

E2EE

[End-to-End Encryption] for iOS Developer.

mihail@gerasimenko.me

Introduction.

- Graduated Master of Applied Mathematics at Odessa State I.I.Mechnikov University.
- 9 years in iOS development.
- 5 years berliner.
- Proud father of Alexander and Daniel.

Motivation.

Why I am giving this talk?

- Help devs understand what the end-to-end encryption is exactly about.
- Motivate to think about privacy when implementing the apps.
- Inspire to create the new E2EE apps.



¹ Image copyright: Wired magazine, all rights reserved.

mihail@gerasimenko.me



² Image copyright: I am sure it is copyrighted.

mihail@gerasimenko.me

WIRE FOR WORKGROUPS™

Version 3.11



About Wire.

- I am the part of the awesome iOS team at Wire.
- Wire is open-source 🎉: <https://github.com/wireapp/wire-ios> and other repos.
- Wire is one of the pioneers of End-to-End encryption: first version released in #TODO.
 - Wire is available as the encrypted Slack replacement.

Disclaimer.

- I am not the inventor of E2EE.
- I am not responsible for the design of the encryption at Wire.

Prologue: The Invention.



mihail@gerasimenko.me



Hoplite with their aspis³

A hoplite was primarily a free citizen who was responsible for procuring his armour and weapon.

Many famous personalities, philosophers, artists and poets fought as hoplites.

³ Public domain, Either Edward J. Krasnoborski or F. Mitchell. - [website](#).

Bottomline.

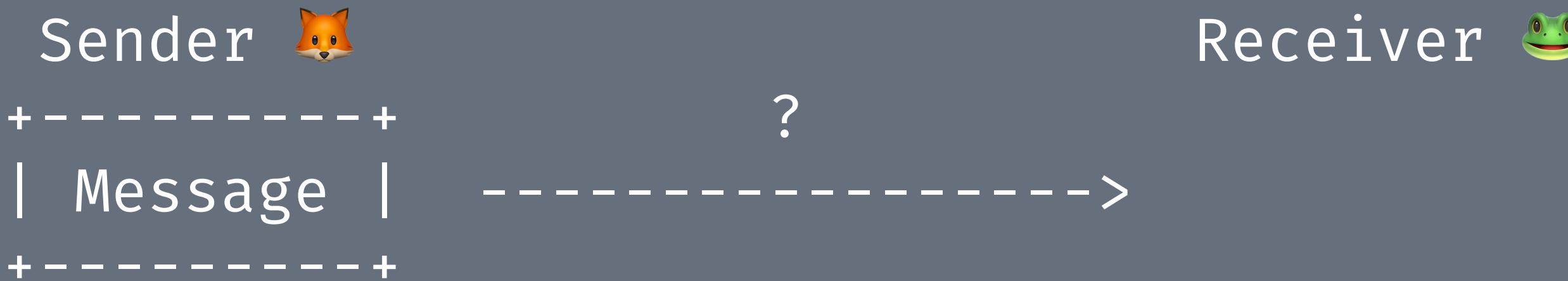
**Forbidding encrypted messaging would
not work.**

**It is the invention and invention is not
possible to forbid. TODO**



**So what is
End-To-End encryption,
exactly?**

Defining the problem.



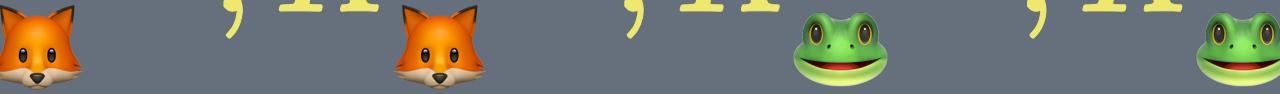
Defining the problem: Confidentiality.



Solution 🎉: Public-key crypto: Diffie-Hellman (DH) key exchange or RSA.

