Seconde approche : Utilisation du chiffrement de bout en bout

Nous nous retrouvons dans un scénario identique, voici les fichiers `sender.py` et `receiver.py`, modifiés pour intégrer une solution de chiffrement de bout en bout. Nous utiliserons une clé partagée définie en statique dans le code.

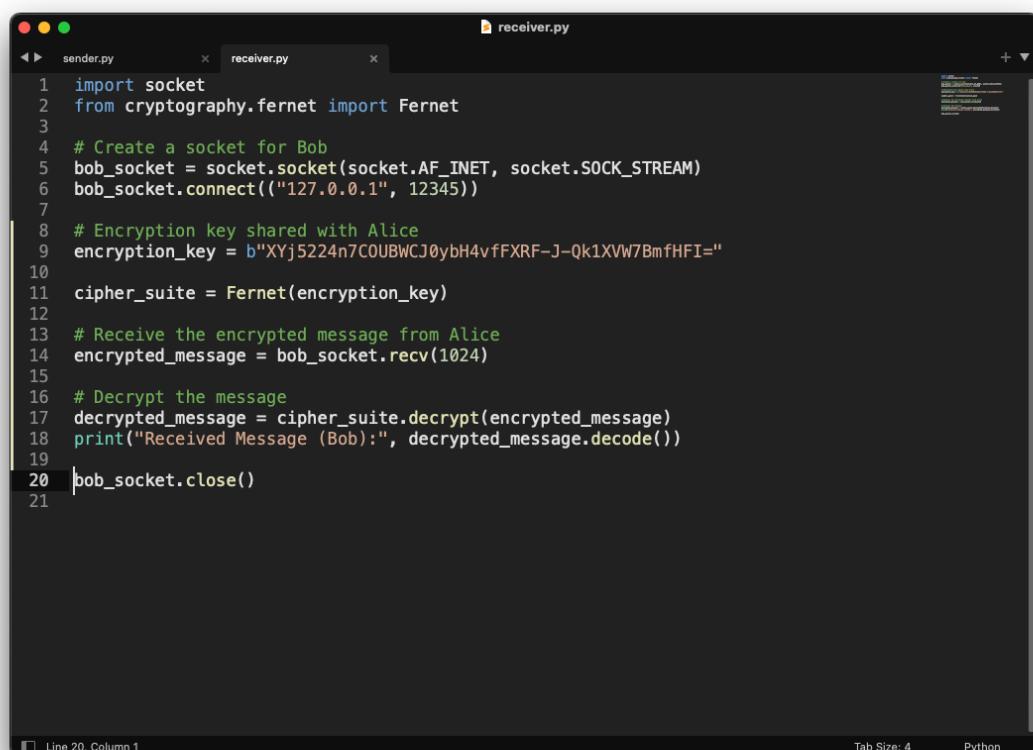


```

1 import socket
2 from cryptography.fernet import Fernet
3
4 # Create a socket for Alice
5 alice_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
6 alice_socket.bind(("127.0.0.1", 12345))
7 alice_socket.listen(1)
8
9 # Shared Key
10 encryption_key = b"XYj5224n7COUBWCJ0ybH4vfFXRF-J-Qk1XVW7BmfHFI="
11 cipher_suite = Fernet(encryption_key)
12
13 message = "Hello, Bob! This is a secret message from Alice."
14
15 # Wait for Bob to connect
16 print("Waiting for Bob to connect...")
17 bob_socket, _ = alice_socket.accept()
18 print("Bob connected!")
19
20 # Encrypt the message and send the message
21 encrypted_message = cipher_suite.encrypt(message.encode())
22
23 bob_socket.send(encrypted_message)
24
25 bob_socket.close()
26 alice_socket.close()
27

```

Line 25, Column 1; Saved ~/Documents/Serveur/Scolarité/EFREI/Cours/Cryptographie/Projet/sender.py (UTF-8) Tab Size: 4 Python



