E2EE

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

mike@wire.com

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.

mike@wire.com

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.

mike@wire.com

I

[1] Image copyright: Wired magazine, all rights reserved.

mike@wire.com

2

[2] Image copyright: I am sure it is copyrighted.

mike@wire.com

WIRE
FOR WORKGROUPS™

Version 3.11

mike@wire.com

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

Disclaimer.

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

mike@wire.com

Prologue: The Invention.

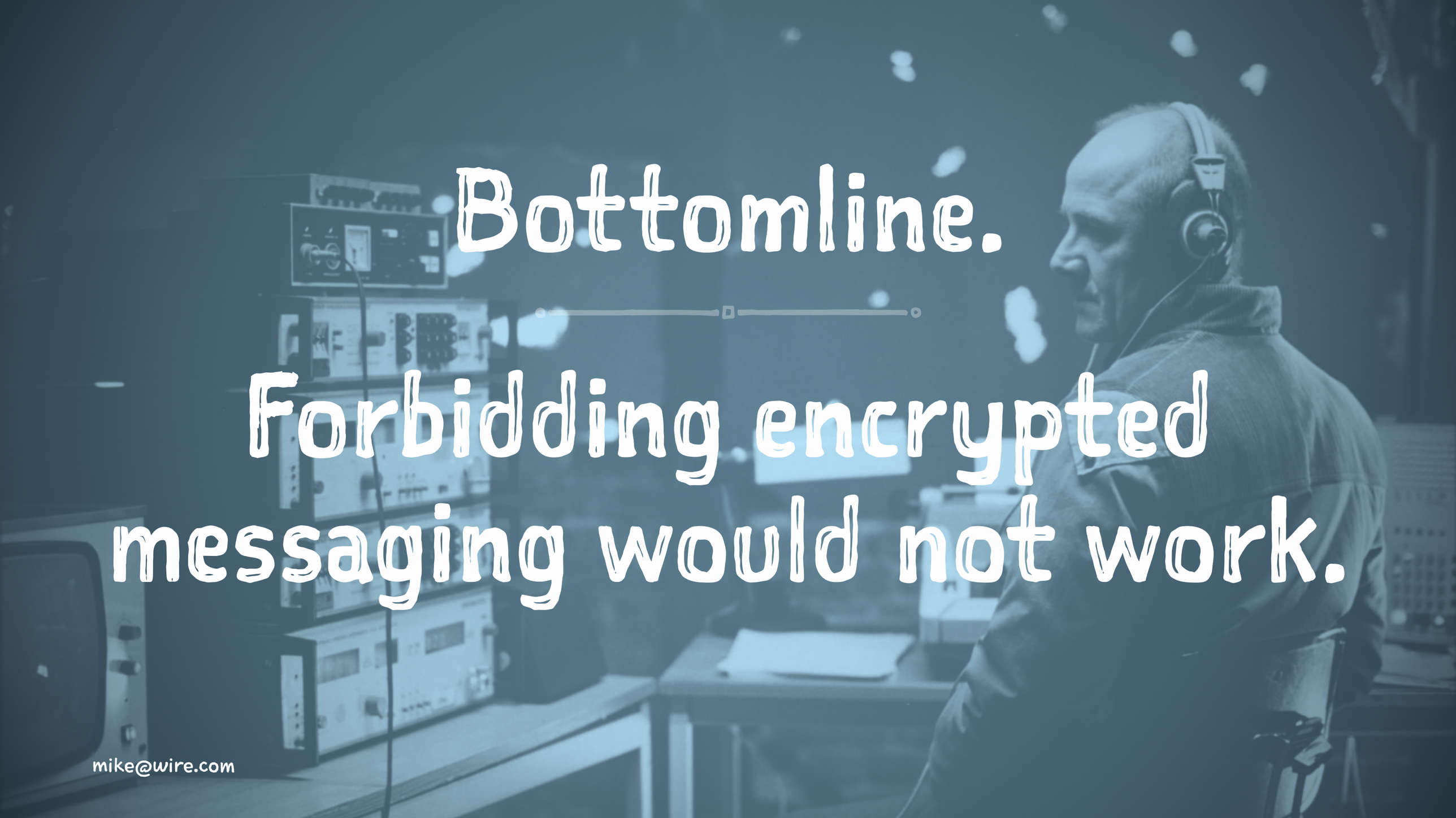
mike@wire.com

Hoplite with their aspis [3]

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.

