

A look into the Mobile Messaging Black Box

33rd Chaos Communication Congress #33c3

Roland Schilling @NerdingByDoing

Frieder Steinmetz @twillnix

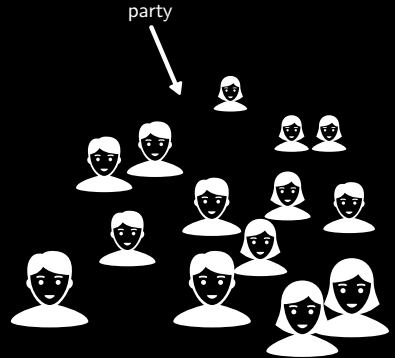
December 21, 2016

Hamburg University of Technology
Security in Distributed Applications

Messaging – Identifying Our Expectations

You're at a party

- Friend approaches you and needs to tell you something **in private**
- What do you expect when you say **private**?
- You enter a separate room, you trust the location
- What does a separate room offer you?

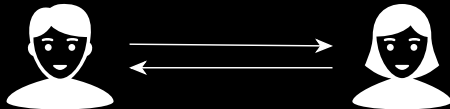


A Private Room

You are now alone in a closed room with your Friend

- Both of you have absolute Confidentiality that you are alone
- Nobody can overhear your talk
- Your exchange is completely private

We call this **confidentiality**



You Know Each Other

Since you're long-time friends, you're absolutely sure, whom you're talking to

- Nobody can impersonate your friend or you, without the other noticing
- You're talking directly, without a phone or webcam in between

We call this **authenticity**

In Sight of Each Other

The room you're in is small enough that you can always see each other

- You know that the words you speak are received just as you spoke them
- There is no way either of you hears something other than the other says

We call this **integrity**

It's a One-Time Talk

Suppose somebody steps into the room

- They could overhear your conversation
- They would only learn the contents of this particular conversation
- They would not learn anything about past conversations you had

We call this **forward secrecy**

→ After leaving they would not be able to listen to any future conversations you might have

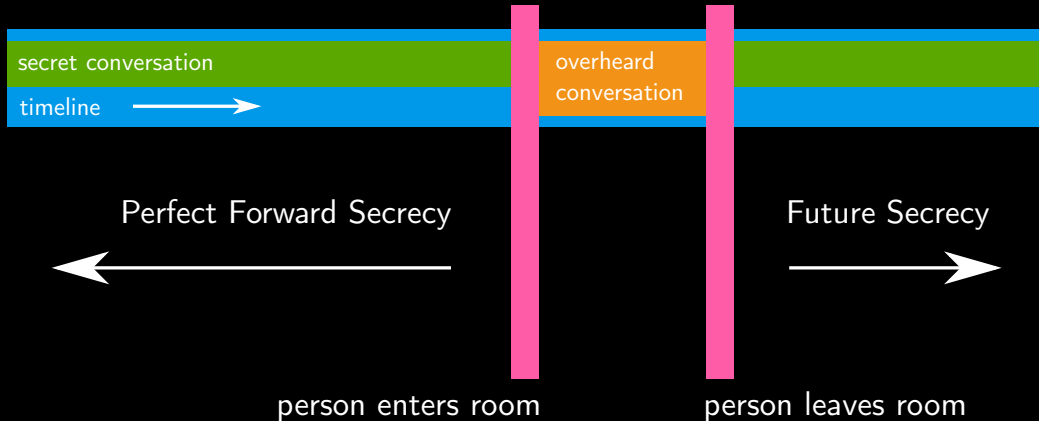
We call this **future secrecy**



It's a One-Time Talk



Perfect Forward- and Future Secrecy



It's a One-Time Talk Between Only You Two

There are no witnesses in the room

- Either of you can later deny to other having made any statement
- Neither of you can prove to other that any of you have made a particular statement

We call this **deniability**

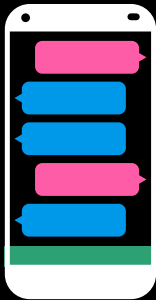
Messaging – Reality Check



Messaging – A More Technical Analogy

We started with a conversation analogy to identify our expectations of messaging

→ Actually **postal services** are better to look at messaging from a technical point of view.



Example: Traditional Messaging

What if our party conversation had taken place via SMS?



Your providers (and other people on the same network)

- would know the contents of your exchange: **no confidentiality**
- could change the contents of your exchange: **no integrity**
- could reroute your messages and impersonate either of you: **no authentication**
- would know all messages you ever exchanged: **no forward Secrecy**
- would know all messages exchanged in the future: **no future secrecy**
- could store all messages and use them as proof of the exchange: **no deniability**

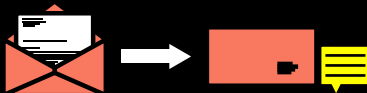
→ Messaging translates badly to our offline communication expectation



From Postcards to Letters



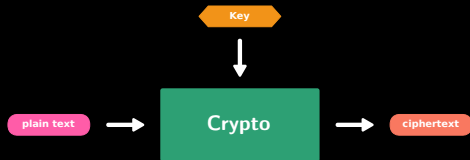
From Postcards to Letters



The Shortest Introduction to Encryption You Will Ever Get

Symmetric Encryption:

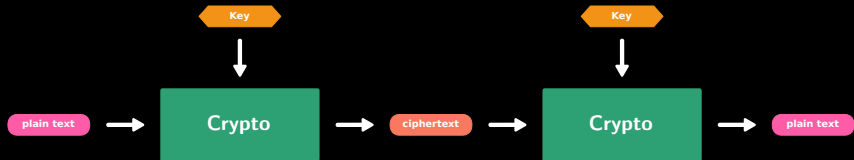
→ Encryption and decryption with the same key



The Shortest Introduction to Encryption You Will Ever Get

Symmetric Encryption:

