ETH zürich

DINFK

Secure Messaging: The Good, The Bad and The Ugly

Kenny Paterson

Applied Cryptography Group
Department of Computer Science
ETH Zurich

14 June 2022

APPLIED CRYPTO GROUP

Agenda

- Secure Messaging
- The Good: Signal

<break>

- The Bad: Bridgefy
- The Ugly: Telegram
- Closing Remarks

Many thanks to Ben Dowling, Raphael Eikenberg, Felix Günther, Lenka Mareková, Igors Stepanovs for slides.

APPLIED CRYPTO GROUP

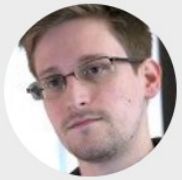
Secure Messaging

Monthly active users in Jan 2022:

According to Statistica 2022.

WhatsApp	$2000 \cdot 10^6$
WeChat	$1263 \cdot 10^6$
FB Messenger	$988 \cdot 10^6$
QQ	$574 \cdot 10^6$
Snapchat	$557 \cdot 10^6$
Telegram	$550 \cdot 10^6$

Signal: $40 \cdot 10^6$



"I use Signal every day."

Edward Snowden

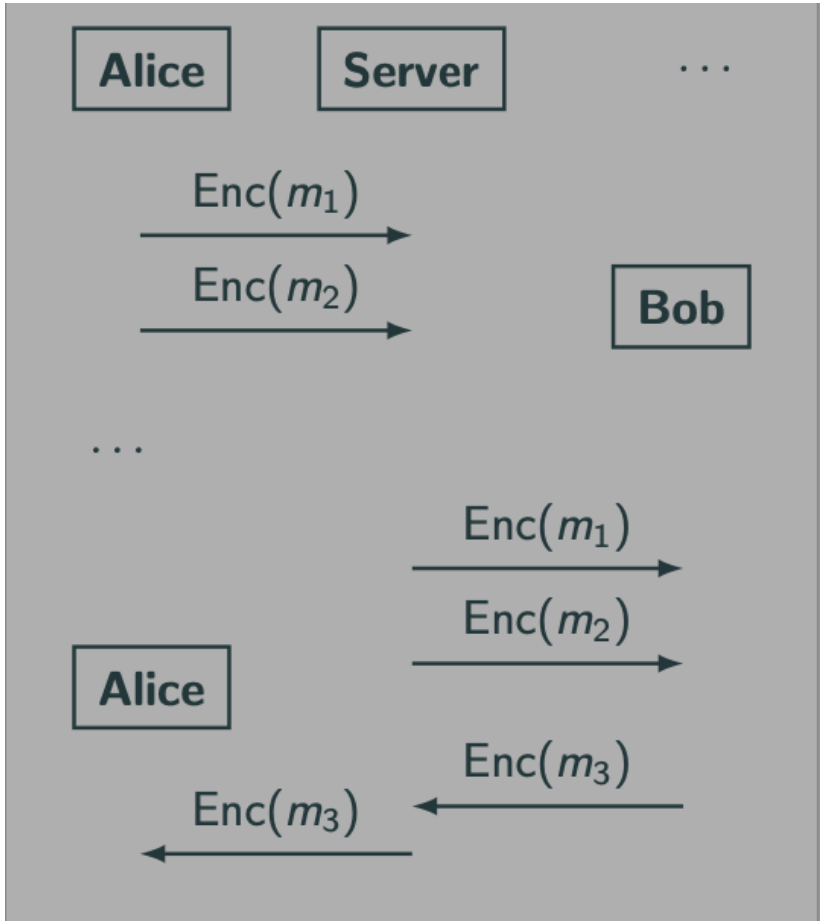
Whistleblower and privacy advocate

https://signal.org/

APPLIED CRYPTO GROUP

End-to-End Encryption (E2EE) in Messaging

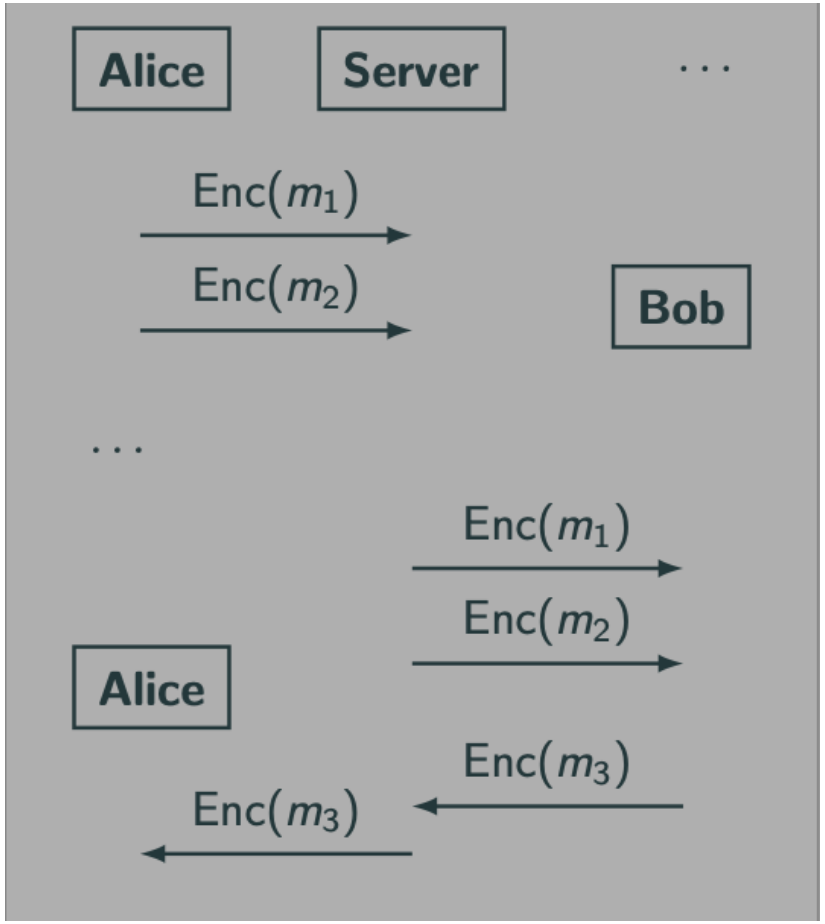
- Alice and Bob wish to communicate securely.
- They rely on a server to store and forward ciphertexts.
- The server only needs sufficient metadata to ensure message delivery (and consistent ordering).
- In particular, the server does not need to be able to see message contents.
- **E2EE**: when only legitimate users Alice and Bob can read messages.
- But EE2E systems still rely on server to also distribute trusted keying material, to bootstrap secure communications.
- Typically use of “out-of-band” mechanisms to ensure authenticity, e.g. human comparison of key fingerprints.



APPLIED CRYPTO GROUP

Asynchronicity in Messaging

- Bob may be offline when Alice sends a message; Alice may be offline later when Bob replies.
- Rely on the server to **store and forward** ciphertexts.
- Any keys need to be established and updated (rotated) in an **asynchronous** manner.
- Communication at “human speed” allows different cryptographic techniques to be used compared to, say, TLS Handshake Protocol.
- For example, Signal does (almost) per-message Diffie-Hellman key exchange.



APPLIED CRYPTO GROUP

Agenda

- Secure Messaging
- The Good: Signal ← We are here

<break>

- The Bad: Bridgefy
- The Ugly: Telegram
- Closing remarks

APPLIED CRYPTO GROUP

Why Study Signal?

- Two party **asynchronous**, **E2EE** messaging protocol.
- Wide uptake:
  Signal, **WhatsApp**, Wire, Facebook Silent Messenger, and many more!
- Interesting security properties:
  Fine-grained forward security and post-compromise security.
- Complex but clean design:
  X3DH: initial key exchange Double Ratchet: asymmetric and symmetric ratcheting...
- Provides a high bar for other messengers to aspire to.

APPLIED CRYPTO GROUP

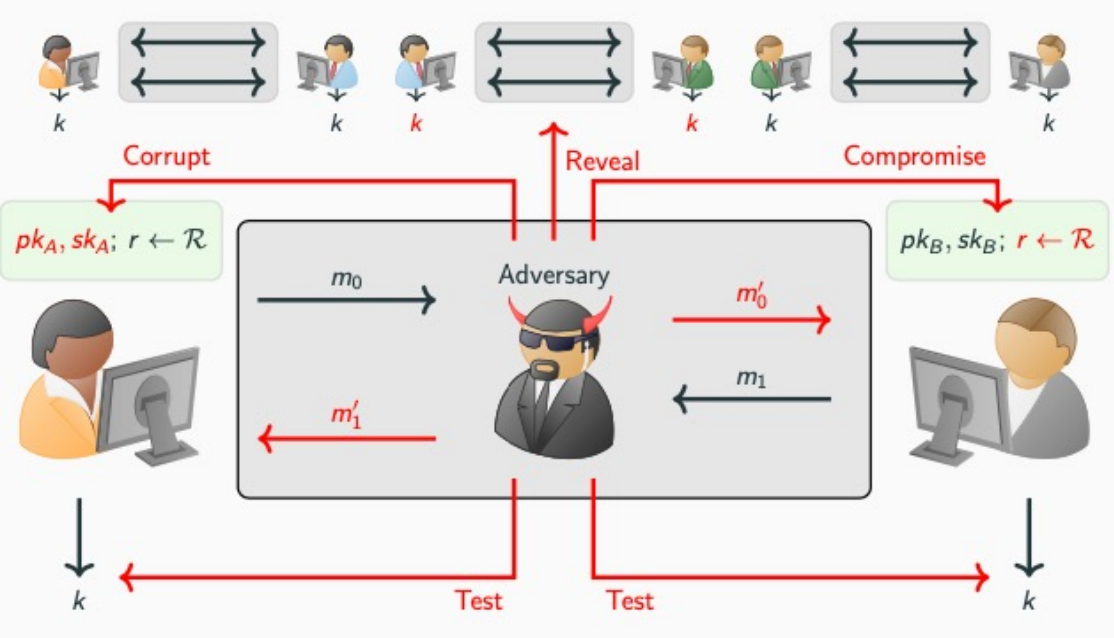
Security Model for Asynchronous Authenticated Key Exchange (AAKE)

The security model implicitly defines attacks that Signal is trying to protect against.

Signal can be proven secure in a very strong attack model in which the attacker:

- is in complete control of the network,
- can Reveal derived message keys,
- can Corrupt long-term secrets,
- can Compromise ephemeral secrets.

Even with these capabilities, **key indistinguishability** holds: an adversary cannot distinguish the message encryption keys used by Signal from random bit-strings*,**,***



*Under suitable computational assumptions.

**Excluding *trivial wins.*

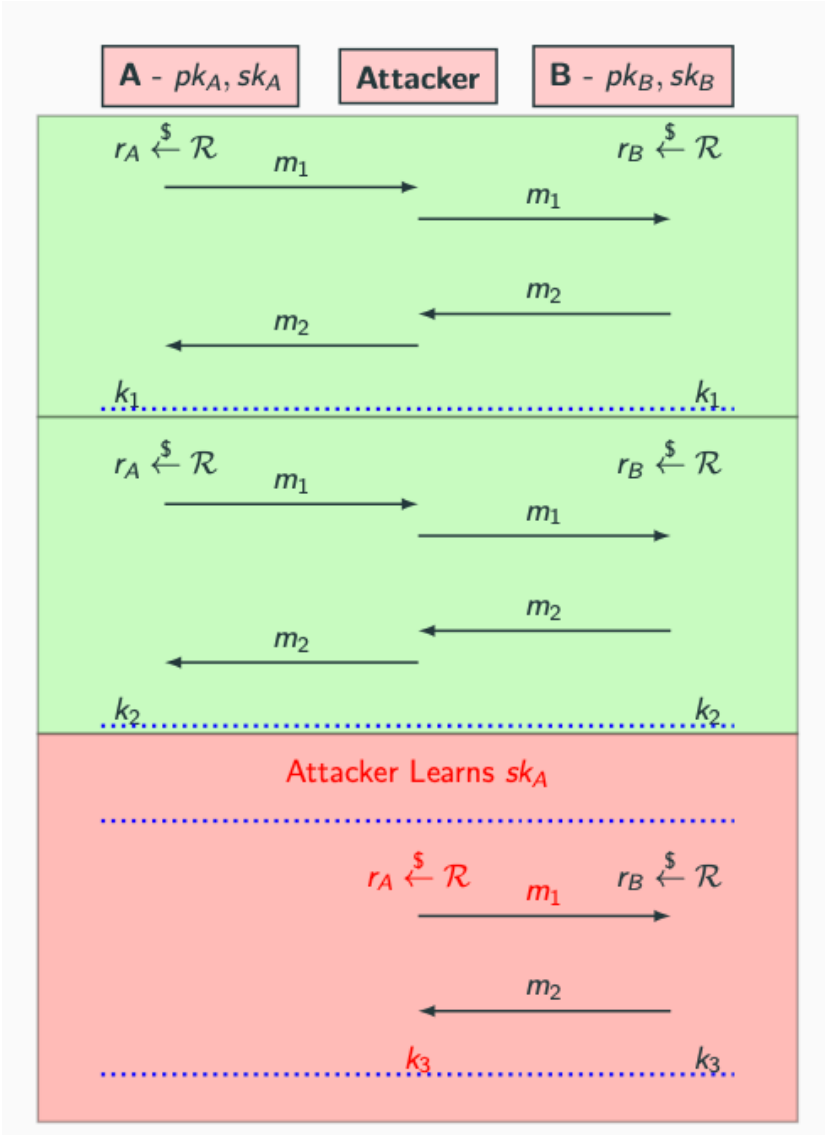
***Don't worry, no proofs today!

APPLIED CRYPTO GROUP

Forward Security in AAKE

Forward Security considers the following scenario:

- Attacker has full control of the network.
- Alice and Bob execute AAKE together.
- Attacker Corrupts Alice's long-term private key sk_A (or even all of Alice's stored state).
- Attacker can use this to impersonate Alice
- So, obviously, protocol executions with "Alice" after this point are not secure.
- But what about before attacker learns sk_A?
- If key indistinguishability still holds for session keys established before compromise, the protocol is said to have **forward security**.