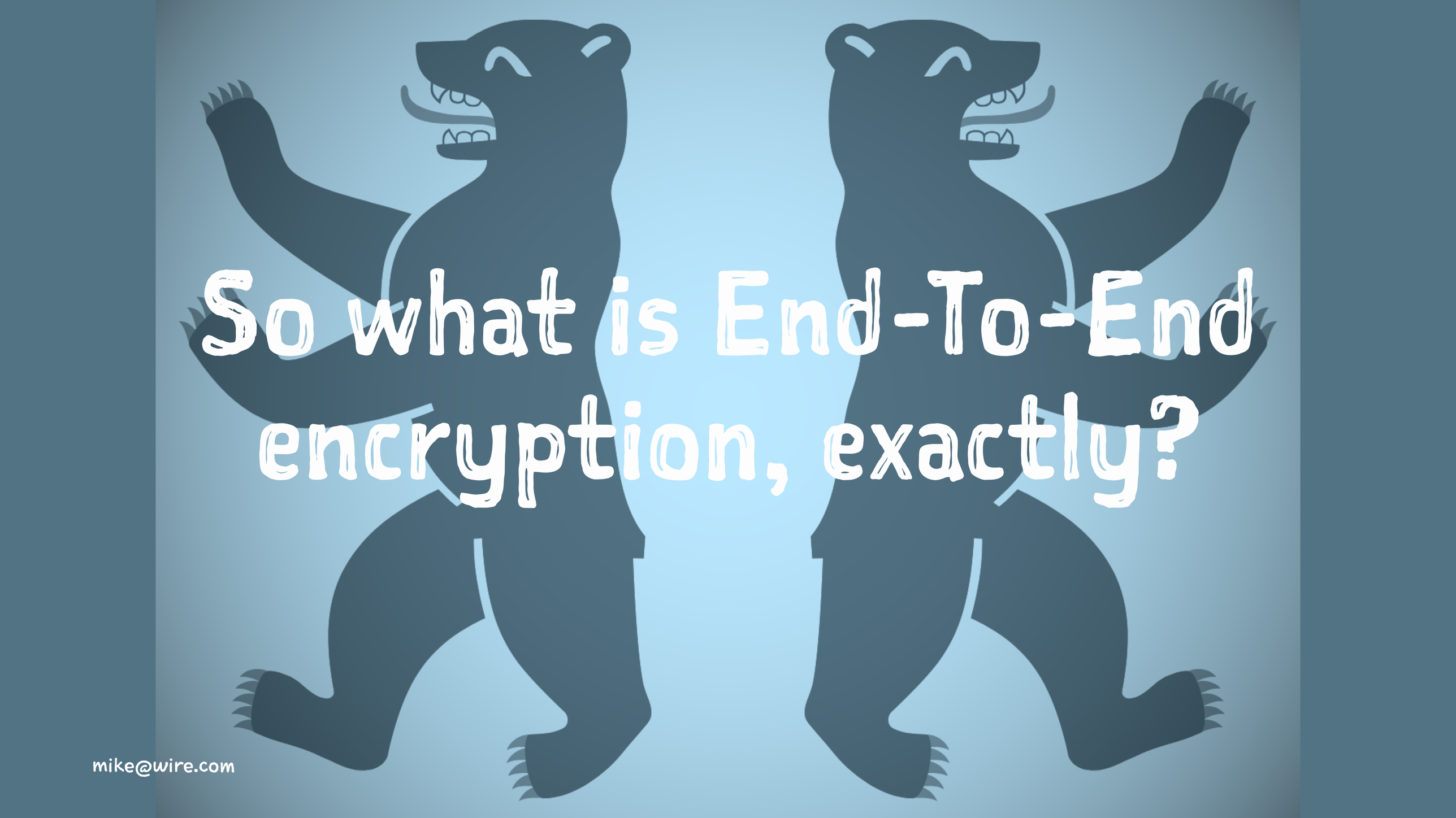
mike@wire.com



Bottomline.

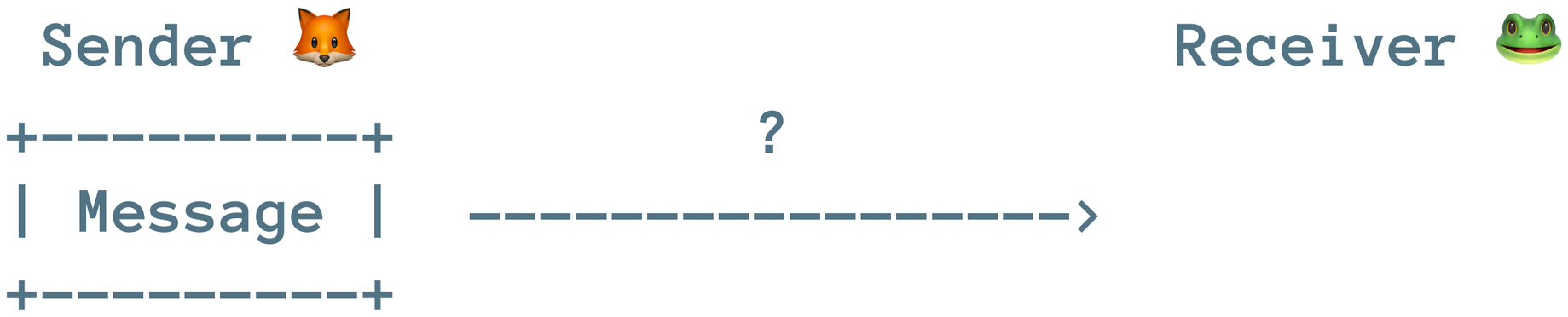
Forbidding encrypted messaging would not work.