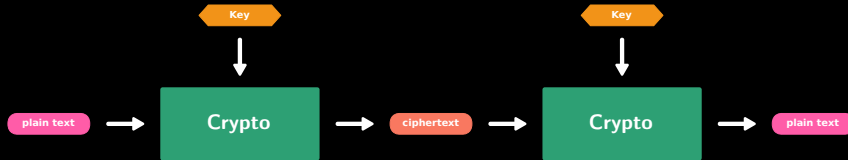
→ Encryption and decryption with the same key



The Shortest Introduction to Encryption You Will Ever Get

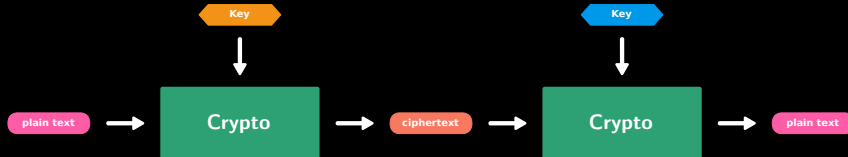
Symmetric Encryption:

→ Encryption and decryption with the same key



Asymmetric Encryption:

→ Encryption and decryption with different keys



The Shortest Introduction to Encryption You Will Ever Get

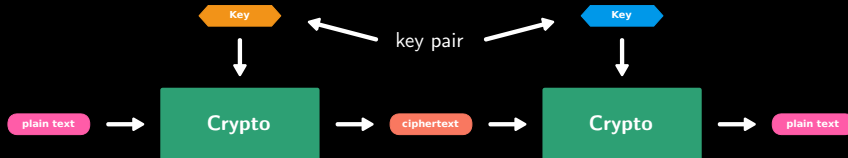
Symmetric Encryption:

→ Encryption and decryption with the same key



Asymmetric Encryption:

→ Encryption and decryption with different keys



Public-Key Cryptography – In a Nutshell



Secret Key

Public Key

Identity



Secret Key

Public Key

Identity

- Both parties publish their identities and public keys
- Any message can be encrypted with anyone's public key and only be decrypted with its corresponding secret key



Public-Key Cryptography – In a Nutshell



- Both parties publish their identities and public keys
- Any message can be encrypted with anyone's public key and only be decrypted with its corresponding secret key



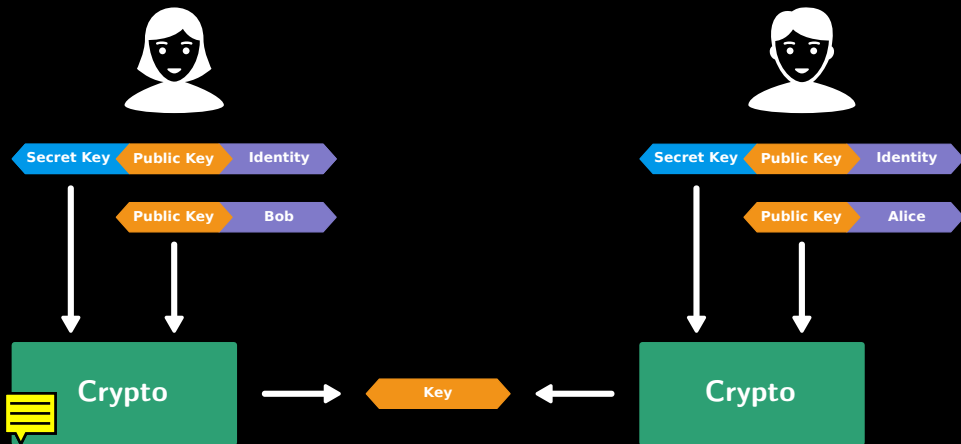
Public-Key Cryptography – In a Nutshell



- Both parties publish their identities and public keys
- Any message can be encrypted with anyone's public key and only be decrypted with its corresponding secret key

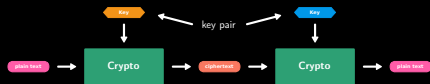


Authenticated Encryption



Recap

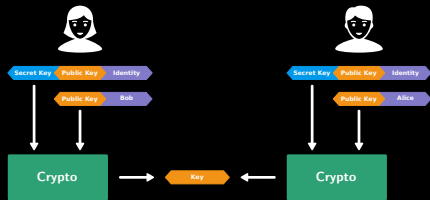
Asymmetric Encryption gives us IDs but is very expensive.



Symmetric Encryption is cheap, but a key has to be shared by all participants **before** communication starts.

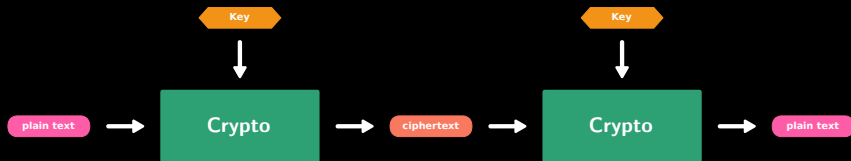


Authenticated Encryption allows us to create symmetric keys based on asymmetric key pairs.



But there's more...