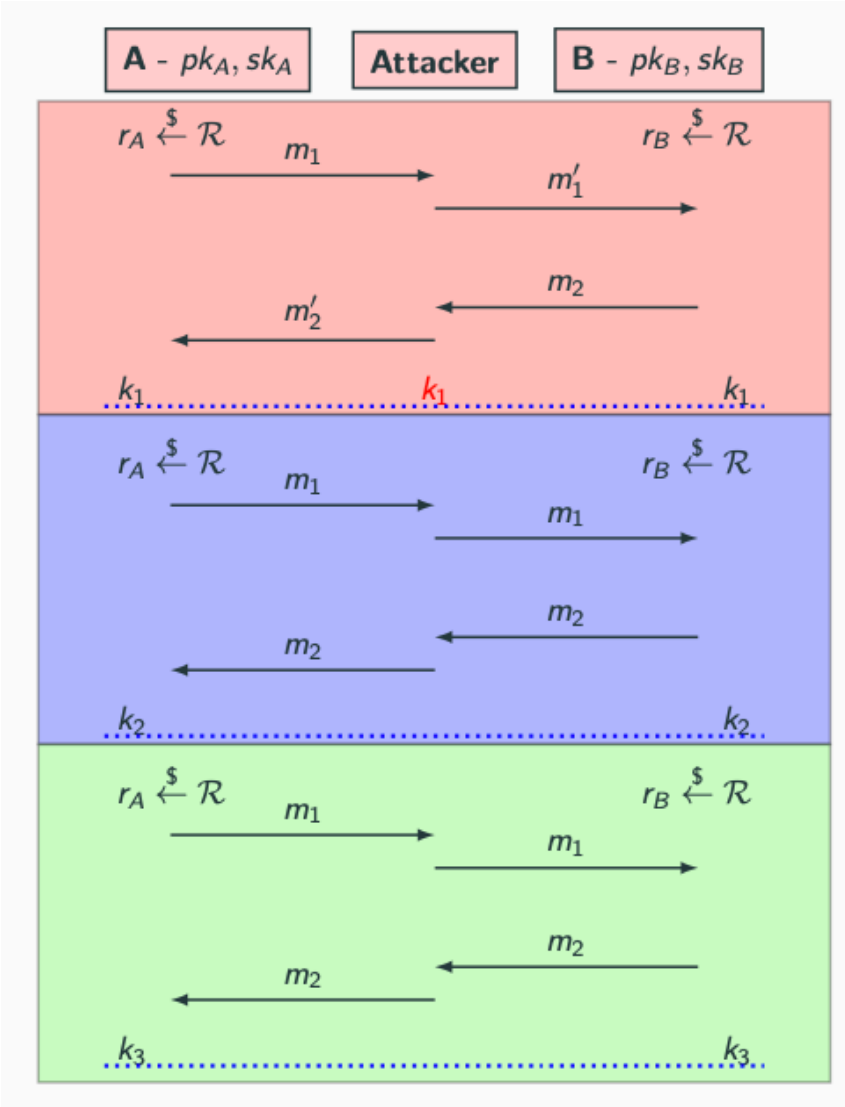


APPLIED CRYPTO GROUP

Post-Compromise Security in AAKE

Post-Compromise Security considers the following scenario:

- Attacker has full control of the network
- Attacker Corrupts and Compromises Alice and Bob, learning their full states.
- Clearly an **active attacker** can now continue the attack and learn future message keys.
- But what if attacker becomes passive for a time?
- So Alice and Bob execute AAKE protocol without adversarial interference for a while.
- If Alice and Bob can recover security as a result (even with adversarial knowledge of sk_A, sk_B) then the protocol is said to have **Post-Compromise Security (PCS)**.
- Roughly, PCS means: “Best possible security after a compromise.”



APPLIED CRYPTO GROUP

Signal: High-level Overview

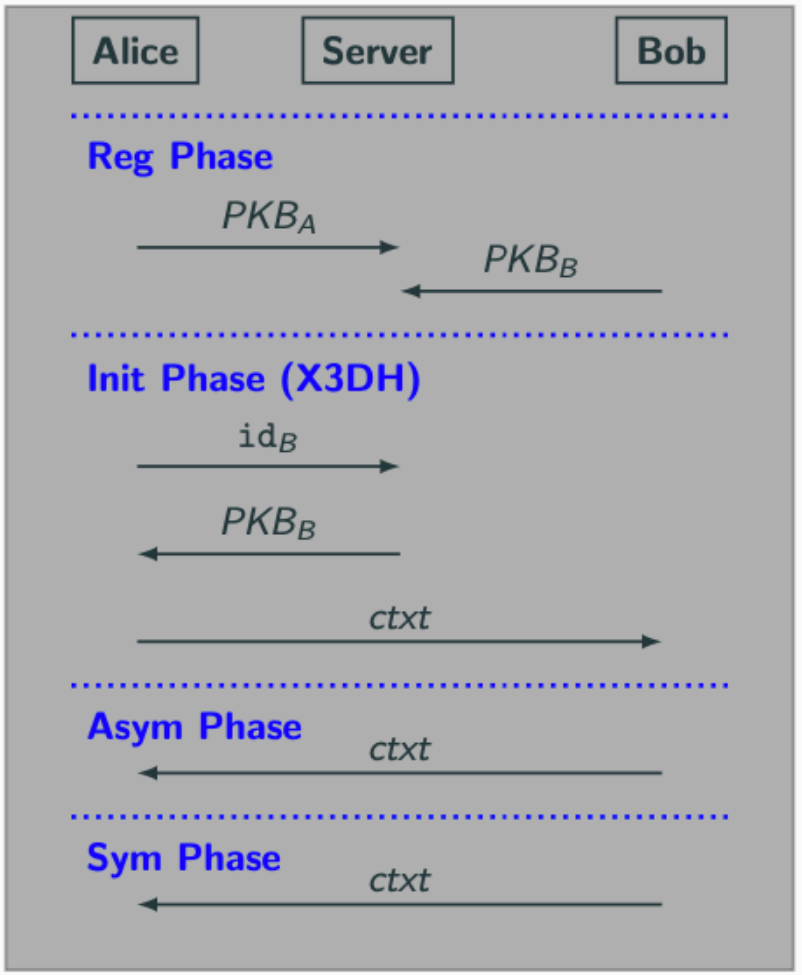
At a high level, the Signal Protocol consists of 4 “distinct” stages.

- A Registration Phase
- An Initialisation Phase
- An Asymmetric Ratchet Phase
- A Symmetric Ratchet Phase

We will focus on the last two.

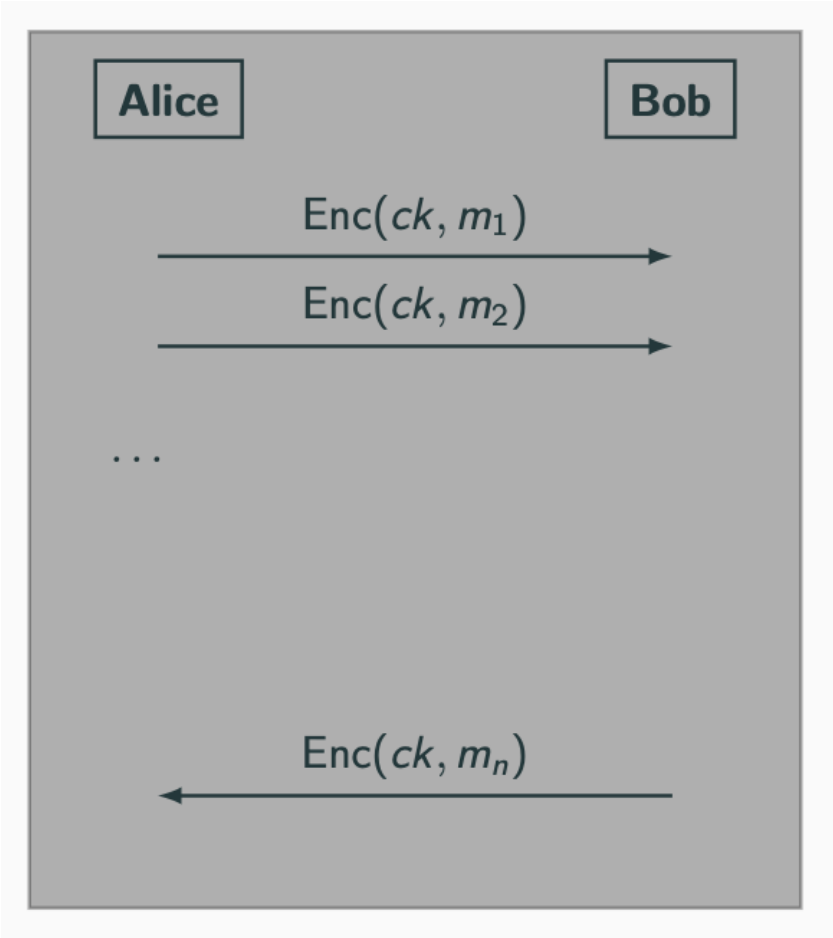
The first two are also interesting…

Many other aspects to Signal too, e.g. multi-device support, safety numbers, contact discovery, group messaging, sealed sender,…



APPLIED CRYPTO GROUP

Signal Symmetric Ratchet: Intuition



Let's assume Alice and Bob have already established a shared symmetric key, somehow.

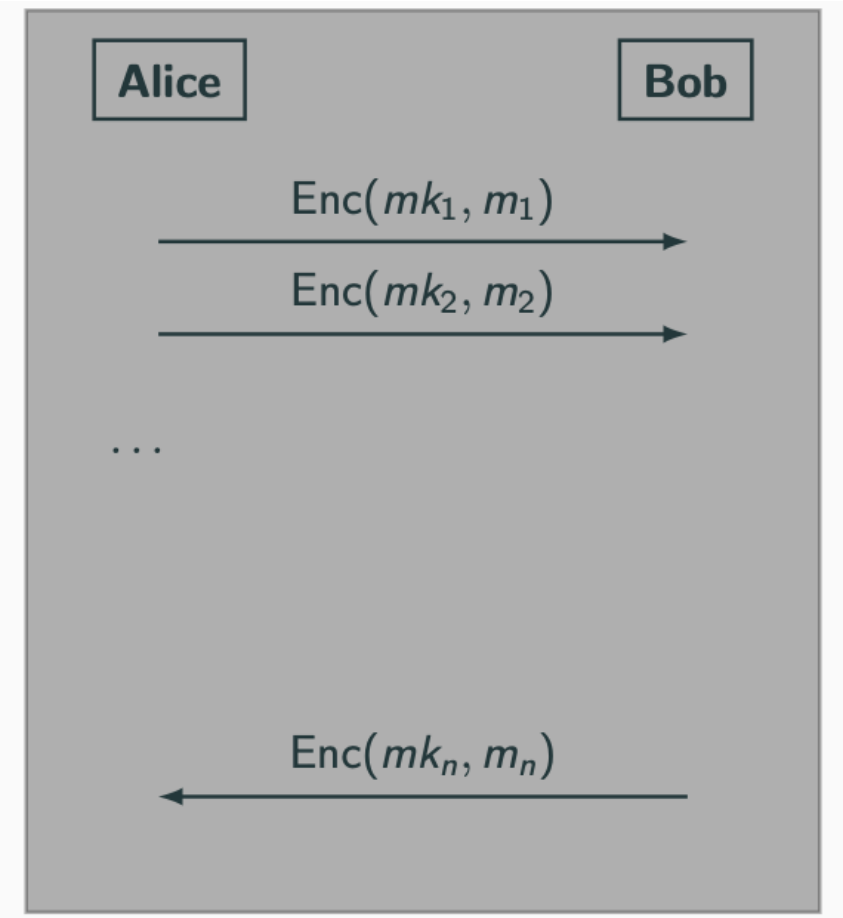
We'll call this the **chain key**, *ck*.

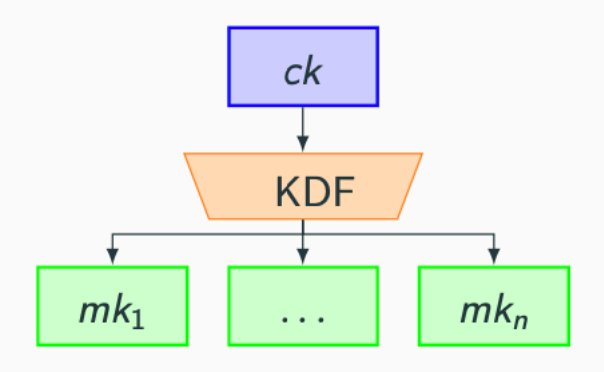
Naive approach: use *ck* to encrypt messages!

- Using a well-designed AEAD scheme, of course!
- What happens if *ck* is later exposed?
- All security is lost!

APPLIED CRYPTO GROUP

Signal Symmetric Ratchet: Intuition

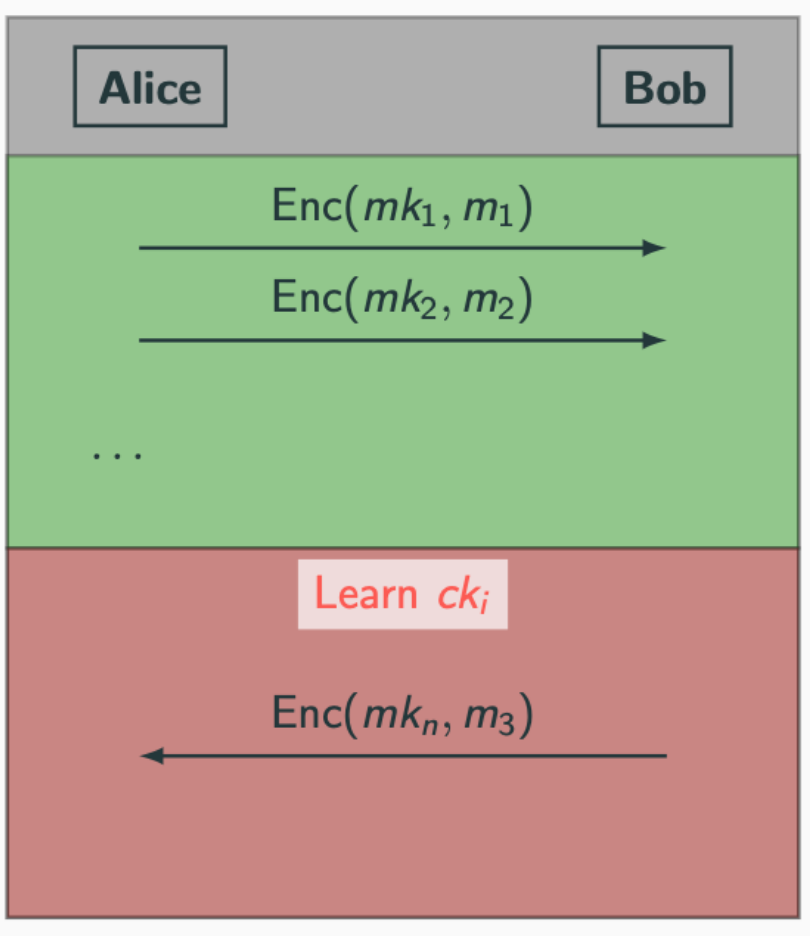


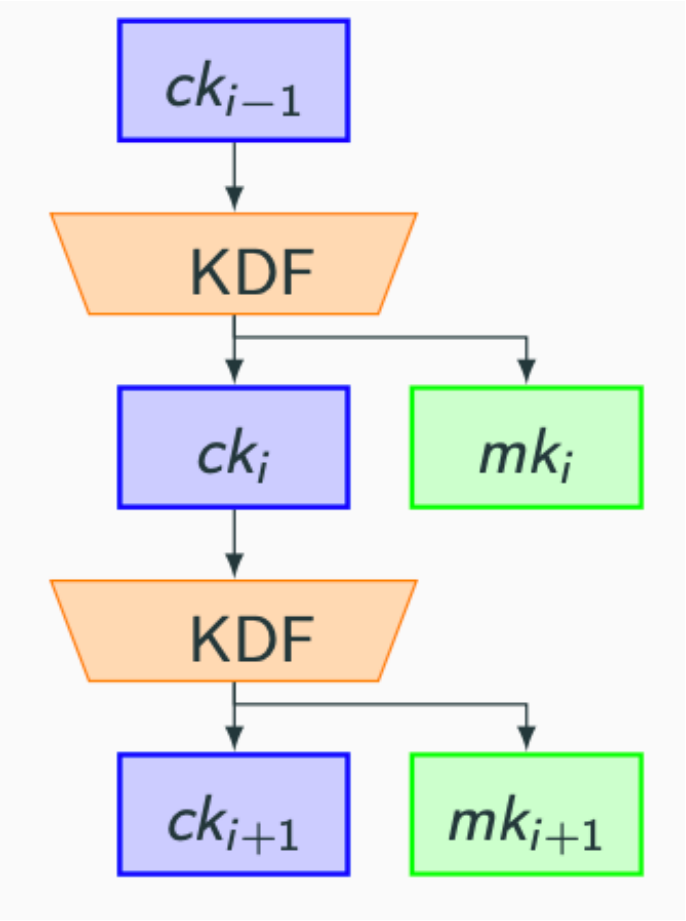


- What if we use *ck* as the input to a KDF?
- Then every message is now encrypted with a new message key mk_i.
- However, we need to keep *ck* around to derive future keys.
- So what happens when *ck* is later exposed?
- Still all security is lost!