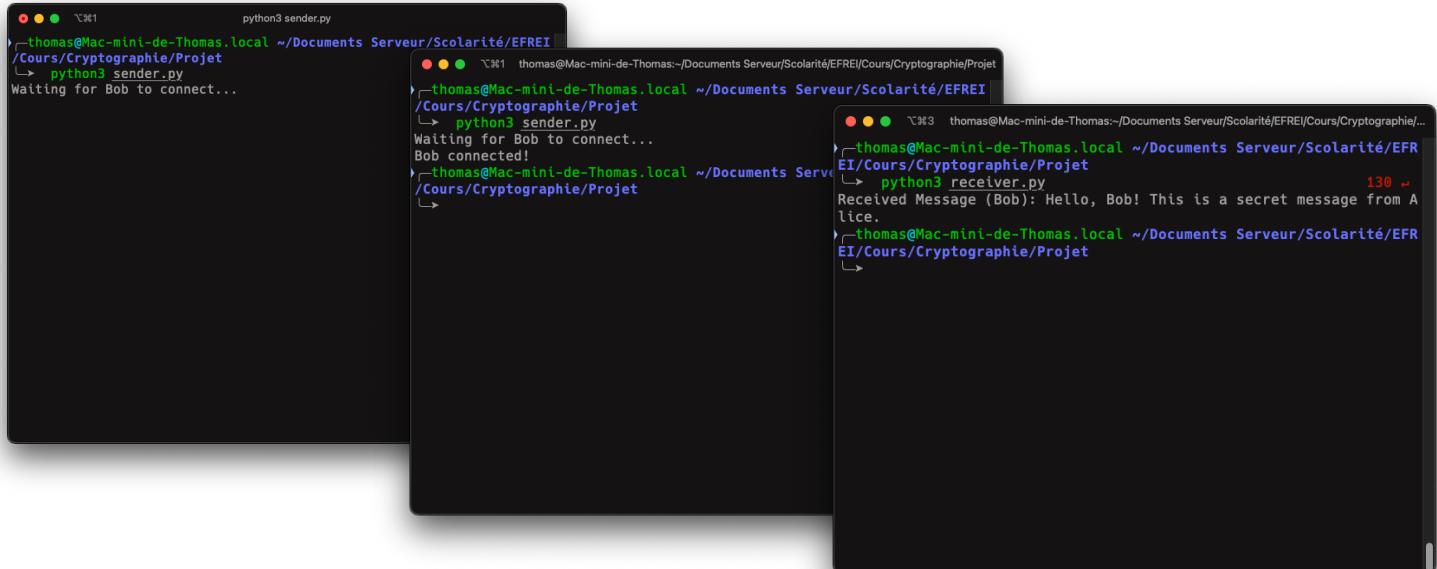
```

1 import socket
2 from cryptography.fernet import Fernet
3
4 # Create a socket for Bob
5 bob_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
6 bob_socket.connect(("127.0.0.1", 12345))
7
8 # Encryption key shared with Alice
9 encryption_key = b"XYj5224n7COUBWCJ0ybH4vfFXRF-J-Qk1XVW7BmfHFI="
10
11 cipher_suite = Fernet(encryption_key)
12
13 # Receive the encrypted message from Alice
14 encrypted_message = bob_socket.recv(1024)
15
16 # Decrypt the message
17 decrypted_message = cipher_suite.decrypt(encrypted_message)
18 print("Received Message (Bob):", decrypted_message.decode())
19
20 bob_socket.close()
21

```

Line 20, Column 1 Tab Size: 4 Python

Par la suite, Alice et Bob ont relancé leurs programmes pour s'envoyer et recevoir un message.