Confidentiality



Deniability

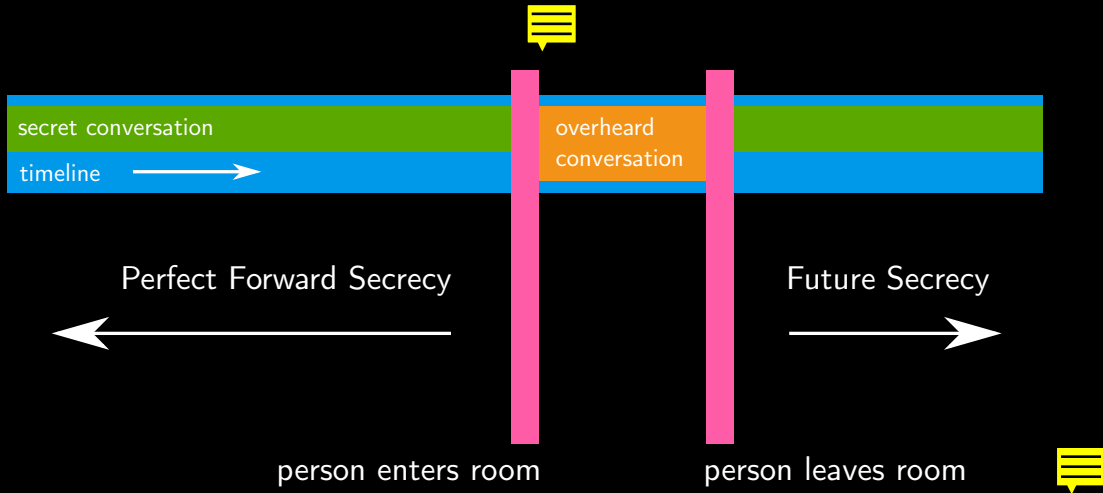
From:
either of us



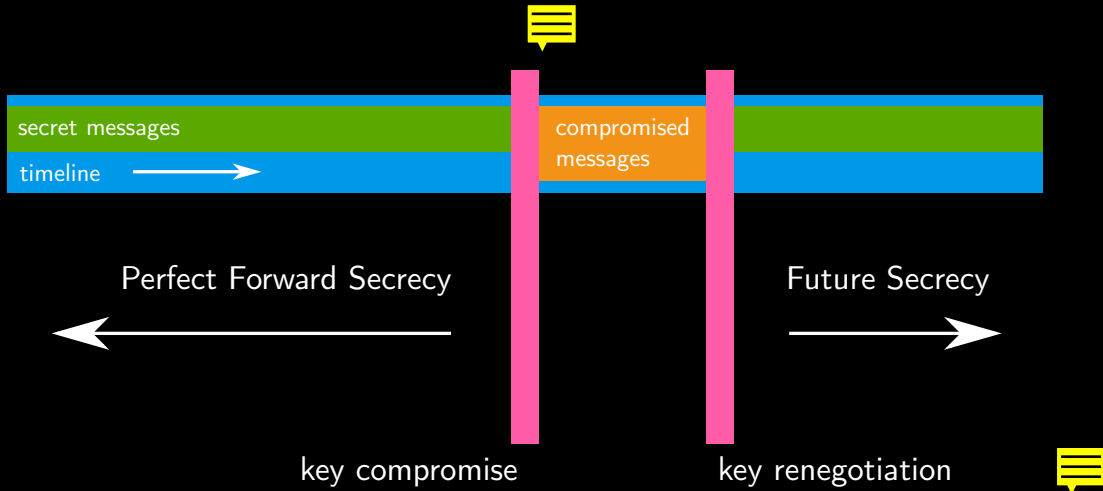
To:
both of us



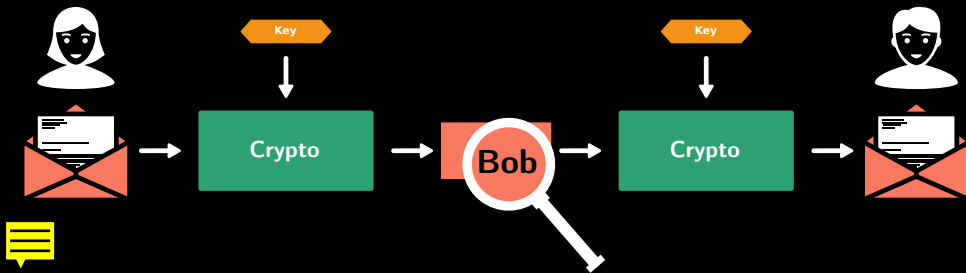
But What About Forward- and Future Secrecy?



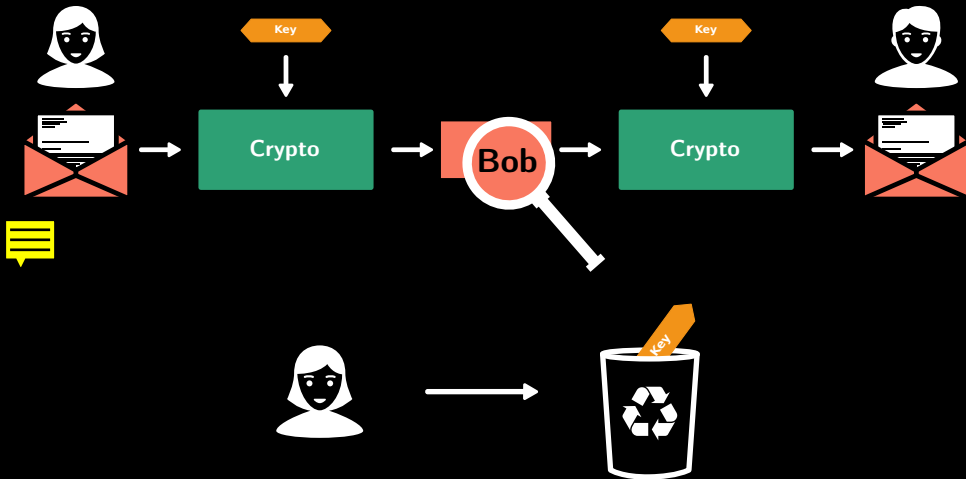
But What About Forward- and Future Secrecy?



But What About Forward- and Future Secrecy?



But What About Forward- and Future Secrecy?



Recap

Authenticated Encryption gives us:

- Confidentiality
- Deniability
- Authenticity

We don't have:

- Perfect Forward Secrecy
- Future Secrecy

→ We are ignoring Integrity here, but we have that, too.

Key Management

Cryptography is rarely, if ever, the solution to a security problem. Cryptography is a translation mechanism, usually converting a communications security problem into a key management problem.