APPLIED CRYPTO GROUP

Signal Symmetric Ratchet: Intuition

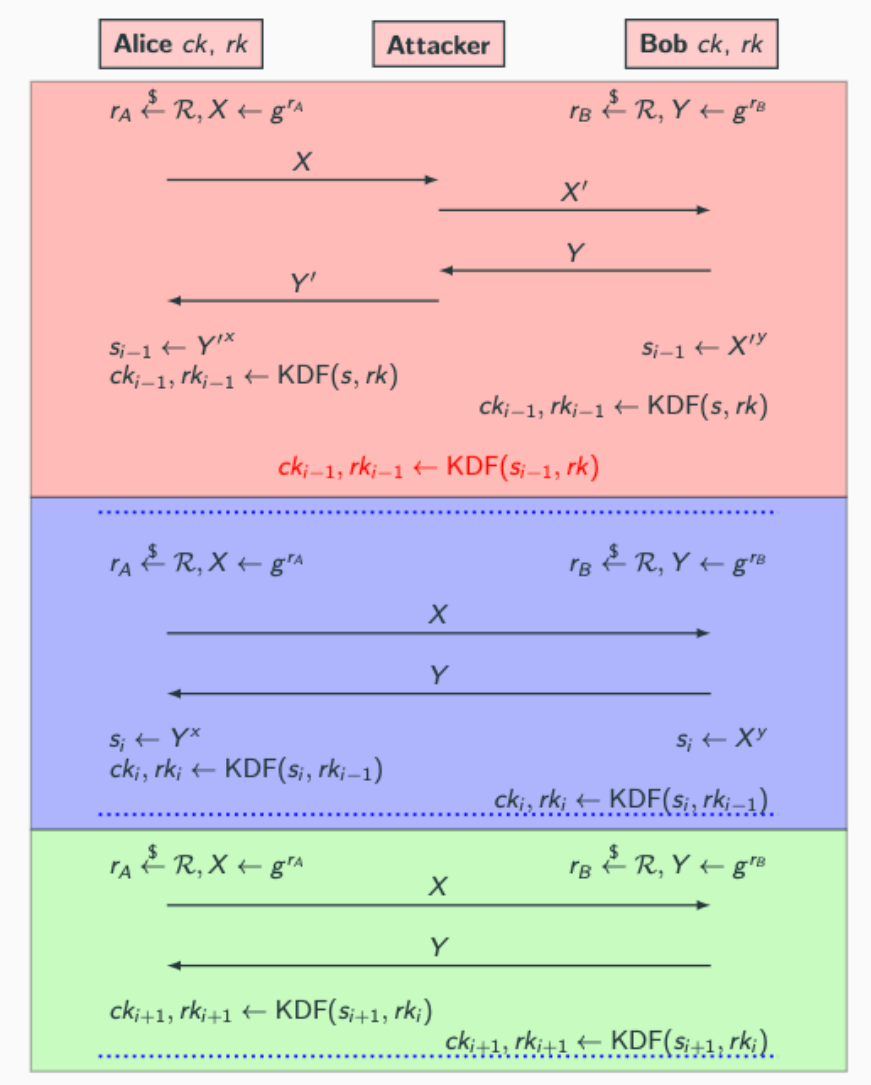




- Now we use the KDF to **ratchet** forward ck_i, using each ck_i only once to derive ck_{i+1},mk_{i+1}.
- We gain FS: only messages encrypted after an exposure are compromised!
- Technique well-known and widely used elsewhere.
- KDF is cheap to compute on a per message basis, e.g. HKDF requires a few hashing operations.

APPLIED CRYPTO GROUP

PCS and the Signal Asymmetric Ratchet: Intuition



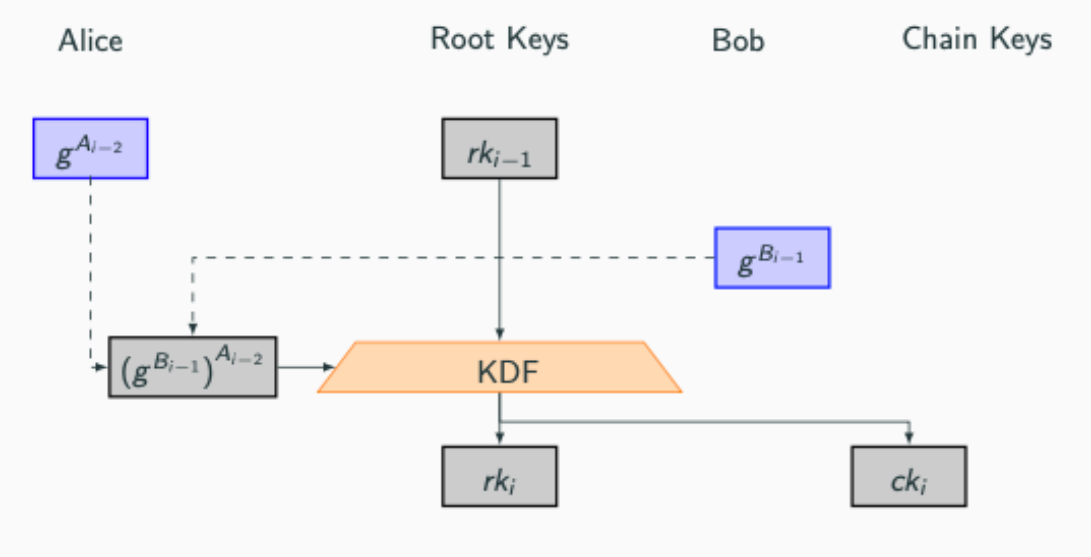
- Suppose Alice and Bob already have root key, *rk*:

$$ck_{i+1} \leftarrow \mathrm{KDF}(rk_i, \ldots)$$

- Same problem as before: if the attacker learns rk_i, security is lost for all ck_j (where $j > i$).
- Can we achieve PCS?
- Hint: symmetric techniques cannot help us here!
- Idea: constantly execute DH, combining rk_{i-1} with DH outputs to derive new ck_i, rk_i.
- Attacker passive in blue phase; untampered DH exchange produces shared DH value s_i; we recover secure ck_i, rk_i.
- DH is relatively cheap to compute on a per message basis (at human communication speeds).
- We need a new *dual PRF* assumption: we want KDF outputs to appear random when either s_i or rk_{i-1} is.
- Problem?
- Asynchronicity!

APPLIED CRYPTO GROUP

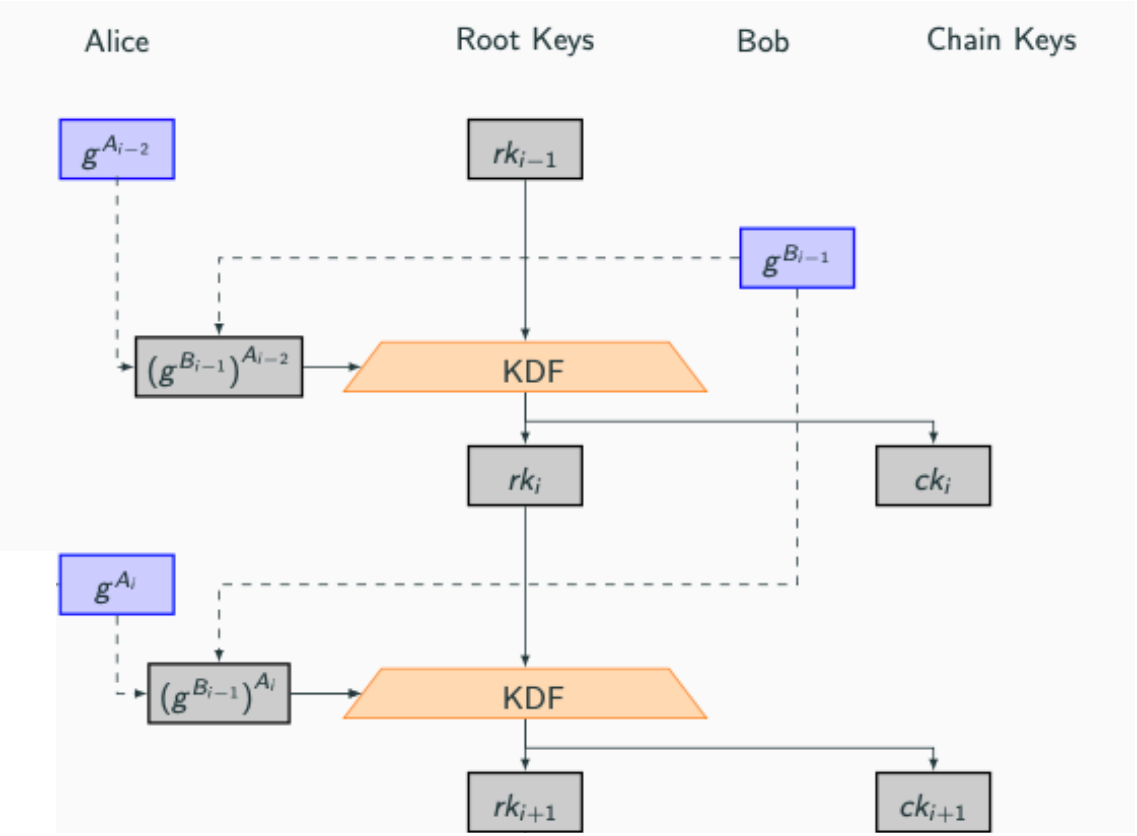
Solution: Asynchronous Ratcheting



- Assume that Alice and Bob *ping-pong* messages: A→ B, then B → A, repeat...
- Let's assume Alice has already sent Bob a DH public value $g^{A\_\{i-2\}}$ and has received $g^{B\_\{i-1\}}$ from Bob.
- Alice can now compute $(g^{B\_\{i-1\}})^{A\_\{i-2\}}$ and feed this into her KDF to compute rk_i and ck_i.

APPLIED CRYPTO GROUP

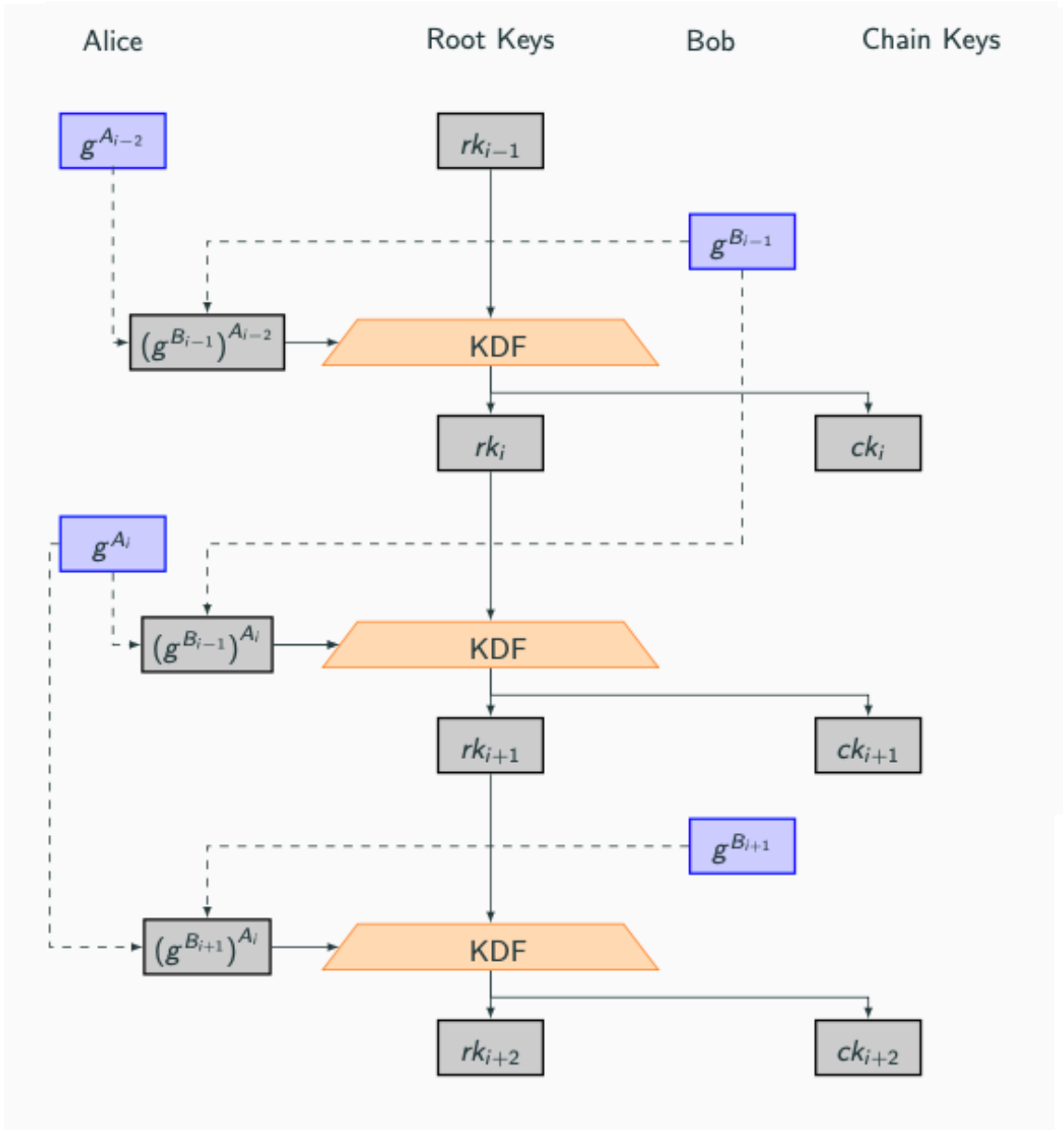
Solution: Asynchronous Ratcheting



- Assume that Alice and Bob *ping-pong* messages: A→ B, then B → A, repeat...
- Let's assume Alice has already sent Bob a DH public value $g^{A_{i-2}}$ and has received $g^{B_{i-1}}$ from Bob.
- Alice can now compute $(g^{B_{i-1}})^{A_{i-2}}$ and feed this into her KDF to compute rk_i and ck_i.
- With her next message to Bob, she includes a new DH value g^{A_i}.
- She combines this with Bob's previous $g^{B_{i-1}}$ to make a new value $(g^{B_{i-1}})^{A_i}$.
- And runs the KDF again to make rk_{i+1} and ck_{i+1}.