Sender 🦊 <----- Public keys -----> Receiver 🐸

1. Sender and receiver generate public and private key: $K_{\text{fox}}^{Publ}, K_{\text{fox}}^{Priv}, K_{\text{frog}}^{Publ}, K_{\text{frog}}^{Priv}$



2. They exchange public keys:  → ,

K_{fox}^{Publ}

Public-key crypto.

Using the Diffie-Hellman (DH) procedure, the shared secret key is created: $K_{\text{Fox}}^{\text{Shared}} = DH(K_{\text{Frog}}, K_{\text{Fox}})$

Secure channel
Sender  <---- encrypted with the ----> Receiver 
shared secret key

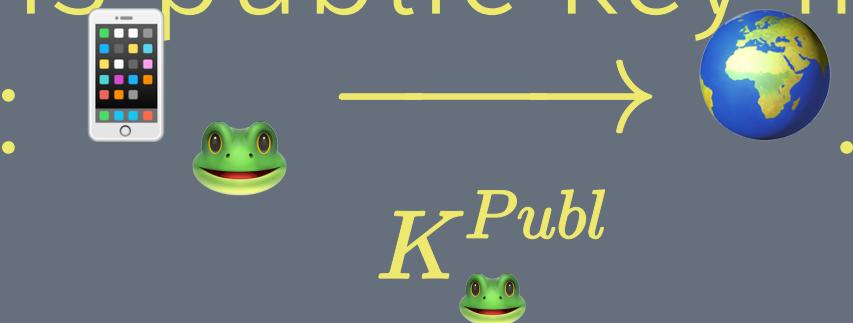
This is how TLS (and HTTPS over it) is working.

Problem 1: Receiver not online !

- Using the DH or RSA, both participants must be online in order to perform the key exchange.
- Not possible for reasons: phone or other device is not online.

Solution.

- Receiver can publish his public key in advance to the server:



- Next time someone wants to communicate with him it is possible to fetch the public key from the server:



In Wire

K^{Publ}



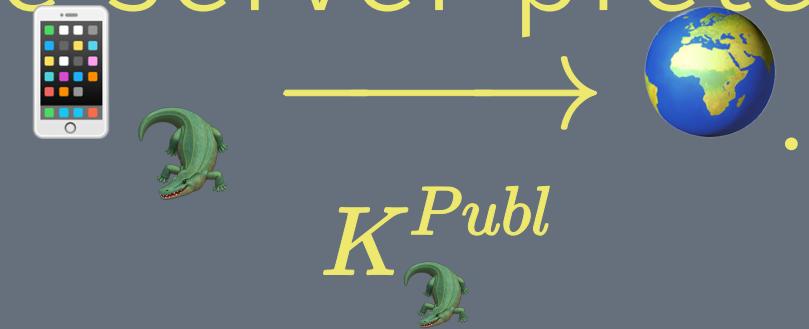
is called the Prekey.

Problem 2: Authenticity !

→ What if someone generates another key pair:

$$K_{\text{alligator}}^{\textit{Priv}}, K_{\text{alligator}}^{\textit{Publ}}$$

→ Upload it to the server pretending he is 🐸:



$$K_{\text{alligator}}^{\textit{Publ}}$$

Problem 2: Authenticity !

- Anyone who would like to talk with  will actually create the shared secret with !
- Then  can decide to create the shared secret with  and relay the messages reading them.

Problem 2: Authenticity !

→ This is called ~~crocodile~~man-in-the-middle attack.



<-- Secure channel -->



<-- Secure channel -->



Solution: Key Verification .

- It is possible to sign the key with another private key.
- In HTTPS: the authority signatures (public keys) are saved in the keychain.
- It is possible therefore to check the signature.
 - In messaging: users must check the key fingerprints of the people they are communicating

In Wire
it is called
the fingerprint
verification.

Problem 3: Forward secrecy.

→ If the  would record all the encrypted communication between  and ...

→ And then find out the K^{Shared} .