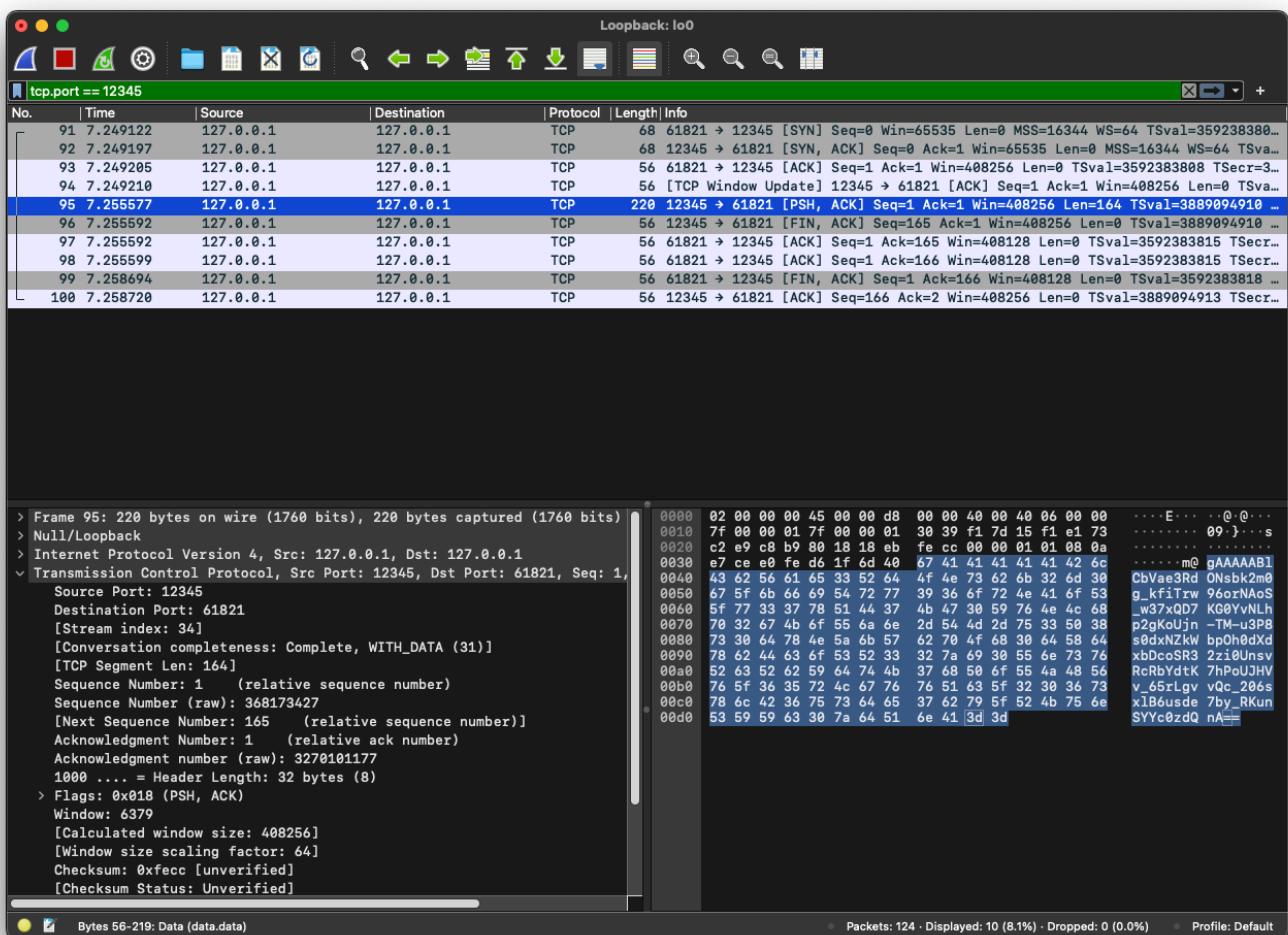


```

thomas@Mac-mini-de-Thomas.local ~/Documents Serveur/Scolarité/EFREI
~/Cours/Cryptographie/Projet
└─> python3 sender.py
Waiting for Bob to connect...
thomas@Mac-mini-de-Thomas.local ~/Documents Serveur/Scolarité/EFREI
~/Cours/Cryptographie/Projet
└─> python3 sender.py
Waiting for Bob to connect...
Bob connected!
thomas@Mac-mini-de-Thomas.local ~/Documents Serveur/Scolarité/EFREI
~/Cours/Cryptographie/Projet
└─>
thomas@Mac-mini-de-Thomas.local ~/Documents Serveur/Scolarité/EFREI
~/Cours/Cryptographie/Projet
└─> python3 receiver.py
Received Message (Bob): Hello, Bob! This is a secret message from A
lice.
thomas@Mac-mini-de-Thomas.local ~/Documents Serveur/Scolarité/EFREI
~/Cours/Cryptographie/Projet
└─>

```

Cette fois-ci à nouveau, BadBob a réussi à intercepter le message transmis. En voici la capture Wireshark.