APPLIED CRYPTO GROUP

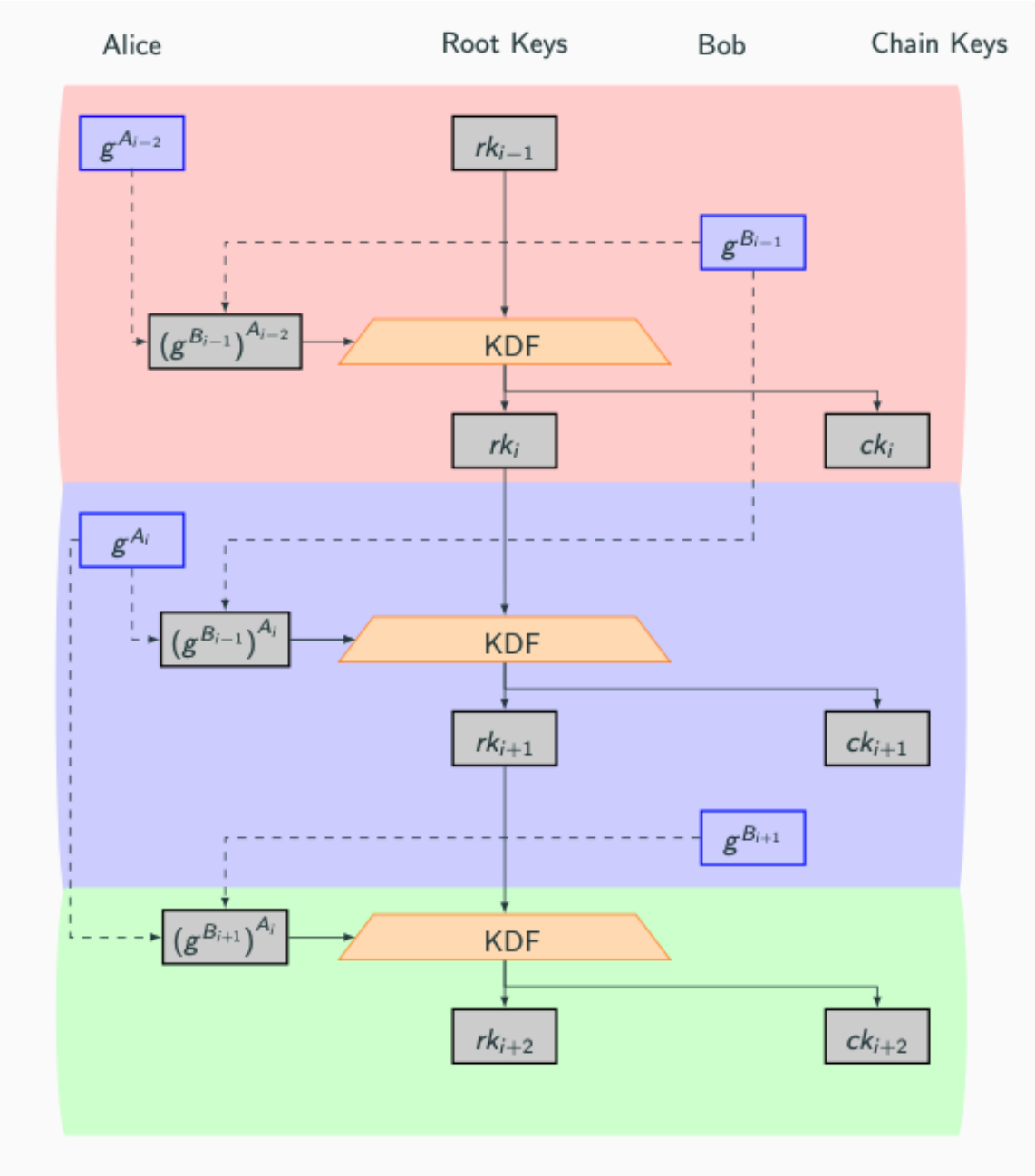
Solution: Asynchronous Ratcheting



- Assume that Alice and Bob *ping-pong* messages: A→ B, then B → A, repeat...
- Let's assume Alice has already sent Bob a DH public value $g^{A_{i-2}}$ and has received $g^{B_{i-1}}$ from Bob.
- Alice can now compute $(g^{B_{i-1}})^{A_{i-2}}$ and feed this into her KDF to compute rk_i and ck_i.
- With her next message to Bob, she includes a new DH value g^{A_i}.
- She combines this with Bob's previous $g^{B_{i-1}}$ to make a new value $(g^{B_{i-1}})^{A_i}$.
- And runs the KDF again to make rk_{i+1} and ck_{i+1}.
- Bob does matching updates on his side.
- We do a half-DH exchange with every message!
- Each DH value is used in two distinct DH computations.

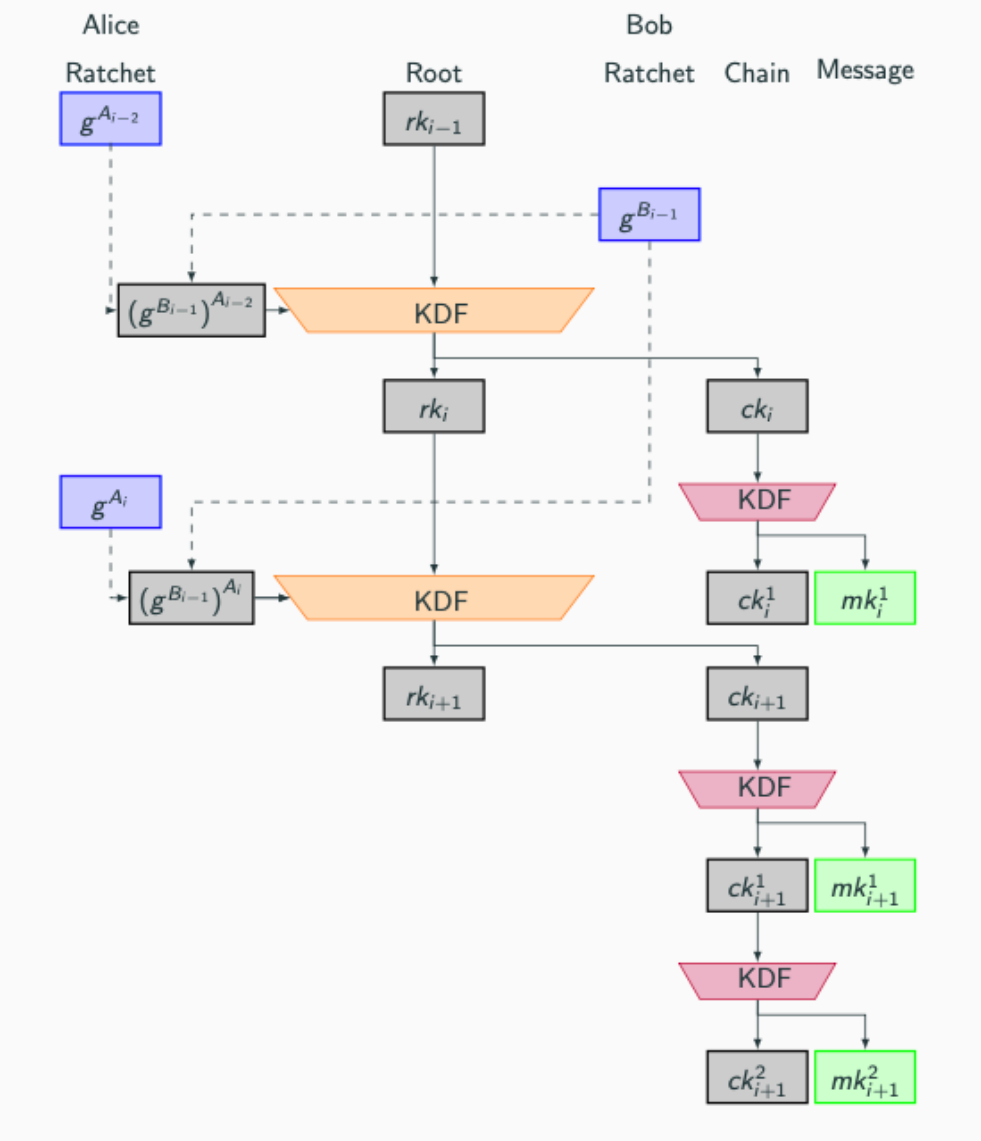
APPLIED CRYPTO GROUP

Solution: Asynchronous Ratcheting



APPLIED CRYPTO GROUP

Putting it All Together: The Signal Double Ratchet Protocol

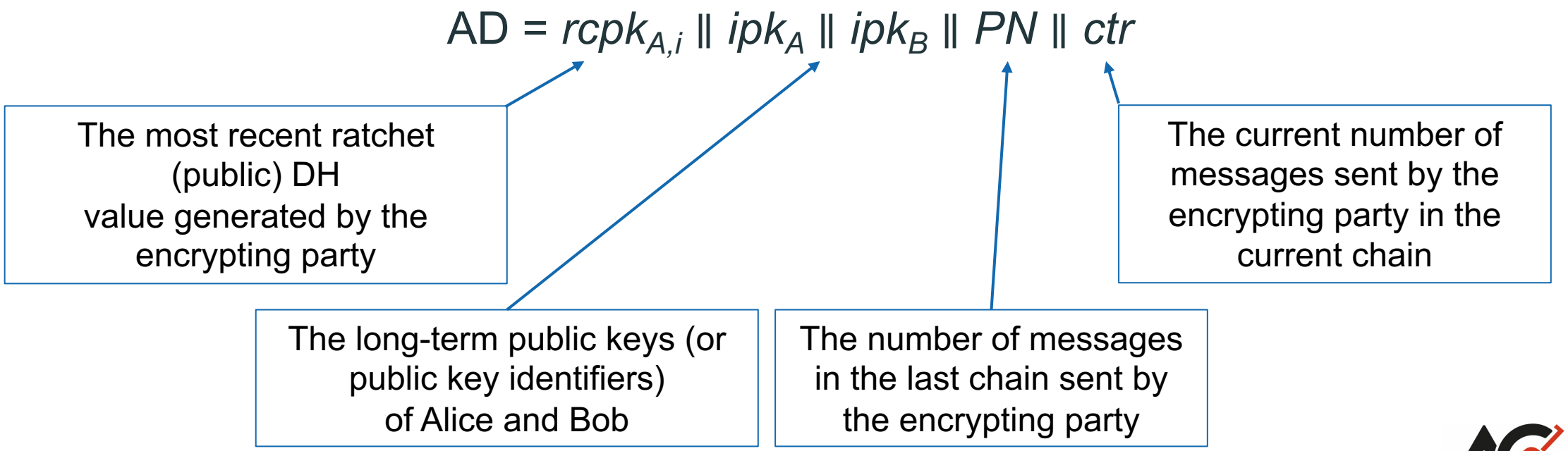


- High level: asymmetric ratchet happens on ping-pong, symmetric ratchet happens on successive messages in one direction.
- FS for all message keys *mk* due to combination of ratchets.
- PCS for message keys *mk* after 2 asymmetric ratcheting steps (equivalent to a single interactive DH exchange).
- Complexity due to possibility of out-of-order delivery of messages (need to cache small number of keys, compromising formal FS guarantees).
- Complexity due to simultaneous transmission (e.g. Alice sends another message after sym ratchet, while Bob sends with asym ratchet at same time).

APPLIED CRYPTO GROUP

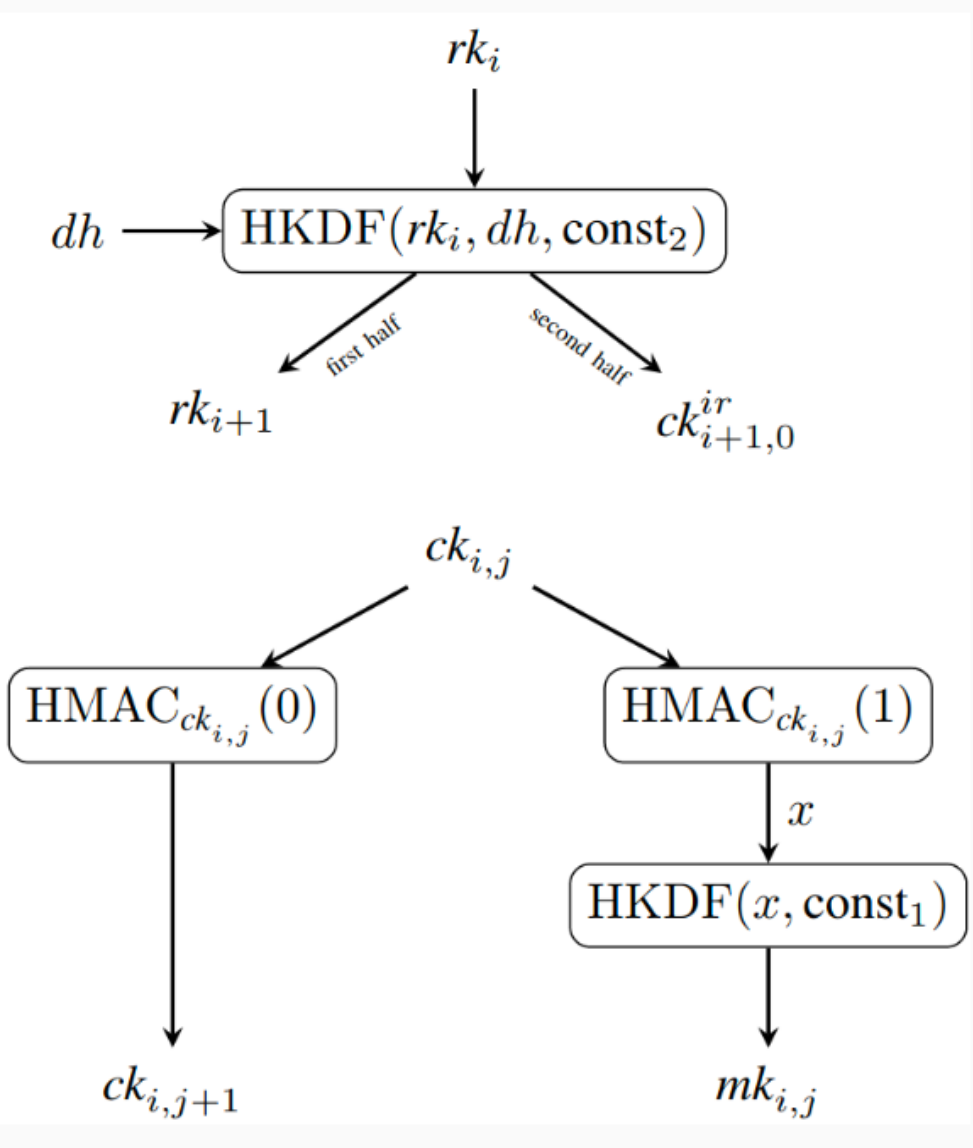
Signal Message Encryption

- Signal uses AEAD encryption:
  - Message is encrypted and integrity protected.
  - Associated data (AD) is integrity protected but not encrypted.