→ All the previous communication can be decrypted.

Solution: Session keys / Key rotation.

- Generate the new key for each message.
- Rotate the key using the Hash Key Derivation Function (HKDF) - basically hash the previous key:
$$K_n = \text{HKDF}(K_{n-1})$$

- It is not possible to find out key K_{n-1} from K_n .


Problem 4: Backward / Future secrecy.

- If 🐊 would find out the K^n ...
🦊🐸...
- He can hash it, too, and find out the next key $n + 1$.

Solution: DH re-negotiation, mixing the initial session keys with the new ones.

- Every message we send, we also include the new public key signed with our initial session key.
- Receiver also generate another key pair and restart the session.

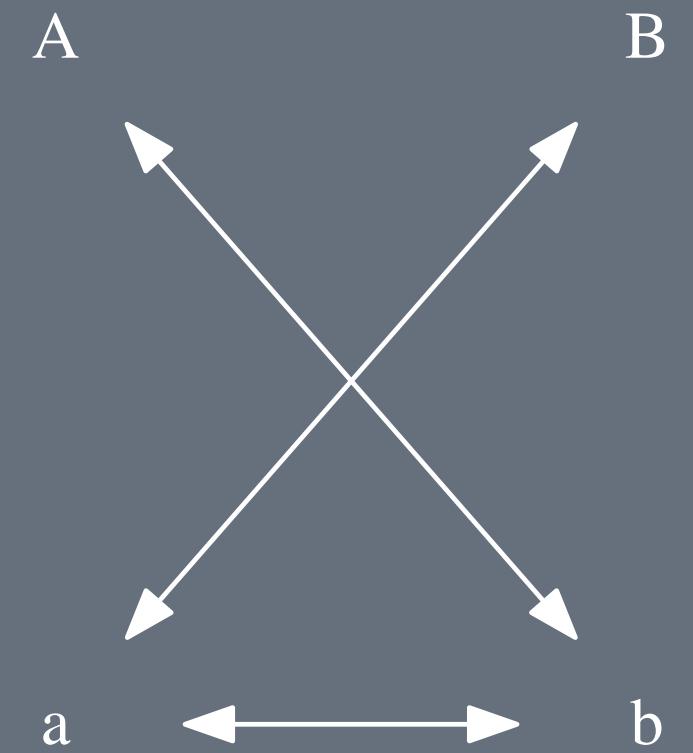
Problem 5: It's not me i.e. *Plausible Deniability.*

- Remember problem 2: authenticity?
- It is good to check if the person talking to you is the one you want to talk to.
- It is also possible to say that only the sender can sign the message, since only sender has his private key.

Plausible Deniability

- So what's the issue here?
- If the messages are getting compromised, it is possible to associate the message with the sender.
- Can be the proof in the court : message is fingerprinted by the sender!

**Solution: Three-way
initial DH key
negotiation.**



$$K_{master} = \text{hash}(dh(A, b), dh(B, a), dh(a, b))$$

Why good E2EE was not available earlier?

- No good protocol to solve problems 3-5.
- The performance of the keypair K_{Publ} , K_{Priv}
 generation improved dramatically, since:
 - Elliptic curve crypto development.
 - Mobile CPUs are way faster nowadays.

Recap: good E2EE is:

1. Confidentiality.
2. Authenticity.
3. Forward Secrecy.
4. Backward (Future) Secrecy.
5. Plausible Deniability.

Wire protocol

that solves problems 1-5

is called

Axolotl/Double Ratchet.

How it applies to iOS.

- Basics of not sharing the data with Apple.
- Push Notifications (APNs) 💔 E2EE.
- Share extension + E2EE.

Basics of not sharing the data with Apple.

- Apple cares about user privacy.
- iTunes and iCloud backups.
- CallKit .

Image and Video metadata.

Every image or video taken on the iPhone has a significant amount of embedded metadata:

- Device location .
- Model .
- Camera information .