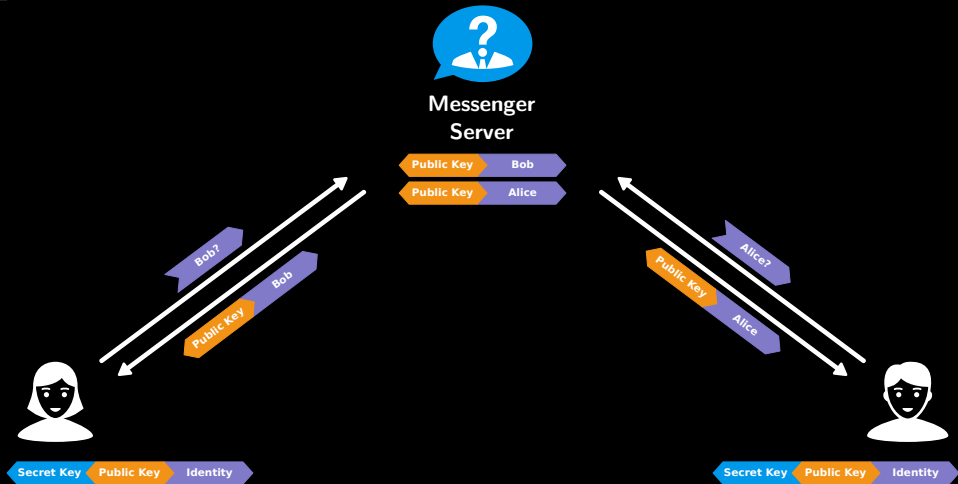
—Dieter Gollmann

Key Management

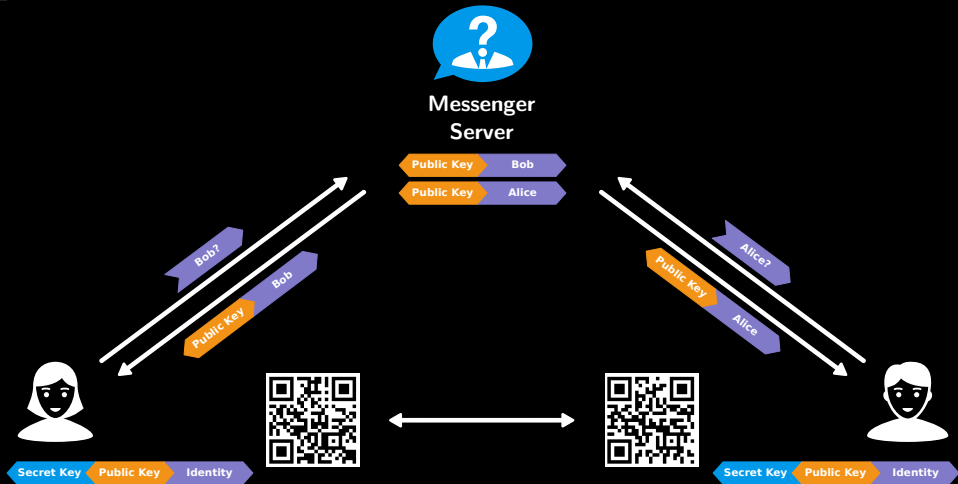


- How does Alice know which is Bob's public key?
- Identity keys stored on devices, what if stolen
- Keys for back-end communication layer hard-coded, almost impossible to replace
- How to deal with key compromise? (answer: key rotation)


Key Management



Key Management



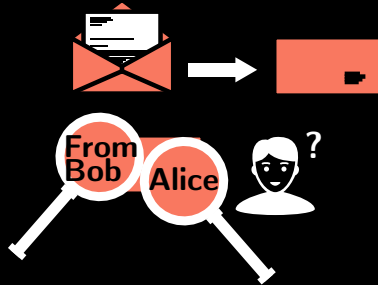
Authenticity

- How to connect a key to a person?
 - Key signing (PGP)
 - Certificates (trusted third party)
 - (Messenger  service-based directory (based on phone numbers or email addresses))
- How to deal with changing keys?
 - warnings are annoying
 - Threema's traffic light system encourages authentication but doesn't deal with changing keys (other than new identities for known phone numbers with yellow dots)

Metadata Handling

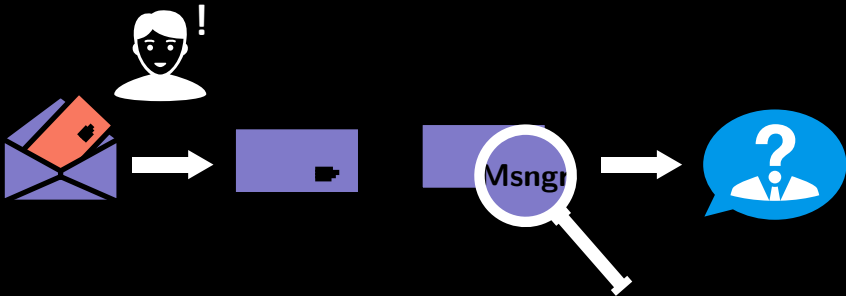
Everybody on the network can see:

- the sender of the message
- the intended receiver of the message

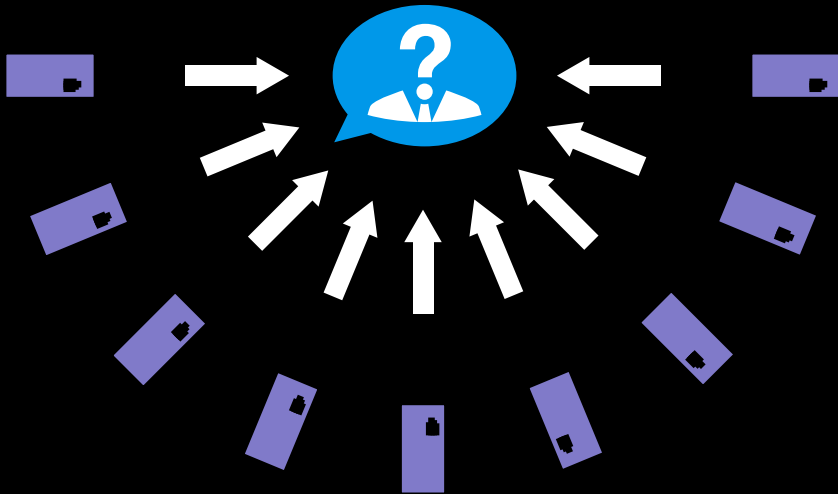


Metadata Handling

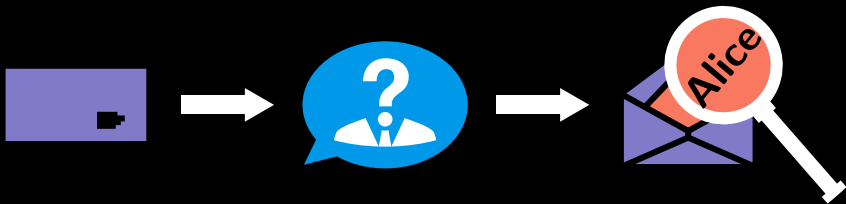
Solution: wrap encrypted message in a second layer of encryption and address it only to the message server.