APPLIED CRYPTO GROUP

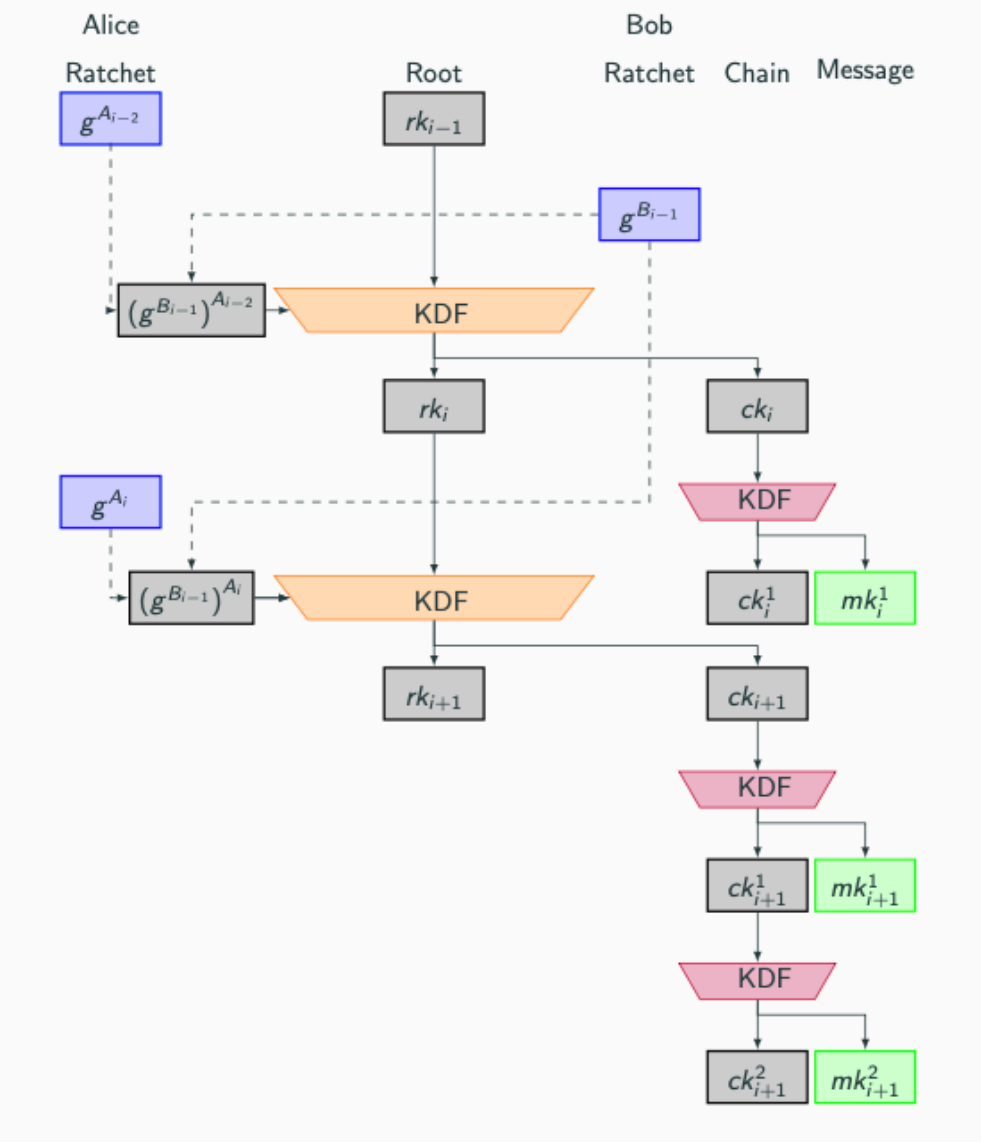
Signal's Cryptographic Primitives



- Key Derivation: Root KDF uses HKDF-SHA256, Chain KDF uses HMAC-SHA256, HKDF-SHA256
- Hash Functions: SHA256
- AEAD Encryption by way of Encrypt-then-MAC: AES256-CBC with PKCS#7 padding, HMAC-SHA256 for MAC
- Signature Scheme: Ed25519
- Diffie-Hellman: Elliptic curve with X25519 or X448

APPLIED CRYPTO GROUP

Formal Security Analysis



- CCDGS17: “A Formal Security Analysis of the Signal Messaging Protocol” – First formal analysis of the Signal Protocol in computational model.
- KBB2017: “Automated verification for secure messaging protocols and their implementations.”
- RMS2018: “More is less: on the end-to-end security of group chats in Signal, WhatsApp, and Threema.”
- ACY2019: “The Double Ratchet: Security notions, proofs, and modularization for the Signal Protocol.”
- …
- Still a very active research area!
- Has inspired theoretical analyses of ratcheting-based protocols more generally.

APPLIED CRYPTO GROUP

Agenda

- Secure Messaging
- The Good: Signal

We are here

- The Bad: Bridgefy
- The Ugly: Telegram
- Closing remarks

APPLIED CRYPTO GROUP

Agenda

- Secure Messaging
- The Good: Signal

- The Bad: Bridgefy ← We are here
- The Ugly: Telegram
- Closing remarks

APPLIED CRYPTO GROUP

The Bad: Bridgefy

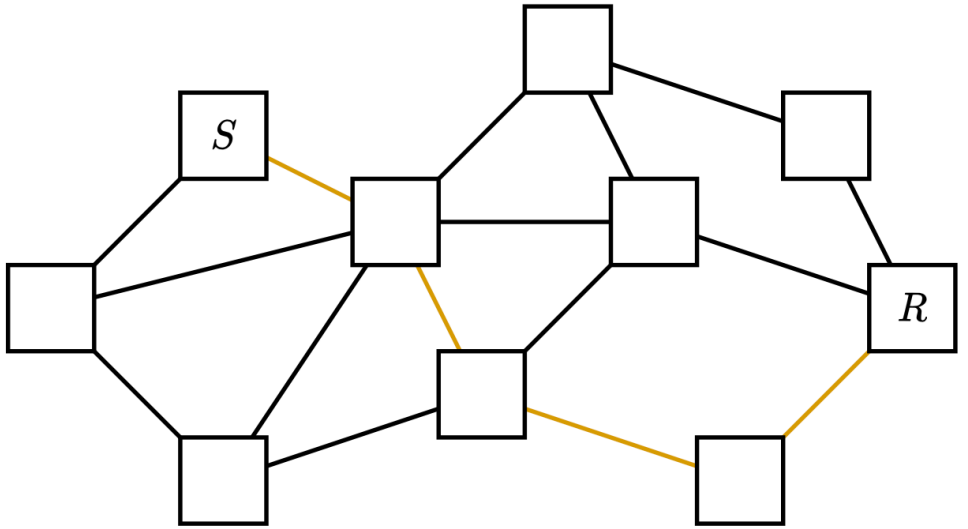
bridgefy

OFFLINE MESSAGING

Bridgefy is the app that lets you send offline text messages when you don't have access to the Internet, by simply turning on Bluetooth. Perfect for connecting with others during natural disasters, at large events, and at school!

Download on the App Store

GET IT ON Google Play



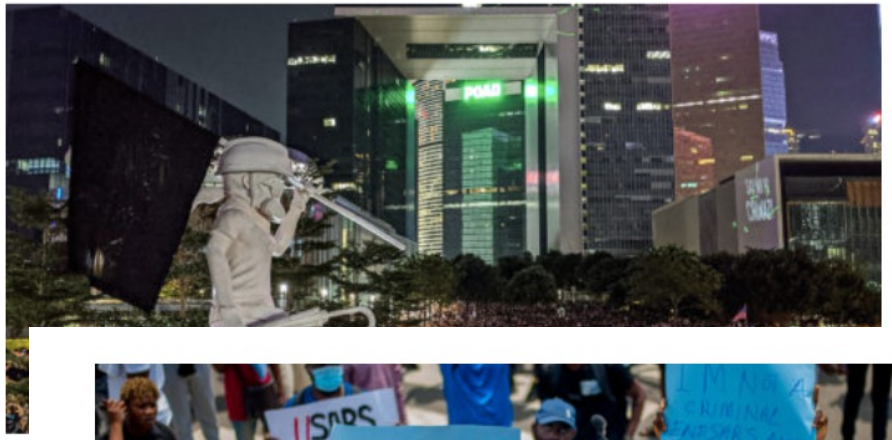
APPLIED CRYPTO GROUP

Bridgefy Actively Promotes Their App for Use by Vulnerable Groups

Myanmar Flocks to Bridgefy to Challenge Military Coup

THE BRIDGEFY APP IN REAL LIFE 05/13/2021

Probably inspired by previous movements, Myanmar massively downloaded the Bridgefy App after the military took the government.



Hong Kong Protesters Adopt Bridgefy amid Sophistication of Techniques

THE BRIDGEFY APP IN REAL LIFE 05/13/2021

Hong Kong protesters adopted the Bridgefy App to escape potential Internet shutdowns as one of several techniques that impressed the

Nigeria Protesters Adopt Bridgefy to Tackle Potential Internet Shutdown

THE BRIDGEFY APP IN REAL LIFE 05/13/2021

Fearing a potential Internet shutdown, Nigerian protesters adopted Bridgefy to tackle it during the #EndSARS demonstrations.

GROUP

bridgefy

Mesh Messaging in Large-Scale Protests: Breaking Bridgefy

Martin R. Albrecht, Jorge Blasco, Rikke Bjerg Jensen, and Lenka Mareková(✉)

Royal Holloway, University of London, London, UK
{martin.albrecht,jorge.blascoalis,rikke.jensen,lenka.marekova}@rhul.ac.uk

CT-RSA 2021
https://eprint.iacr.org/2021/214

APPLIED CRYPTO GROUP

Breaking Bridgefy: [ABB-JM21]

- Ability to track users (can construct social graph)
- No authentication mechanisms (trivial impersonation attacks)
- Lack of resilience against adversarially crafted messages (trivial to DoS the entire network)
- Bleichenbacher-style attack against RSA encryption allowing plaintext recovery:

> *Encryption scheme* One-to-one Bluetooth (mesh and direct) messages in Bridgefy, represented as MessagePacks, are first compressed using Gzip and then encrypted using RSA with PKCS#1 v1.5 padding. The key size is 2048 bits and the input is split into blocks of size up to 245 bytes and encrypted one-by-one in an ECB-like fashion using Java SE's "RSA/ECB/PKCS1Padding", producing output blocks of size 256 bytes. Decryption errors do not produce a user or network visible direct error message.

Bridgefy Adopt `libsignal`

- Bridgefy decided to adopt `libsignal` as a reaction to [ABB-JM21].
- Bridgefy developer team stated:
  - *All messages will be end-to-end encrypted*
  - *A third person will no longer be able to impersonate any other user Man-in-the-middle attacks done by modifying stored keys will no longer be possible*
  - *One-to-one messages sent over the mesh network will no longer contain the sender and receiver IDs in plain text*
  - *A third person will no longer be able to use the server's API to learn others' usernames*
  - *All payloads will be encrypted*
  - *Historical proximity tracking will not be possible*

  - *This basically means that all messages and users are now safe from unwanted prying eyes. [...] We are aware of the tremendous responsibility we have towards our users, and we're committed to improving our security continuously to make sure the chances of attacks are reduced even further.*

APPLIED CRYPTO GROUP

bridgefy

Mesh Messaging in Large-Scale Protests: Breaking Bridgefy

Martin R. Albrecht, Jorge Blasco, Rikke Bjerg Jensen, and Lenka Mareková(✉)

Royal Holloway, University of London, Lond
{martin.albrecht,jorge.blascoalis,rikke.jensen,lenka

CT-RSA 2021
https://eprint.iacr.org/2021/214

Breaking Bridgefy, again: Adopting libsignal is not enough

Martin R. Albrecht
Information Security Group,
Royal Holloway, University of London

Raphael Eikenberg
Applied Cryptography Group,
ETH Zurich

Kenneth G. Paterson
Applied Cryptography Group,
ETH Zurich

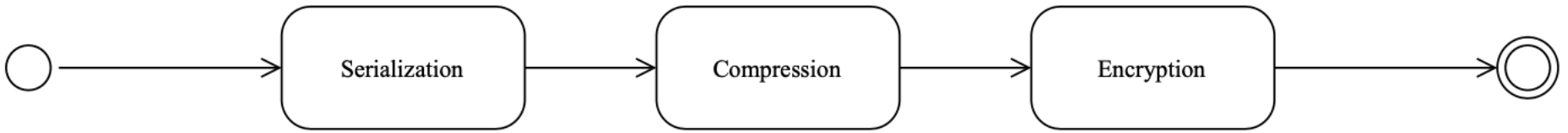
USENIX Security 2022
https://eikendev.github.io/breaking-bridgefy-again/

APPLIED CRYPTO GROUP

Breaking Bridgefy Again

Packet processing steps:

- Serialised using MessagePack
- Compressed using gzip
- Encrypted using `libsignal` (peer-to-peer) or AES-ECB (broadcast)



APPLIED CRYPTO GROUP

Breaking Bridgefy Again: Encryption Overview

Data Category	Handshake	Broadcast	Multi-hop	One-to-one
Metadata	AES-ECB	AES-ECB	AES-ECB	AES-ECB
Payload	AES-ECB	AES-ECB	libsignal	libsignal

- Each app employing the Bridgefy SDK is assigned a dedicated shared key for broadcast messages.
  - All devices using that app have the shared key.
- The main Bridgefy messenger is publicly available.
  - So any adversary knows the shared key of the Bridgefy messenger!
- Even in a bespoke app based on the SDK, a fixed broadcast key is used throughout the network.
  - So an adversary compromising one node of the network can learn the key!

APPLIED CRYPTO GROUP

Breaking Bridgefy Again: Handshake Protocol

$$
\begin{aligned}
A \to B: &\quad \texttt{ResponseTypeGeneral}(\mathrm{userId}_A) && (1)\\
B \to A: &\quad \texttt{ResponseTypeGeneral}(\mathrm{userId}_B) && (2)\\
A \to B: &\quad \texttt{ResponseTypeKey}(\mathrm{PKB}_A) && (3)\\
B \to A: &\quad \texttt{ResponseTypeKey}(\mathrm{PKB}_B) && (4)
\end{aligned}
$$

- Bridgefy adopts a “Trust on First Use” (TOFU) approach.
- Exchange of keys is not authenticated.
- Trivial attacks for a MITM adversary who is present at the first key exchange.
- (How is this attack prevented in Signal?)