Strip metadata using ImageIO.

Load image from Data to imageSource ⁴:

```
guard let imageSource = CGImageSourceCreateWithData(data, nil),  
    let type = CGImageSourceGetType(imageSource) else {  
throw MetadataError.unknownFormat  
}
```

⁴ Source: <https://github.com/wireapp/wire-ios-images/blob/develop/Sources/Image%20Processing/NSData+MediaMetadata.swift>

Strip metadata using ImageIO.

Create the new image `imageDestination`:

```
let count = CGImageSourceGetCount(imageSource)
let mutableData = NSMutableData(data: self as Data)
guard let imageDestination = CGImageDestinationCreateWithData(mutableData,
                                                               type,
                                                               count,
                                                               nil) else {
    throw MetadataError.cannotCreate
}
```

Strip metadata using ImageIO.

Reset the metadata:

```
for sourceIndex in 0..
```

iTunes and iCloud backups.

- The content of the backup is stored plaintext in the iTunes or (even worse) in the iCloud.
- Since we care not to put user data on our backend, we also have to care not to put it on the Apple backend.

iTunes and iCloud backups.

Like that⁵:

```
var resourceValues = URLResourceValues()  
resourceValues.isExcludedFromBackup = true  
try mutableURL.setResourceValues(resourceValues)
```

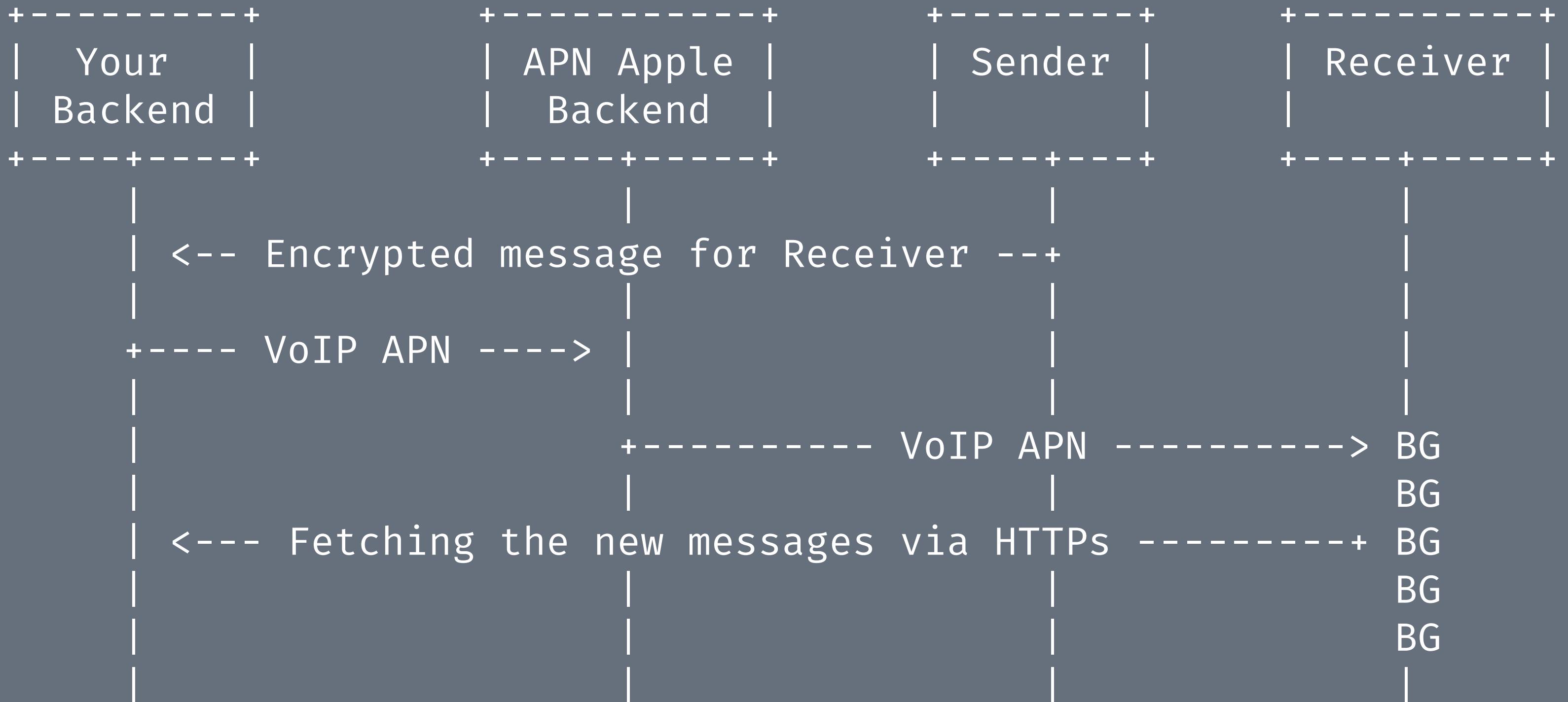
⁵ Source: <https://github.com/wireapp/wire-ios/blob/develop/WireExtensionComponents/Utilities/URL%2BBackup.swift>