Metadata Handling

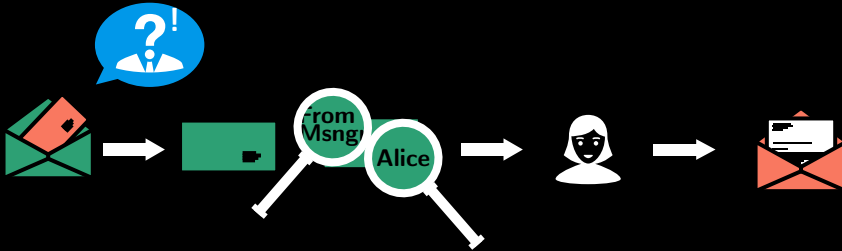


Metadata Handling



Metadata Handling

The message server will remove the outer layer and add a new one, targeted at the receiver.



Metadata Handling

This leaves us with an encrypted **end-to-end tunnel**, transmitted through two **transport layer** encryption tunnels.



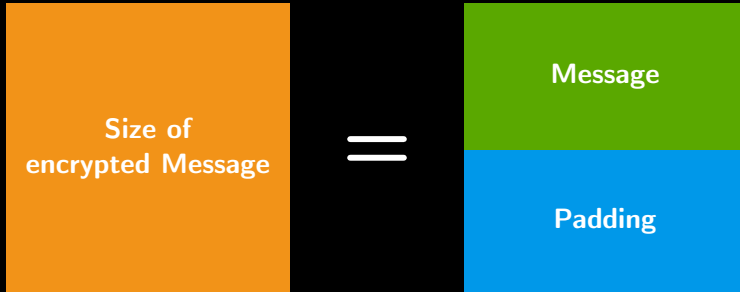
The message server still knows both communication partners!

Metadata Handling

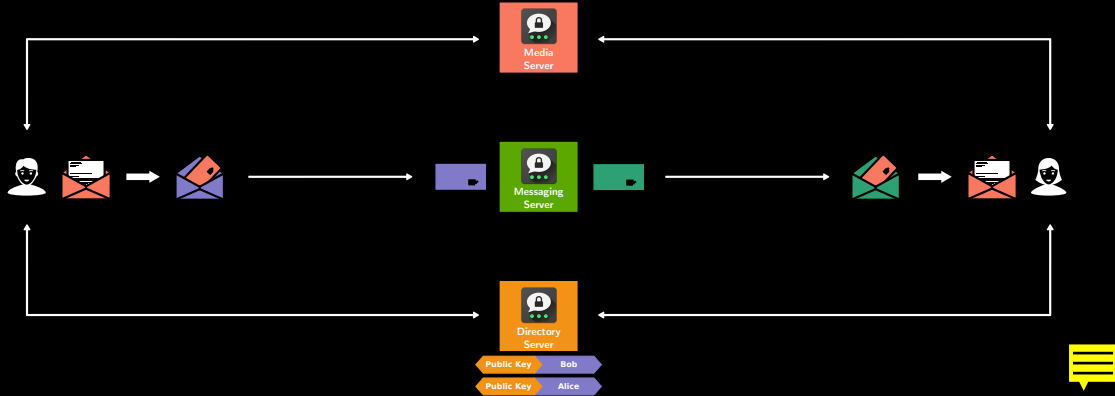
We can obfuscate the size of a message with `padding`