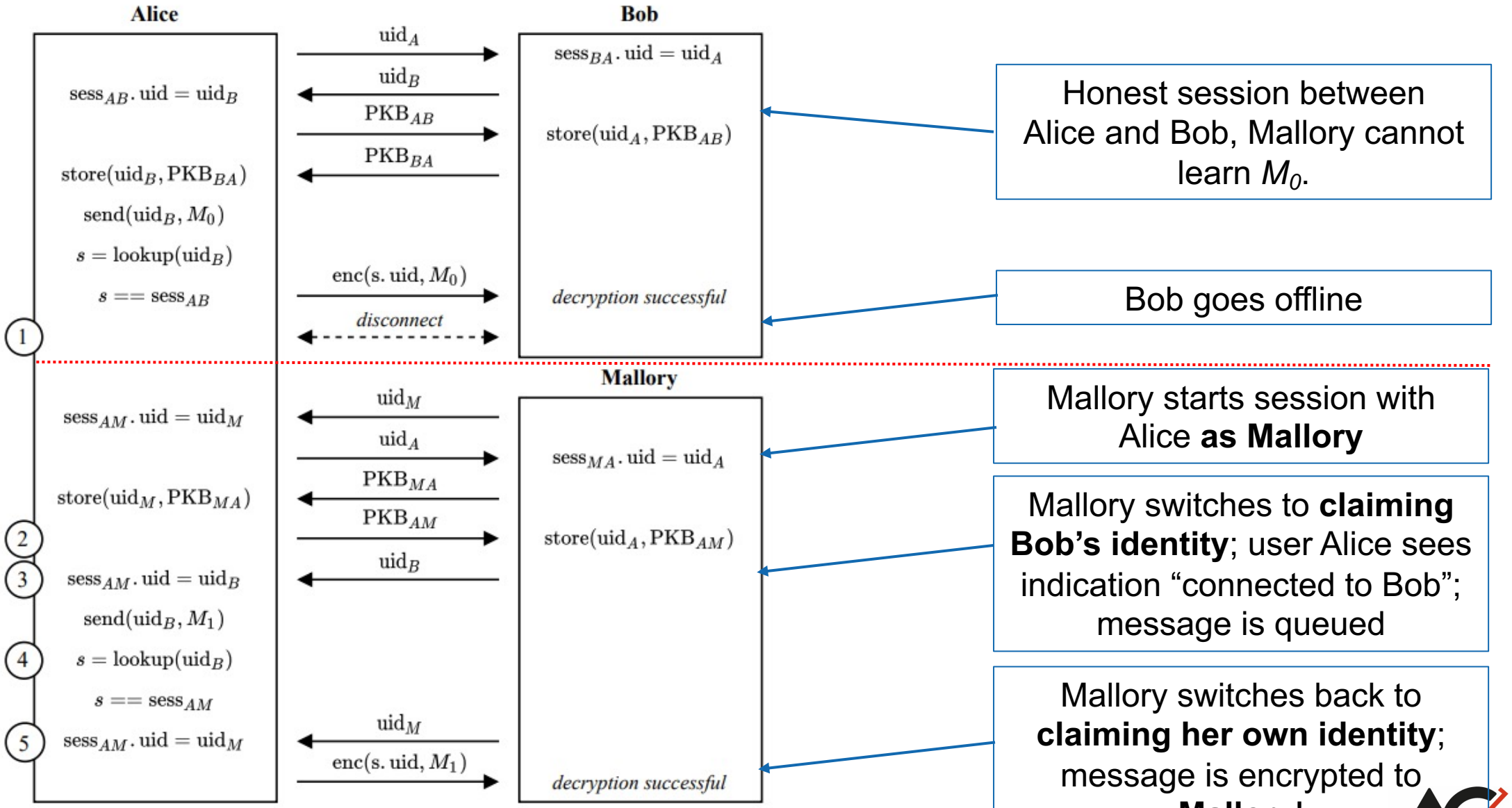
Breaking Bridgefy Again: TOCTOU Attack

- An attack breaking confidentiality was still possible, even if the TOFU attack opportunity was missed.
- Based on mis-integration of `libsignal`.
- An instance of a TOCTOU attack: **T**ime **O**f **C**heck to **T**ime **O**f **U**se:

1. Mallory establishes a Bluetooth session and keys with Alice as Mallory.
2. Mallory then claims Bob's ID in that session, displayed to Alice.
3. Alice queues a message for encryption to Bob.
4. But then Mallory switches back to claiming Mallory's ID in that session.
5. When message is dequeued, it is encrypted by `libsignal` to Mallory instead of Bob!

APPLIED CRYPTO GROUP

Breaking Bridgefy Again: TOCTOU Attack



Honest session between Alice and Bob, Mallory cannot learn M_0.

Bob goes offline

Mallory starts session with Alice **as Mallory**

Mallory switches to **claiming Bob's identity**; user Alice sees indication "connected to Bob"; message is queued

Mallory switches back to **claiming her own identity**; message is encrypted to **Mallory**!

APPLIED CRYPTO GROUP

Breaking Bridgefy Again: Broadcast Encryption Attack

- For completeness, we also broke the ECB-mode broadcast encryption mechanism in Bridgefy.

APPLIED CRYPTO GROUP

ECB information leakage

Tux the Penguin, the Linux mascot. Created in 1996 by Larry Ewing with The GIMP. lewing@isc.tamu.edu

ECB-Tux

Breaking Bridgefy Again: Broadcast Encryption Attack

- Many header fields are unpredictable, these influence the subsequent compression, so **ECB encryption is not fully deterministic**.
- Degree of compression obtained depends on payload (unknown target) and header fields (including hop count).
- Attacker observes a **vector** of packet lengths as a payload traverses the network.
- Attacker observes a **collection of vectors** if the same payload is transmitted many times.
- Given such a collection, attacker performs Bayesian inference to find most likely payload.
- This actually works. ☺

Breaking Bridgefy Again: Disclosure & Remediation

- We notified Bridgefy about the Signal TOCTOU attack on 2021-05-21.
- Bridgefy confirmed receipt soon after.
- But they informed us on 2021-07-21 they will not publicly address it.
- They promised to remove the term ‘end-to-end’ from their marketing.
- The TOCTOU exploit stopped working in version 3.1.7 released on 2021-08-14.
- We disclosed the attacks on Bridgefy’s broadcast encryption mechanism on 2021-09-07.
- On 2021-09-09, the developers informed us that they were aware of the vulnerability and were actively working on fixing it.
- We asked the developers to comment on the remediation progress on 2022-02-04; no reply.
- State of the remediation remains unclear today (2022-06-14).

APPLIED CRYPTO GROUP

Bridgefy Public Statement: 2021-07-16

bridgefy

Home FAQ Blog SDK

Our commitment to the privacy and security of all Bridgefy users

PRIVACY AND SECURITY 07/16/2021

At Bridgefy, we're constantly looking for new and better ways to improve our users' lives: by making our app easier to use, improving our website, or by updating the Bridgefy SDK to allow developers to integrate it quicker. Our responsibility increases along with our user base, and we're excited to see our company grow into areas we never imagined.

Bridgefy was originally created to help people communicate with each other before/during/after natural disasters and music festivals, but it has evolved into much more. Since mid-2019 we've seen Bridgefy being used in more than 170 countries we never dreamt would adopt our technology- during all kinds of natural disasters, as we once envisioned, but also during social unrest movements, protests and gatherings. This lead to us having to pay closer attention to how we protected our users.

APPLIED CRYPTO GROUP

Bridgefy Public Statement: 2021-07-16

Technology is ever-changing. What once worked to keep systems secure stops working soon after- which is why every single organization using technology is perpetually at risk of having its security systems penetrated. The US Federal Government, Apple, Microsoft and Netflix, Intel and Cisco, and even meat suppliers have been affected recently, despite spending millions (and sometimes billions) of dollars on cybersecurity. It just goes to show that privacy, security, and anonymity have to constantly be improving and evolving, regardless of the size of your company.

During late 2020, we did a complete revamp of the security around Bridgefy. The best, quickest and easiest way for us to significantly advance in this department was to start working on implementing the Signal Protocol, an advanced encryption protocol that even Whatsapp, Skype, and Facebook Messenger use. The true challenge was modifying the Signal Protocol so that it would also work offline, and we made gigantic internal breakthroughs in this aspect. We were happy that our app was encrypted* to the best of our capacity.

Now that Bridgefy has passed the 6-million downloads mark, our pledge to do our best at keeping our users safe is reiterated. We're launching versions of the Bridgefy App and the Bridgefy SDK that will contain significant security improvements to protect the wellbeing of our users and customers. We're giddy with excitement about what's coming this year, and can't wait to get feedback from our beloved users! And remember: try to avoid sharing sensitive information through any digital means, as no technical product is 100% safe!

APPLIED CRYPTO GROUP

Agenda

- Secure Messaging
- The Good: Signal

- The Bad: Bridgefy
- The Ugly: Telegram ⟵ We are here
- Closing remarks

APPLIED CRYPTO GROUP

The Ugly: Telegram

Telegram

a new era of messaging

Telegram for **iPhone** / **iPad**

General Questions

Q: What is Telegram? What do I do here?

Telegram is a messaging app with a focus on speed and security, it's super-fast, simple and free. You can use Telegram on all your devices **at the same time** — your messages sync seamlessly across any number of your phones, tablets or computers. Telegram has over **500 million** monthly active users and is one of the **10 most downloaded apps** in the world.

APPLIED CRYPTO GROUP

The Ugly: Telegram

Four Attacks and a Proof for Telegram

Martin R. Albrecht*, Lenka Mareková*, Kenneth G. Paterson† and Igors Stepanovs†
*Information Security Group, Royal Holloway, University of London, {martin.albrecht,lenka.marekova.2018}@rhul.ac.uk
†Applied Cryptography Group, ETH Zurich, {kenny.paterson,istepanovs}@inf.ethz.ch

IEEE Security and Privacy Symposium 2022

https://mtpsym.github.io/

APPLIED CRYPTO GROUP

Telegram's Popularity

Monthly active users in Jan 2022:

According to Statistica 2022.

WhatsApp	$2000 \cdot 10^6$
WeChat	$1263 \cdot 10^6$
FB Messenger	$988 \cdot 10^6$
QQ	$574 \cdot 10^6$
Snapchat	$557 \cdot 10^6$
Telegram	$550 \cdot 10^6$

Collective Information Security in Large-Scale Urban Protests: the Case of Hong Kong

Martin R. Albrecht, Jorge Blasco, Rikke Bjerg Jensen, and Lenka Mareková, *Royal Holloway, University of London*

Predominant in Hong Kong protests.

Perceived more secure than competitors.

Advantages of **Telegram**:

Group chats for up to 200000 people.

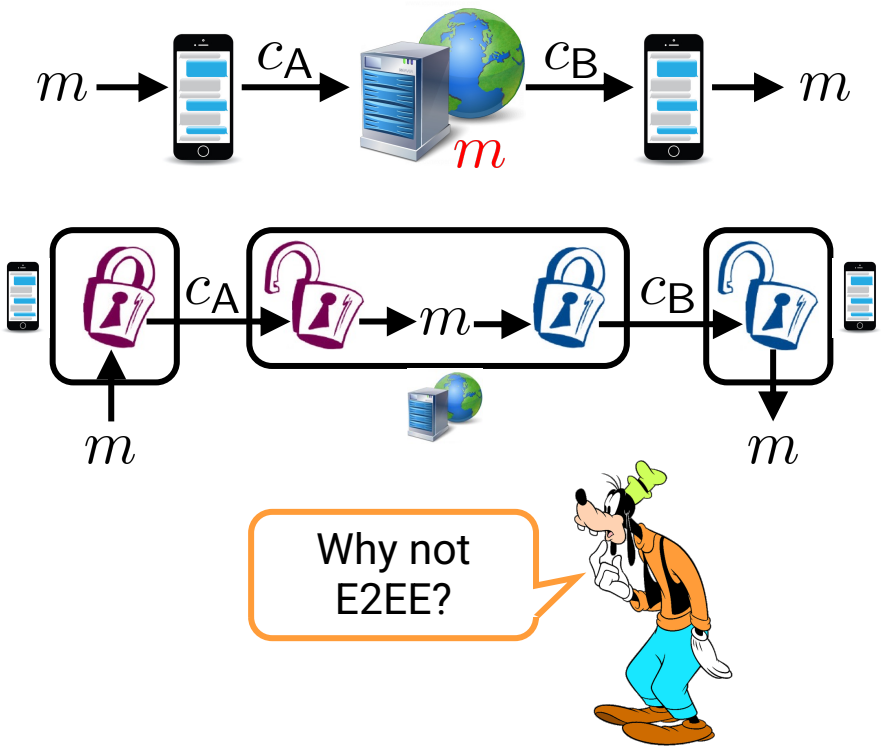
Support of pseudonyms in group chats.

Other features:

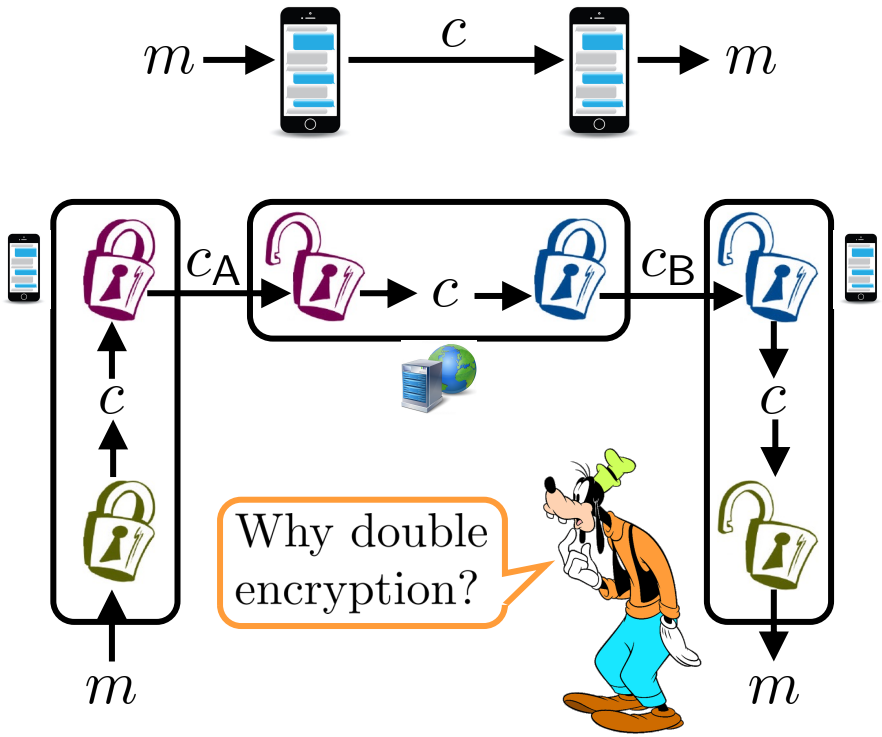
> ..., anonymous polls, disappearing messages, timed or scheduled messages, ability to delete messages sent by others, ...

Telegram: Cloud Chats and Secret Chats

Cloud Chats



Secret Chats

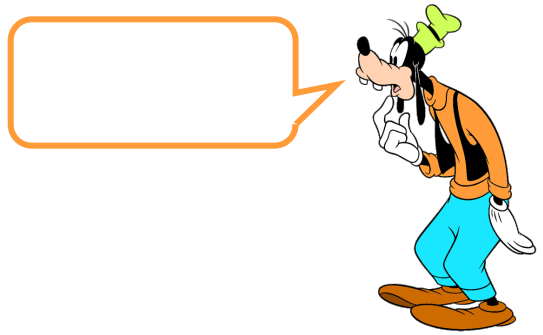


Encryption Decryption

Telegram: Cloud Chats and Secret Chats

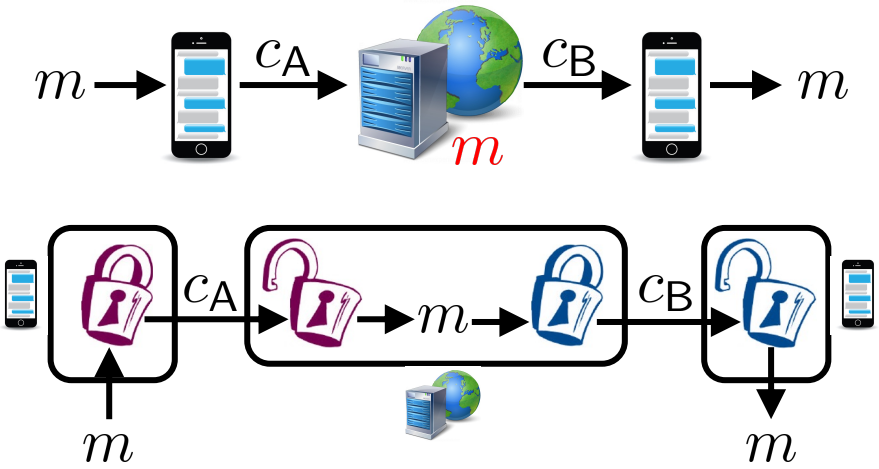
	Cloud Chats	Secret Chats
Encryption	client-server	end-to-end
Groups	✓	✗
1-on-1	✓	✓
Enabled by default	✓	✗

- Default usage for **Telegram is NOT E2EE**.
- E2EE facility exists but needs to be enabled.
- E2EE not available at all for groups.
- So how good is Telegram's infrastructure security?