mike@wire.com



mike@wire.com

Defining the problem.



mike@wire.com

Defining the problem.

mike@wire.com

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

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

1. Sender and receiver generate public and private key:
   $K_{🦊}^{Publ}, K_{🦊}^{Priv}, K_{🐸}^{Publ}, K_{🐸}^{Priv}$
2. They exchange public keys: 📱🦊 $\xrightarrow[K_{🦊}^{Publ}]{}$ 🐸, 📱🐸 $\xrightarrow[K_{🐸}^{Publ}]{}$ 🦊

mike@wire.com

Public-key crypto.

Using the Diffie-Hellman (DH) procedure, the shared secret key is created: $K^{Shared}_{🦊🐸}$

This is how TLS work.

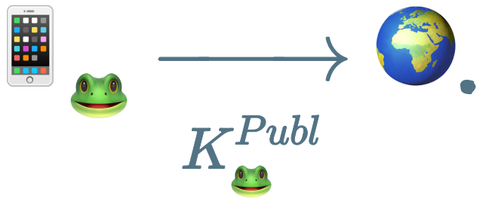
mike@wire.com

Problem 1: Reachability ⚠️

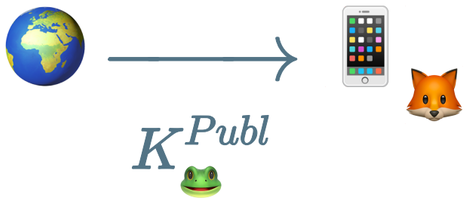
- Using the DH or RSA, both participants must be online in order to perform the key exchange.
- Not possible for reasons: phone or other device is not online.

mike@wire.com

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: 

.

mike@wire.com

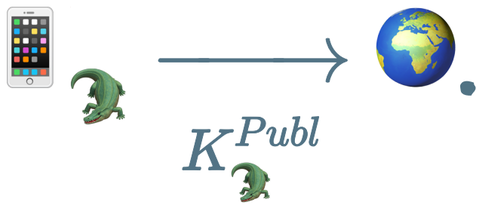
In Wire K^{Publ}  is called the

Prekey.

mike@wire.com

Problem 2: Credibility ⚠️

- What if someone generates another key pair: $K^{Priv}_{🐉}, K^{Publ}_{🐉}$.
- Upload it to the server pretending he is 🐸:



mike@wire.com

Problem 2: Credibility ⚠️

- 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.

mike@wire.com

Problem 2: Credibility ⚠️

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

🦊 <-- Secure channel --> 🐊 <-- Secure channel --> 🐸

mike@wire.com

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 fingerpints of the people they are communicating with.

mike@wire.com

In Wire it is called the fingerprint verification.

mike@wire.com

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.

mike@wire.com

Solution: Session keys / Key rotation.

- Generate the new key for each message:
- Either do the new key exchange while exchanging the messages.
- When not possible (no messages coming back): rotate the key using the Hash Key Derivation Function (HKDF) - basically hash the previous key.

mike@wire.com

Why good E2EE was not available earlier?

- The performance of the keypair $K^{Publ}_{🦊}$, $K^{Priv}_{🦊}$ generation improved dramatically, since:
- Elliptic curve crypto development.
- CPU performance improvements.

mike@wire.com

How it applies to iOS.

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

mike@wire.com

Basics of not sharing the data with Apple.

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

mike@wire.com

Image and Video metadata.

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

- Device location 📍.
- Model 📱.
- Camera information 📷.

mike@wire.com

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

mike@wire.com

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
}
```

mike@wire.com

Strip metadata using ImageIO.

Reset the metadata:

```
for sourceIndex in 0..<count {
    CGImageDestinationAddImageFromSource(imageDestination,
                                         imageSource,
                                         sourceIndex,
                                         nullMetadataProperties)
}

guard CGImageDestinationFinalize(imageDestination) else {
    throw MetadataError.cannotCreate
}
```

mike@wire.com

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.

mike@wire.com

iTunes and iCloud backups.

Like that[5]:

```
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

mike@wire.com

CallKit.