Metadata Handling

We can obfuscate the size of a message with **padding**



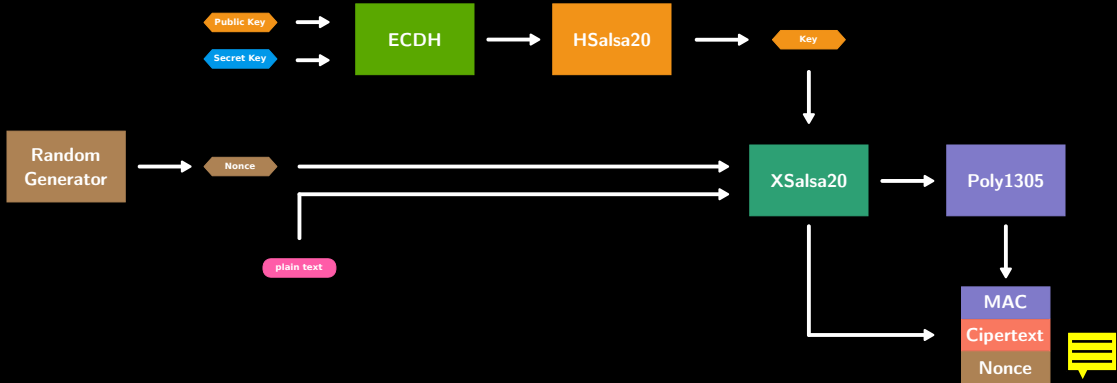
Threema's Architecture



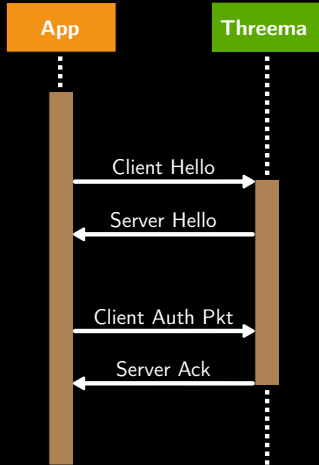
Threema Fingerprints



NaCl and Threema

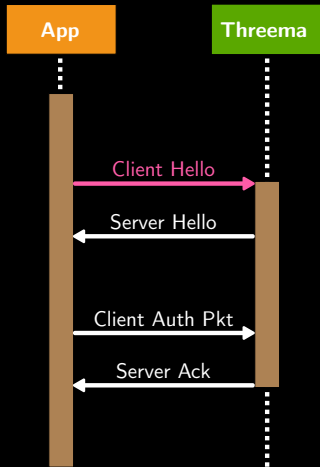


Threema's App-to-Server Handshake

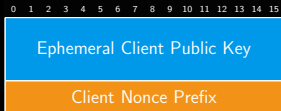


Initial Text goes here

Threema's App-to-Server Handshake

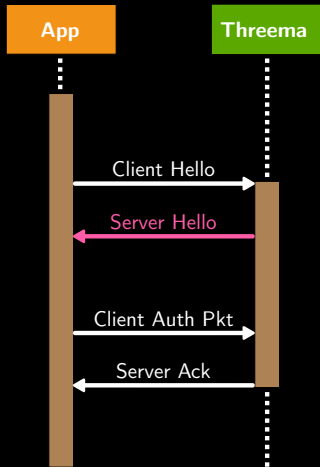


Client Hello Packet

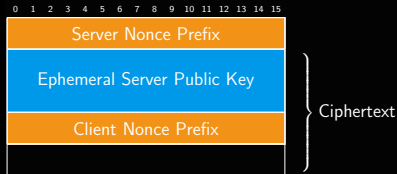


- Client generates a **ephemeral** key pair
- Client generates random nonce prefix

Threema's App-to-Server Handshake



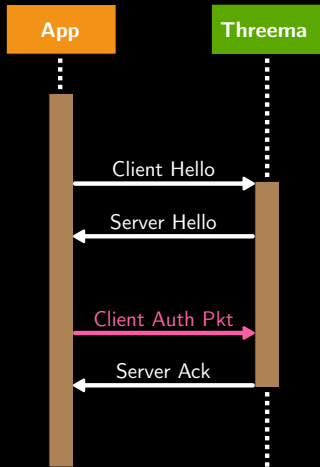
Server Hello Packet



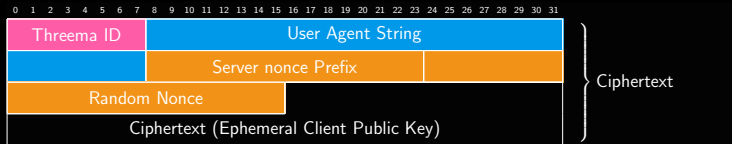
- Server generates ephemeral key pair
- Server generates random nonce
- Ciphertext encrypted with Server Nonce, Client Ephemeral Key and Server Long-Term Key



Threema's App-to-Server Handshake

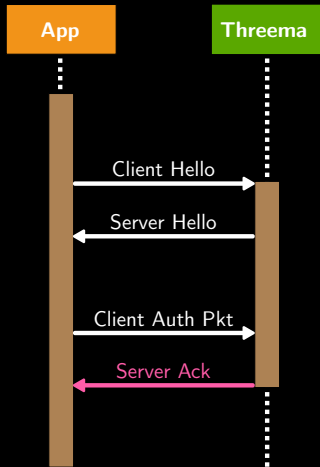


Client Authentication Packet

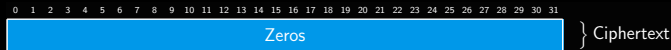


- Outer Encryption with ephemeral Keys
- Ciphertext links clients ephemeral key pair to it's long term key pair

Threema's App-to-Server Handshake



Server Acknowledgement Packet

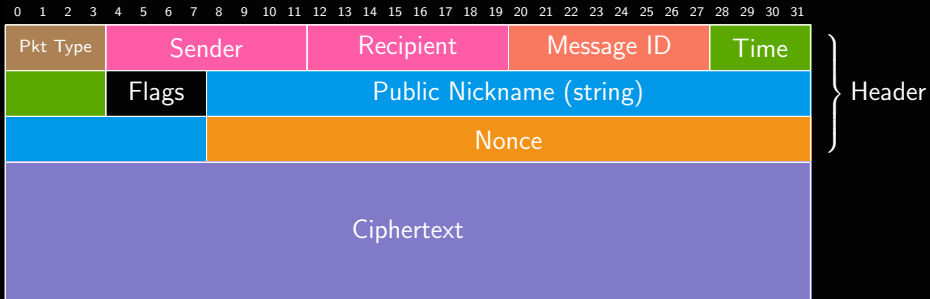


- Server confirms everything worked fine by encrypting something with both ephemeral keys