Telegram: Cloud Chats and MTProto 2.0

Cloud Chats



MTProto 2.0

MTProto 2.0 is **Telegram**'s equivalent of the TLS record protocol.

- 2013 **Telegram** launched with MTProto 1.0.
- 2016 MTProto 1.0 is not CCA-secure [JO16].
- 2017 Input validation bug (message replay) [SK17].
  Telegram released MTProto 2.0.
- 2018 Input validation bug in key exchange [K18].
- 2020 MTProto 2.0 secure in symbolic model [MV20].
  (assuming ideal building blocks)

MTProto 2.0 is not well-studied.

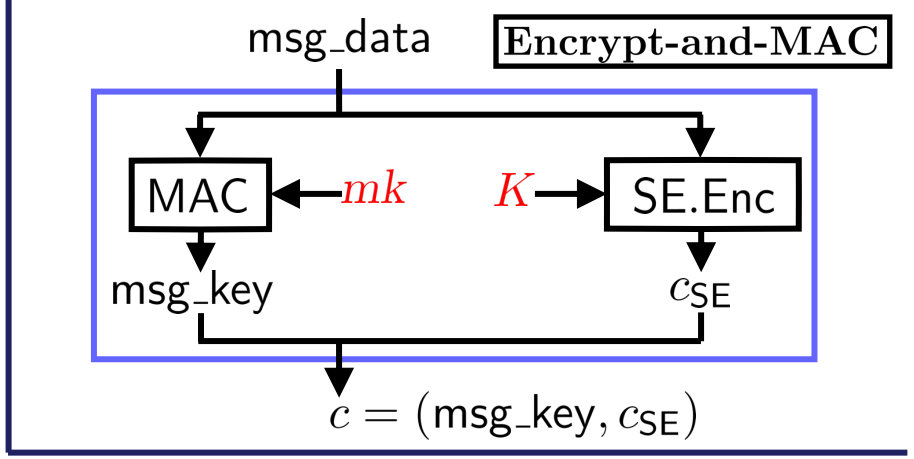
Telegram: Cloud Chats and MTProto 2.0

https://core.telegram.org/techfaq

Telegram FAQ

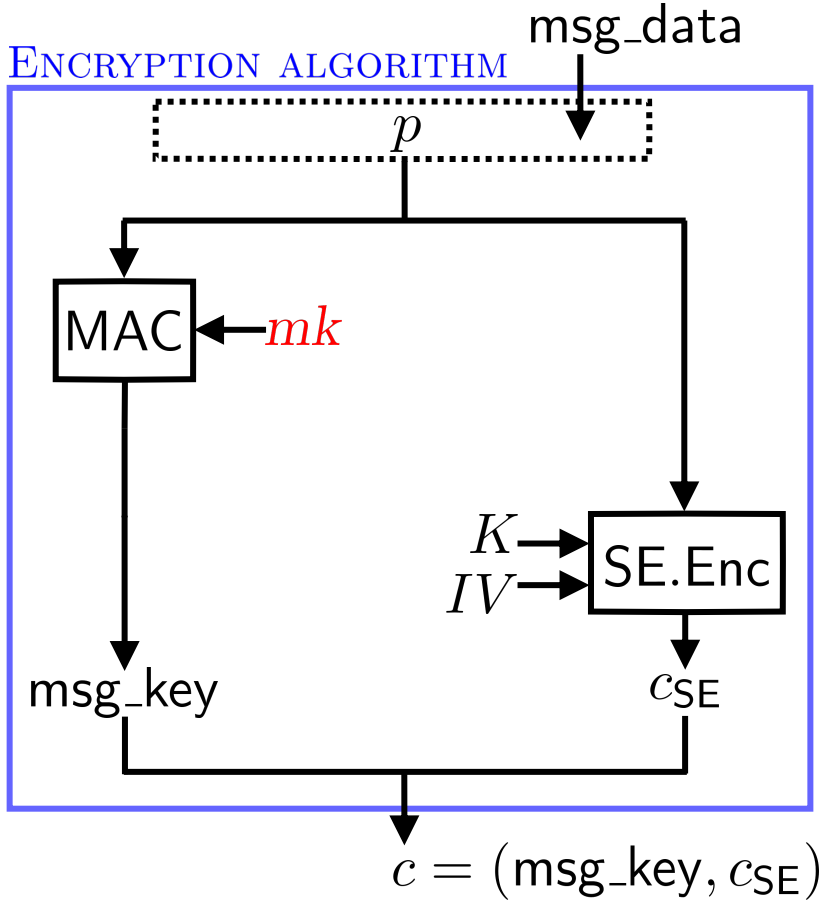
Q: Why are you not using X? (insert solution)
While other ways of achieving the same cryptographic goals, undoubtedly, exist, we feel that the present solution is both robust and also succeeds at our secondary task of beating unencrypted messengers in terms of delivery time and stability.

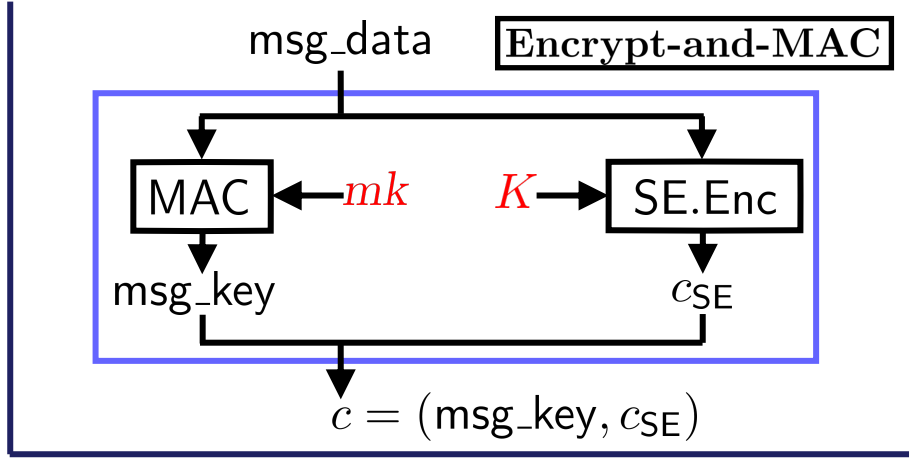
The Design of MTProto 2.0



MAC – Message Authentication Code
SE – Symmetric Encryption Scheme

The Design of MTProto 2.0



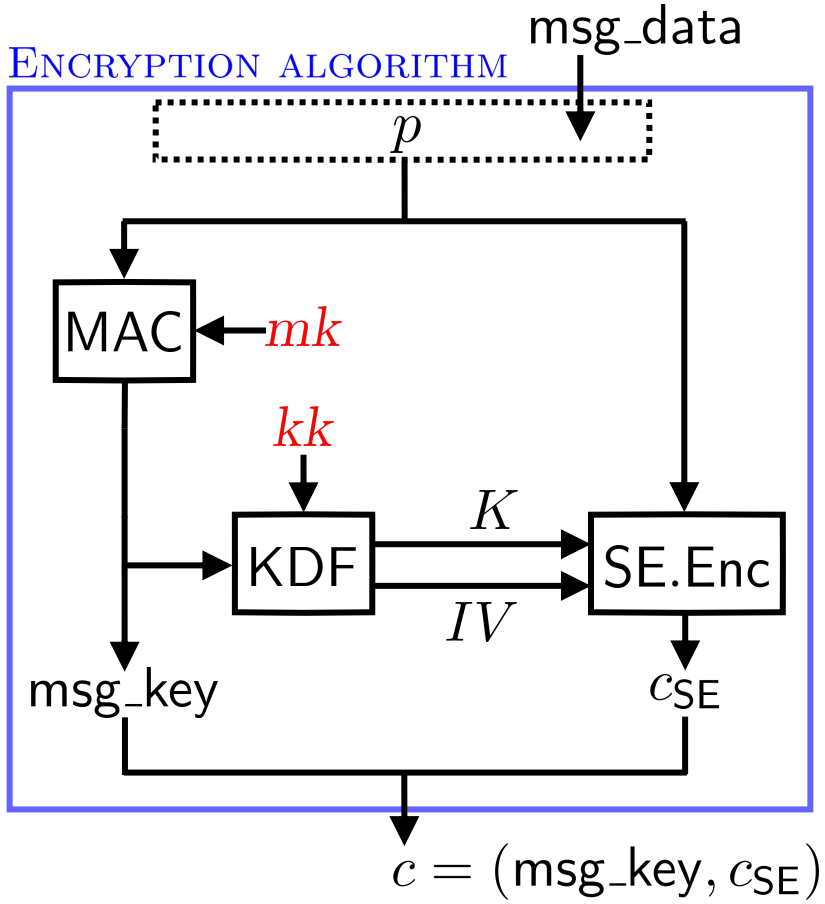


MAC – Message Authentication Code
SE – Symmetric Encryption Scheme

payload p

Random values:	server_salt	64 bits
	session_id	64 bits
Message sequence number:	msg_seq_no	96 bits
Message length:	msg_length	32 bits
Message body:	msg_data	$\leq$ 16 MB
Random padding:	padding	12-1024 bytes

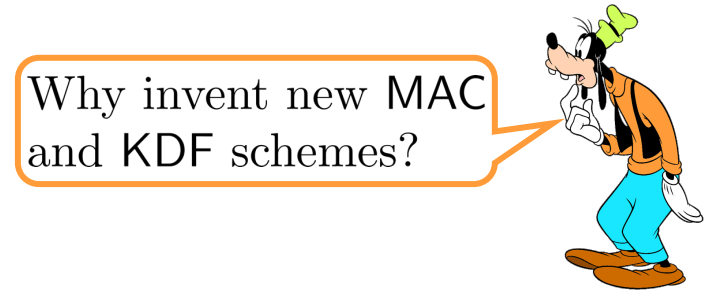
The Design of MTProto 2.0



MTProto defines **ad-hoc** MAC and KDF schemes.

$\underline{\mathsf{MAC}(mk, p)}$
$\mathsf{msg\_key} \leftarrow \mathsf{SHA\text{-}256}(mk \| p)[64 : 192]$
Return msg_key

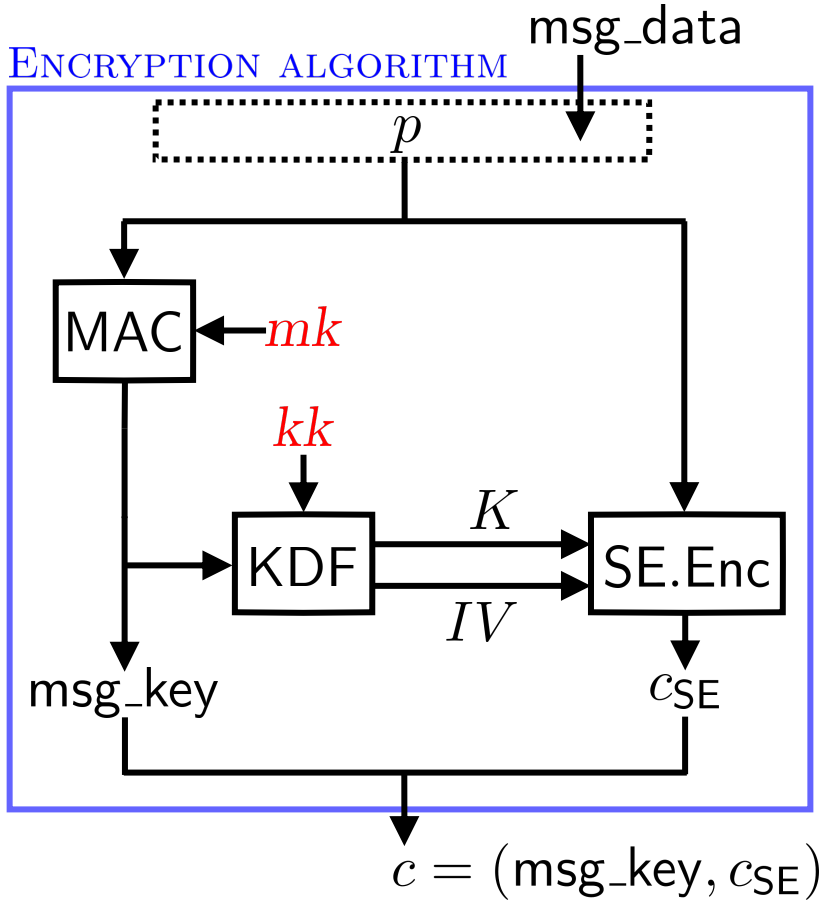
$\underline{\mathsf{KDF}(kk, \mathsf{msg\_key})}$
$(kk_0, kk_1) \leftarrow kk$
$K \leftarrow \mathsf{SHA\text{-}256}(\mathsf{msg\_key} \| kk_0)$
$IV \leftarrow \mathsf{SHA\text{-}256}(kk_1 \| \mathsf{msg\_key})$
Return K, IV