No.	Time	Source	Destination	Protocol	Length	Info
91	7.249122	127.0.0.1	127.0.0.1	TCP	68	61821 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=16344 WS=64 TSval=359238380...
92	7.249197	127.0.0.1	127.0.0.1	TCP	68	12345 → 61821 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=16344 WS=64 TSva...
93	7.249205	127.0.0.1	127.0.0.1	TCP	56	61821 → 12345 [ACK] Seq=1 Ack=1 Win=408256 Len=0 TSval=3592383808 TSecr=3...
94	7.249210	127.0.0.1	127.0.0.1	TCP	56	[TCP Window Update] 12345 → 61821 [ACK] Seq=1 Ack=1 Win=408256 Len=0 TSva...
95	7.255577	127.0.0.1	127.0.0.1	TCP	220	12345 → 61821 [PSH, ACK] Seq=1 Ack=1 Win=408256 Len=164 TSval=3889094910 ...
96	7.255592	127.0.0.1	127.0.0.1	TCP	56	12345 → 61821 [FIN, ACK] Seq=165 Ack=1 Win=408256 Len=0 TSval=3889094910 ...
97	7.255592	127.0.0.1	127.0.0.1	TCP	56	61821 → 12345 [ACK] Seq=1 Ack=165 Win=408128 Len=0 TSval=3592383815 TSecr...
98	7.255599	127.0.0.1	127.0.0.1	TCP	56	61821 → 12345 [ACK] Seq=1 Ack=166 Win=408128 Len=0 TSval=3592383815 TSecr...
99	7.258694	127.0.0.1	127.0.0.1	TCP	56	61821 → 12345 [FIN, ACK] Seq=1 Ack=166 Win=408128 Len=0 TSval=3592383818 ...
100	7.258720	127.0.0.1	127.0.0.1	TCP	56	12345 → 61821 [ACK] Seq=166 Ack=2 Win=408256 Len=0 TSval=3889094913 TSecr...

Frame 95: 220 bytes on wire (1760 bits), 220 bytes captured (1760 bits)
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 12345, Dst Port: 61821, Seq: 1, Source Port: 12345
Destination Port: 61821
[Stream index: 34]
[Conversation completeness: Complete, WITH_DATA (31)]
[TCP Segment Len: 164]
Sequence Number: 1 (relative sequence number)
Sequence Number (raw): 368173427
[Next Sequence Number: 165 (relative sequence number)]
Acknowledgment Number: 1 (relative ack number)
Acknowledgment number (raw): 3270101177
1000 = Header Length: 32 bytes (8)
> Flags: 0x018 (PSH, ACK)
Window: 6379
[Calculated window size: 408256]
[Window size scaling factor: 64]
Checksum: 0xfeecc [unverified]
[Checksum Status: Unverified]

Bytes 56-219: Data (data.data)

Packets: 124 · Displayed: 10 (8.1%) · Dropped: 0 (0.0%) · Profile: Default

Nous pouvons, cette fois-ci, noter que le paquet contenant le message n'est plus lisible en clair, le message est donc devenu inconnu de BadBob.

Conclusion

L'end-to-end encryption (E2EE) a marqué une avancée significative dans la protection de la vie privée et de la sécurité des données dans un monde numérique en constante évolution. Elle a radicalement transformé la manière dont nous sécurisons nos communications en ligne et a renforcé la confiance des utilisateurs dans la confidentialité de leurs échanges.

Cependant, les défis techniques et les débats en cours sur l'équilibre entre la sécurité et les besoins des forces de l'ordre continueront à façonner l'avenir de l'E2EE. Il est essentiel de rester attentif aux développements futurs dans ce domaine en constante évolution pour garantir une sécurité et une confidentialité continues dans nos communications en ligne.

Enfin, par l'intermédiaire d'un simple exemple, nous avons pu comprendre les implications non-négligeables de l'utilisation de ce type de chiffrement.

Bibliographie

État de l'art

Wikipédia - https://fr.wikipedia.org/wiki/Chiffrement_de_bout_en_bout

Google Messages - <https://support.google.com/messages/answer/10252671?hl=fr>

Kaspersky - <https://www.kaspersky.fr/blog/what-is-end-to-end-encryption/15668/>

IBM - <https://www.ibm.com/fr-fr/topics/end-to-end-encryption>

BlogDuModérateur - <https://www.blogdumoderateur.com/chiffrement-bout-en-bout-e2ee/>

Section.io - <https://www.section.io/engineering-education/end-to-end-encryption/>

NordVPN - <https://nordvpn.com/fr/blog/what-is-end-to-end-encryption/>

Cas pratique

Wikipédia - https://en.wikipedia.org/wiki/Network_socket

GeeksForGeeks - <https://www.geeksforgeeks.org/socket-in-computer-network/>

Oracle - <https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html>

Python - <https://docs.python.org/3/library/socket.html>

Python Cryptography - <https://pypi.org/project/cryptography/>