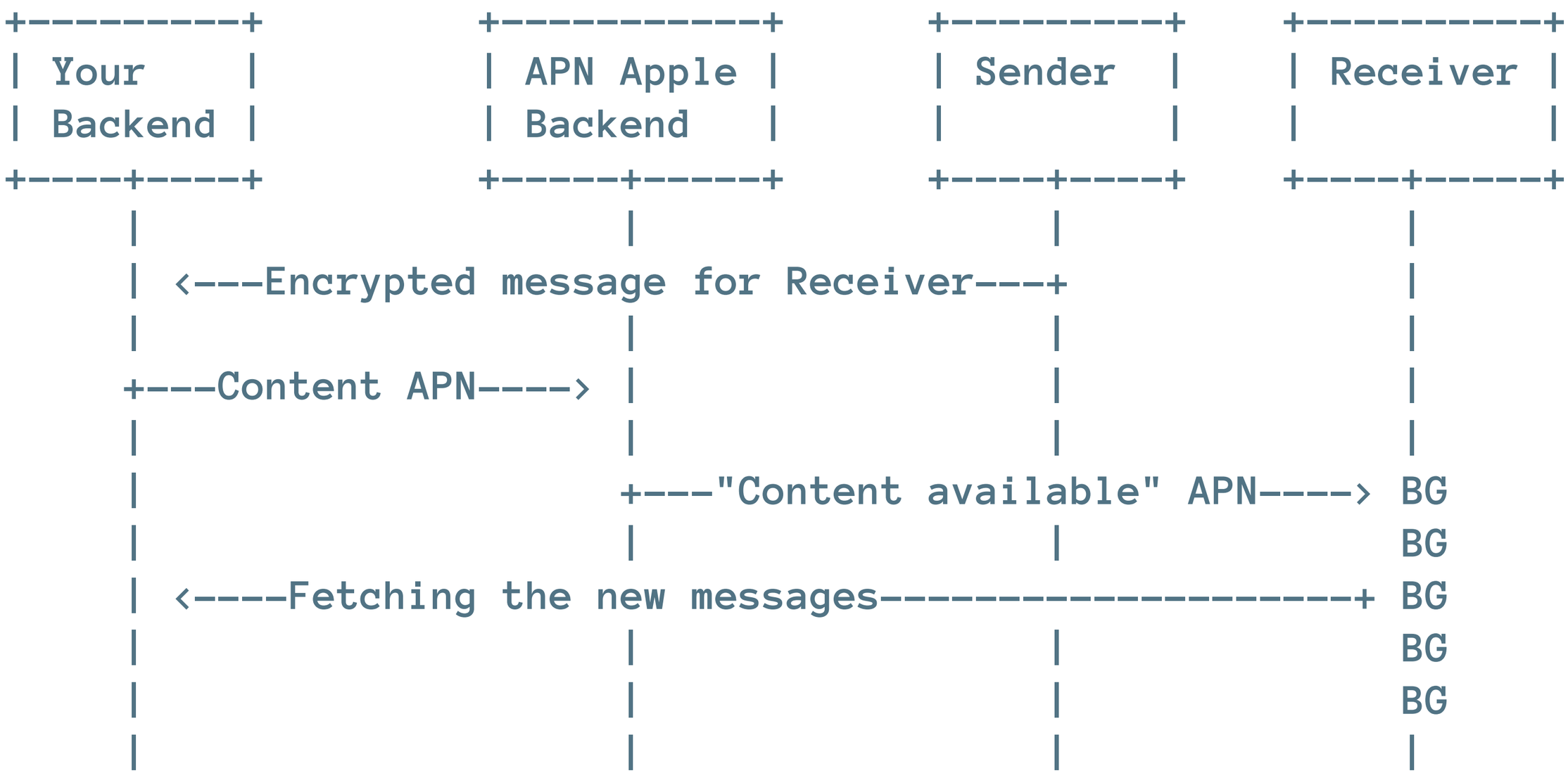
- CallKit used to sync the calls metadata between the devices.
- iOS 11 fixed that.

mike@wire.com

Push Notifications (APNs) 💔 E2EE.

- The message sent via the push notification is visible to Apple.
- APNs allows sending the "VoIP Puhs 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 push

mike@wire.com



mike@wire.com

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.

mike@wire.com

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)
```

mike@wire.com

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.

mike@wire.com

Big chats 🤕 E2EE.

- The message must be delivered to each participant.
- To the each participant's device!
- So when you send one message, say I Kilobyte of data in the conversation with N participants where each participant has K devices you actually have to send $1Kilobyte \times N\ timesK$ messages.
- 100 participants, each have 5 devices: **0.5 Megabyte.**

mike@wire.com

Is it worth it?

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

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

mike@wire.com

Points of failure.

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

mike@wire.com

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.

mike@wire.com

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/

mike@wire.com

Thanks!

Security Whitepaper

github.com/mikeger

twitter.com/GerasimenkoMiha

mike@wire.com