CallKit.

- CallKit used to sync the calls metadata between the devices.
 - iOS 11 fixed that.

Push Notifications (APNs) 💔 E2EE.

- The message sent via the push notification is visible to Apple.
- APNs allows sending the "VoIP Push Notifications".
- Using **VoIP** push, the iOS application can run the code while not active.
- During this time, the application must fetch the message from the backend and display the local



Problems.

- If the client cannot manage to fetch the message content from the backend in time, the push notifications are not going to be delivered.
- If the push notification receiver is offline, then the push notification scheduler can drop some notifications, so the client would not have a chance to fetch messages.

Possible solution: Background Fetch.

- It is possible to enable the background fetching, so when the device comes online the app would have the chance to fetch messages.

```
application.setMinimumBackgroundFetchInterval(timeInterval)
```

Share Extensions 💔💔💔 E2EE.

- On iOS, the share extension is the separate process.
- Database and the crypto material must be moved to the shared container.
- File sync is necessary: read up here ⁶.

⁶ <https://medium.com/@wireapp/the-challenge-of-implementing-ios-share-extension-for-end-to-end-encrypted-messenger-dd33b52b1e97>

Big chats 🤕 E2EE.

- The message must be delivered to each participant.
 - To the each participant's device!
- So when you send one message, say 1 Kilobyte of data in the conversation with N participants where each participant has K devices you actually have to send $1\text{Kilobyte} \times N \times K$ messages.

Is it worth it?

As every tech out there, E2EE has its pros and cons.

- Harder to implement.
 - Need to think.
 - Less points of failure.

Points of failure.

- Probability of the data leak $0 < P(\text{漏水}) \ll 1$.
- For N points: total probability is $\bigcup_{i=1}^N P(\text{漏水}_i) = P_{\text{漏水}}$.
- Let's remove the point N : $\bigcup_{i=1}^{N-1} P(\text{漏水}_i) = P'_{\text{漏水}}$.
- $P_{\text{漏水}} > P'_{\text{漏水}}$.

Sneak peek: MLS.

- MLS stands for the message Messaging Layer Security.
- IETF initiative to develop the common protocol for secure instant messaging.
- The protocol is being developed in cooperation between IETF, Twitter, Mozilla, Google, Facebook and Wire.

MLS: Stay tuned!

- Improves the message sending in the big (>100) group conversations.
- One standard that can potentially unify the different messengers.
- <https://www.ietf.org/mailman/listinfo/MLS>
- <https://datatracker.ietf.org/doc/draft-omara-mls-architecture/>

Thanks!

Wire Security Whitepaper



github.com/mikeger



twitter.com/GerasimenkoMiha



CC BY 4.0