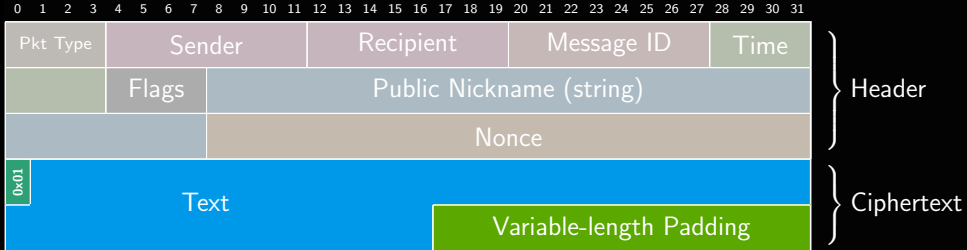


Lila und Gruen sind klar
Jetzt noch kurz rot

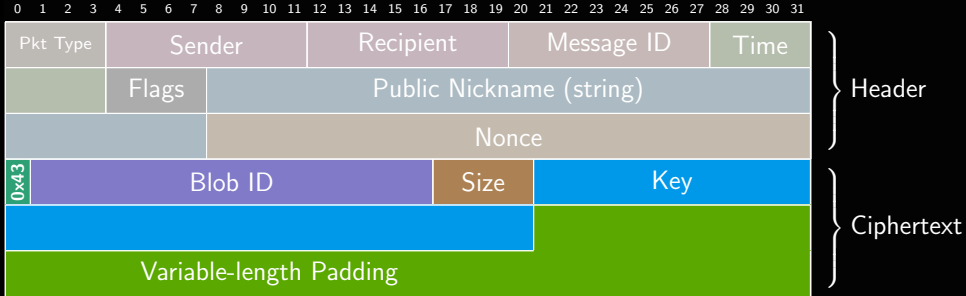
Threema Packet Format



Threema Text Messages



Threema Image Messages



Sending an Image Message

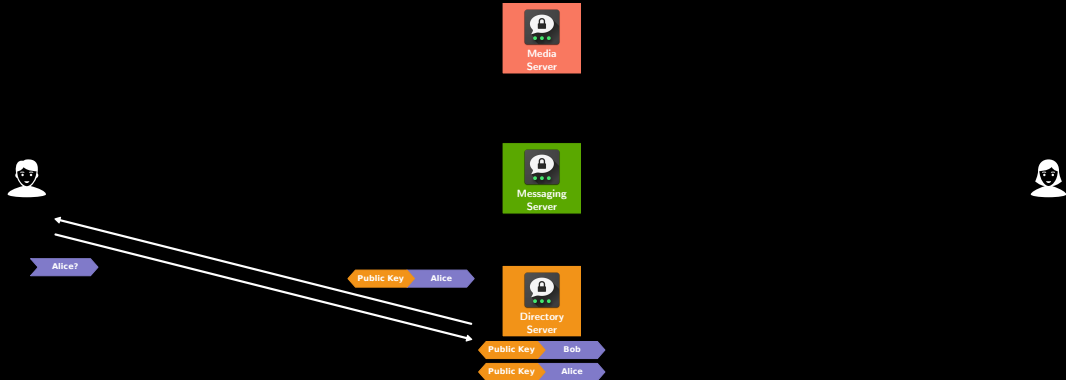


Public Key Bob

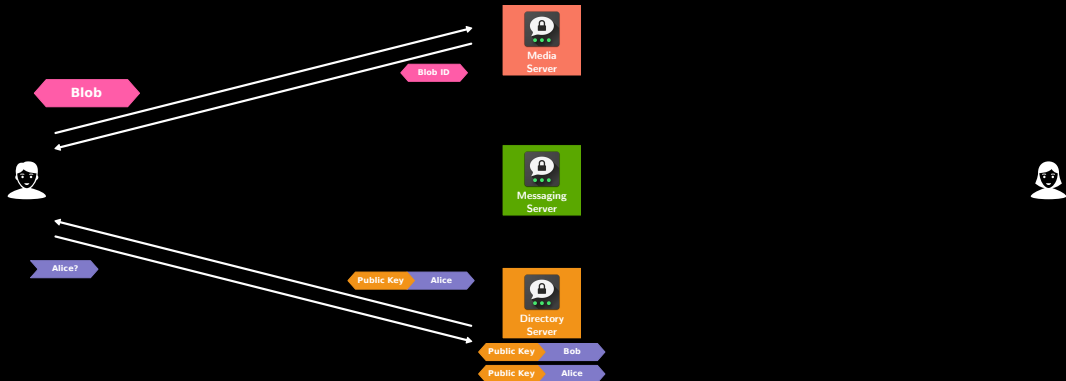
Public Key Alice



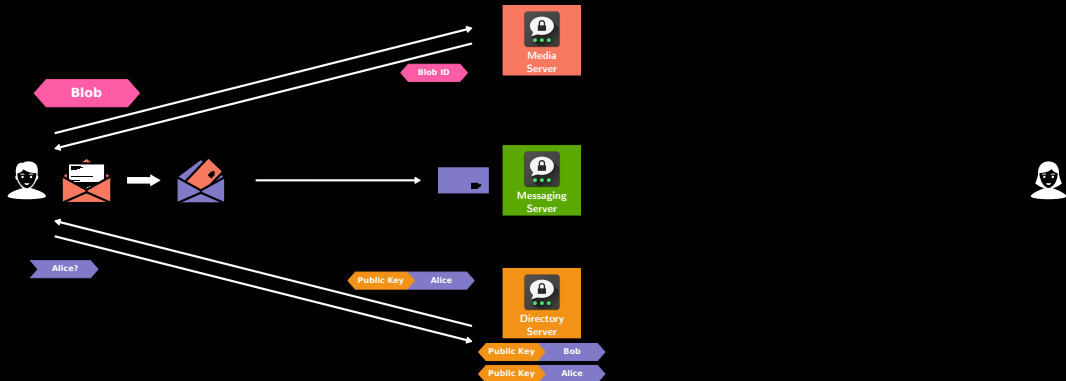
Sending an Image Message



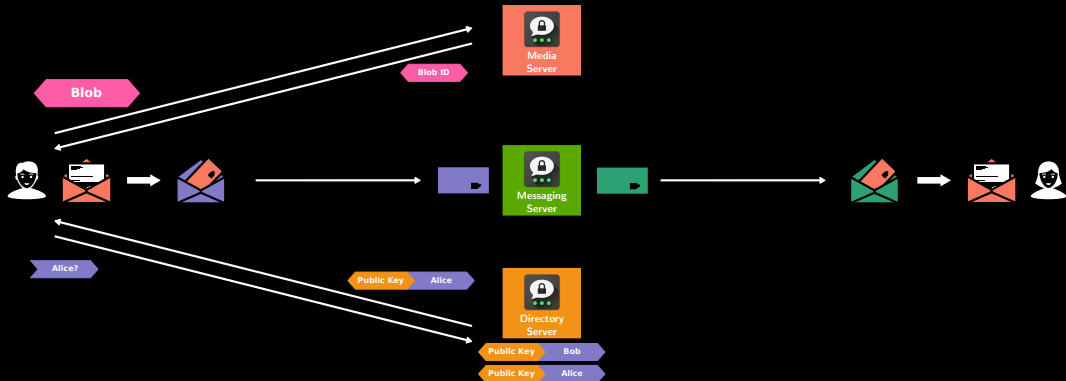
Sending an Image Message



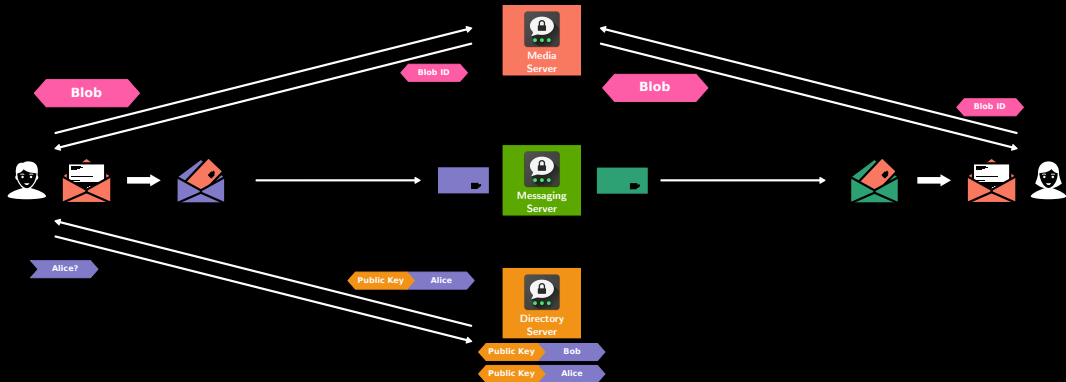
Sending an Image Message



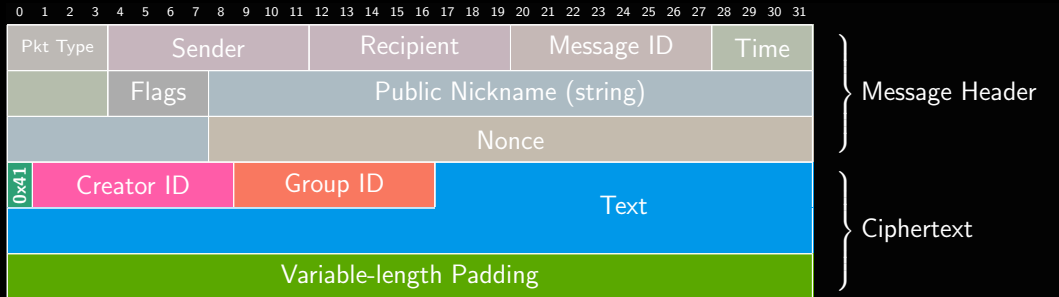
Sending an Image Message



Sending an Image Message



Group Messages



Group Messages

- Group IDs aren't unique. They are created locally and only work together with the group creator's ID
- The structure of group messages makes it impossible for Threema to introduce multiple group administrators on a protocol level without breaking compatibility to older clients.

The Devil's in the Detail

Sammlung kleinerer Dinge, die uns aufgefallen sind

- Media messages could be StageFright attach vectors
- The protocol implementation looks sound to us but the message design prevents feature upgrades on the protocol (not text-protocol) level

Threema - Notes and Open Questions

Notes:

- PFS only on transport layer (attacker sniffing packets from the outside will not learn contents after private key acquisition)

Q:

- How often is the handshake performed?

Signal Forward and Future Secrecy

- If you want forward secrecy, you need to use asymmetric crypto and dispose of your keys as often as possible
- To do that you need to perform a new handshake frequently
- Since it is not always given that both parties are online for that handshake, the good people of Whispersystems have come up with this:
 1. Both parties upload a bunch of keys to the server. Those are signed identity keys as we've seen previously and a large number of **prekeys**
 2. Using a prekey, any party can perform their part of the handshake offline and end up with a new session key to use
 3. These session keys are renewed by **ratcheting** with each message transfer

Signal - Notes

- Server-side cached short-term keys (**prekeys**) fetched by sender
- Pairwise long-lived symmetric secret key between participants
- Multiple messages without answer → perform KDF on **chaining key**
- ECDH: Curve25519, AES-CTR (no padding), AES-CBC (PKCS7 padding)
- HMAC-SHA256 for integrity
- Future secrecy only if private keys are not leaked (duh!). Since private keys go into new shared keys during ratcheting, the attacker lacks material to compromise next key after obtaining current one.
- Since shared keys are only deleted after messages are received, there is a window in which keys could be compromised before reception.
- Deniability is always a theoretical claim as long as a transmission server has the ability to log messages, their senders and recipients.

Enter Threema

Threema

- Gained popularity in Germany after Facebook purchased WhatsApp
- All promise – no proof; first openly contemplating to OSS the code, later backing away from that statement
- Interest in its inner workings

Quick Shoutouts

- Jan Ahrens for releasing his findings about the handshake before we did
- OpenMittsu for releasing the first working OSS client

Reverse-Engineering – What to look for?


- Test for common pitfalls in implementation
 - Handling of TLS
 - Handling of keys and nonces
 - NaCl implementation errors
 - Uncommon data leaks
 - Bugs
 - ...?
- Find out how protocol is designed
 1. Understand handshakes
 2. Understand protocol
 3. decipher messages


Positive side-note: Threema had released a security white paper early on

Done!


Thank You!


Roland Schilling

 schilling@tuhh.de

 @NerdingByDoing

Frieder Steinmetz

 frieder.steinmetz@tuhh.de

 @twillnix

Beamer Theme: [Metropolis](#) by Matthias vorgelsang

Color Theme: [Owl](#) by Ross Chirchley

Icons: [The BIG collection](#) by Sergey Demushkin

[Foundation Icon Fonts 3](#) by ZURB

Thanks to [Jan Ahrens](#) and [Philipp Berger](#) – their work has made ours somewhat easier

Thanks to [Maximilian Köstler](#) for his initial work on Threema