ETH zürich

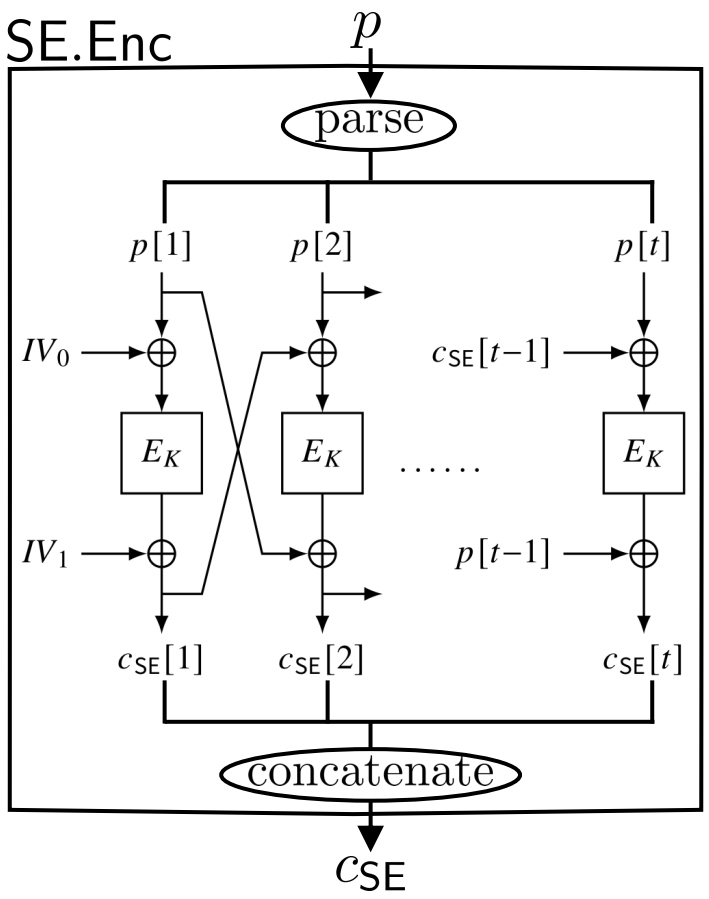
Applied Crypto Group

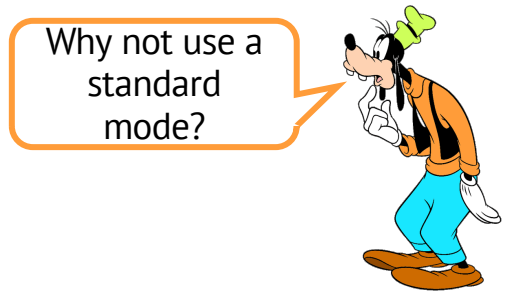
The Design of MTProto 2.0



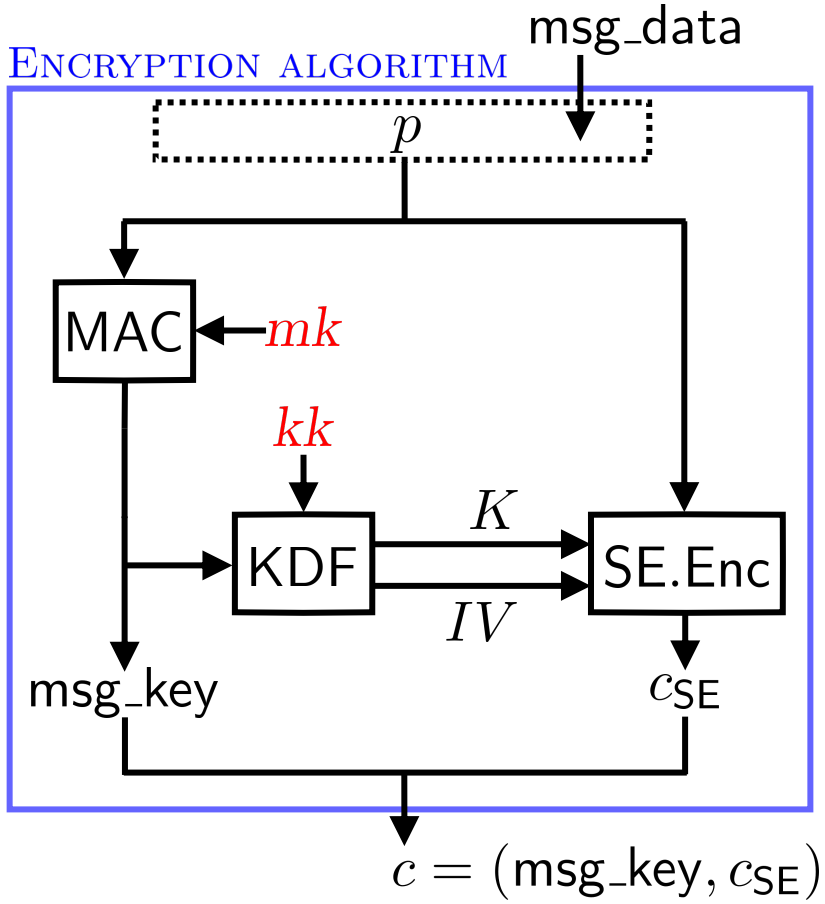
Infinite Garble Extension (IGE)

block cipher mode of operations





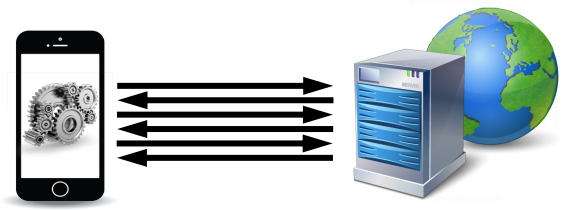
The Design of MTProto 2.0



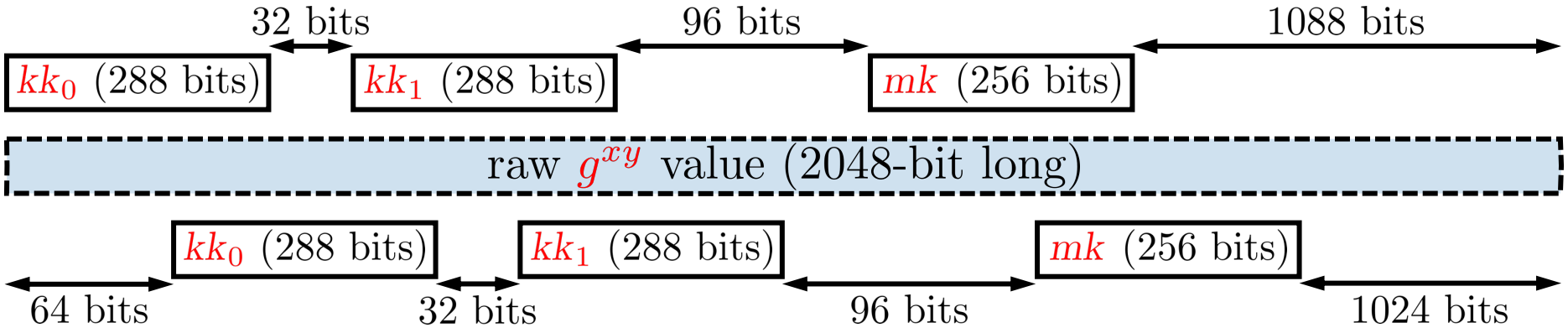
How are mk and kk derived?

The Design of MTProto 2.0

MTProto uses Diffie-Hellman key exchange to agree on a raw shared secret g^{xy}.



Keys used for **client** → **server** encryption.



Keys used for **server** → **client** encryption.

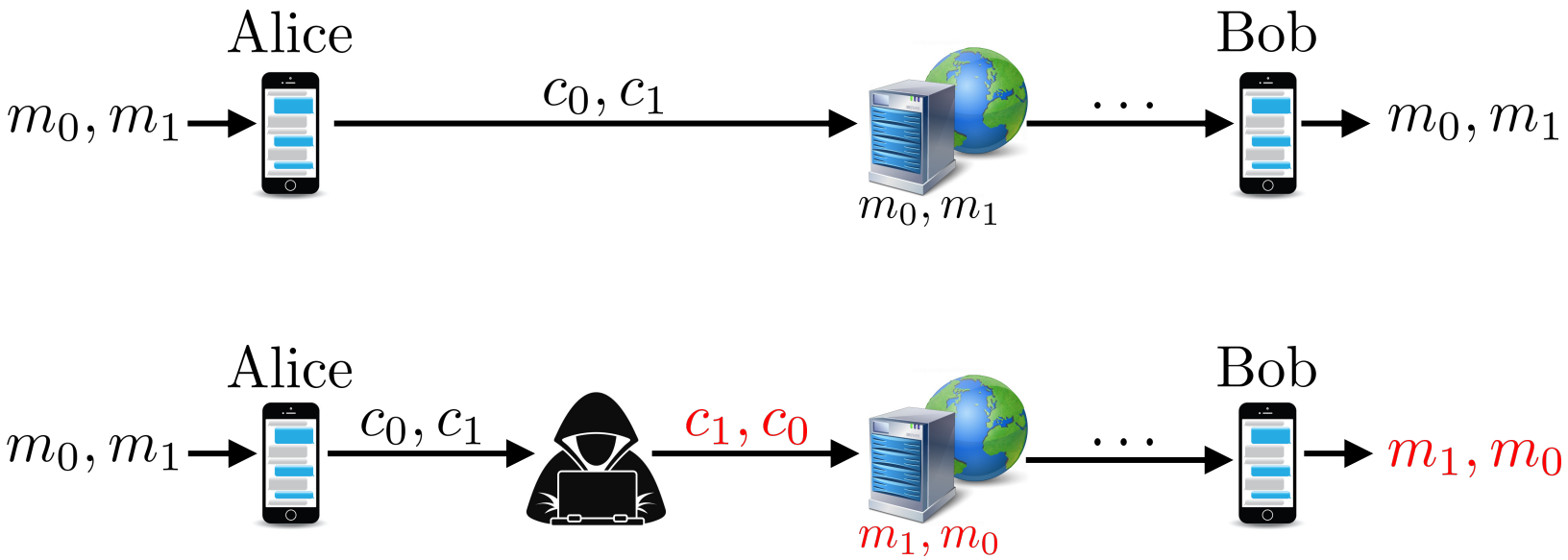
Four Attacks Against MTProto 2.0

April 16, 2021 — We reported 4 vulnerabilities to **Telegram**.
April 22, 2021 — **Telegram** confirmed the receipt of our e-mail.
June 08, 2021 — **Telegram** acknowledged the reported behaviours.
July 16, 2021 — Public disclosure (mutually agreed date).
2021 — **Telegram** awarded bug bounty for attacks and analysis.

All vulnerabilities fixed as of
7.8.1 for **Android**
7.8.3 for **iOS**
2.8.8 for **Desktop**

Telegram informed us that they
. . . do no security/bugfix releases except for post-release crash fixes.
(could not commit to release dates for specific fixes)
(fixes were rolled out as part of regular updates)
. . . did not wish to issue security advisories at the time of patching.

Message Reordering Attack



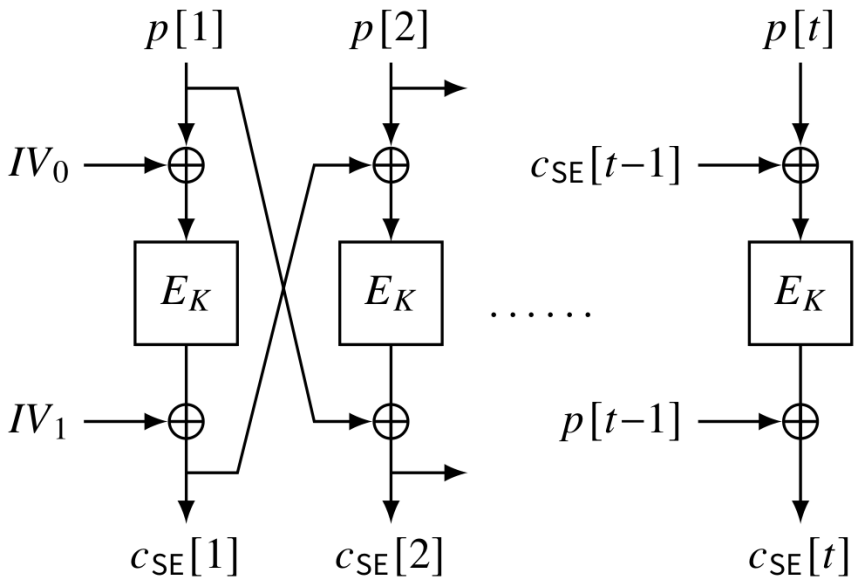
Technically trivial. Easy to exploit.

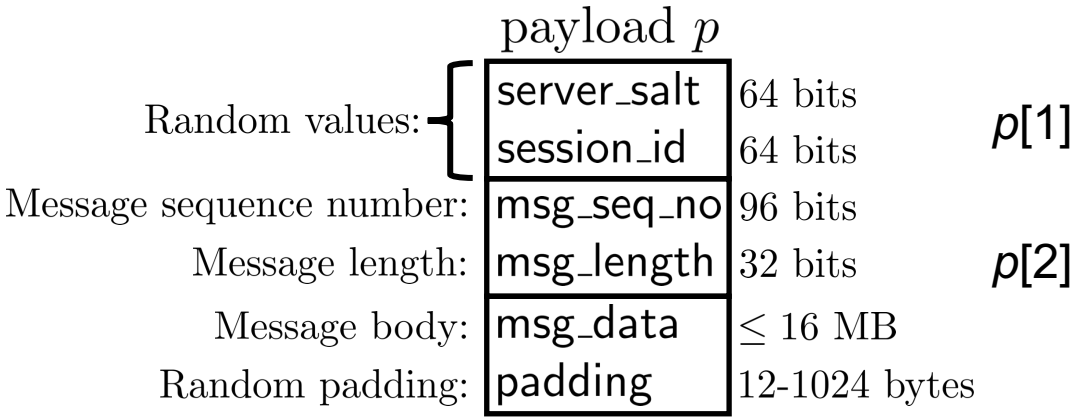
More Theoretical Attacks

1. Plaintext recovering timing attack against E&M construction in MTProto 2.0 based on sanity-checking of length field before MAC check on plaintext during decryption.
   - Similar attack vector to [APW09] on SSH.
   - “Garbling” of IGE thought to prevent it, but IGE is still malleable.
   - Telegram “spec” warns about it but several official clients got it wrong.
2. Timing attack against MTProto 2.0 key exchange protocol allowing recovery of server_salt and message_id (needed for first attack).
3. IND-CPA attack against MTProto 2.0’s message retransmission feature.

   All three attacks are **quite theoretical** but illustrate the fragility of MTProto 2.0 design.

Details of Timing Attack on E&M

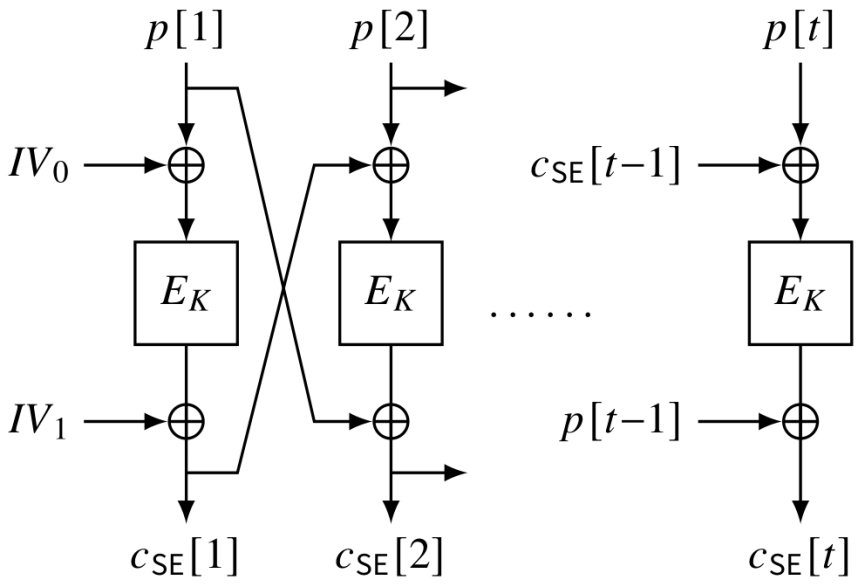


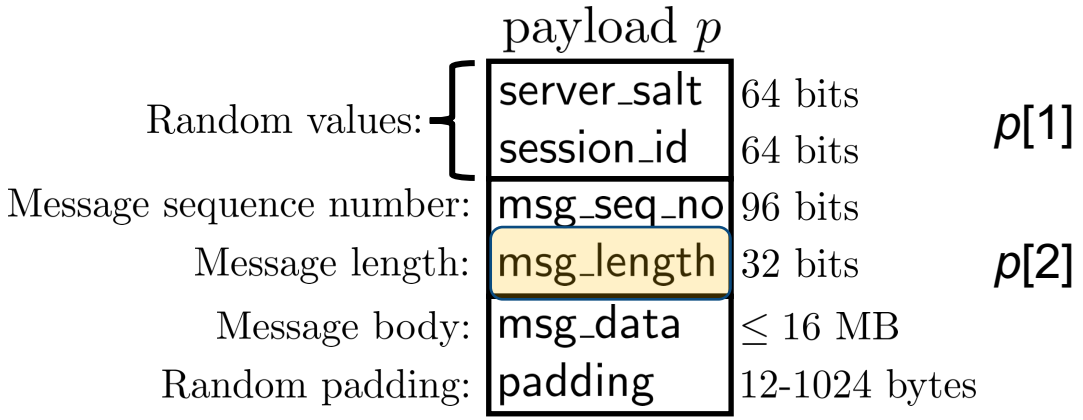


Ideal decryption process:

1. Parse c as (`msg_key`, c_{SE}).
2. Use msg_key to rederive IGE key and IV.
3. Run IGE decryption on c_{SE} to get payload $p = p[1]\ p[2]\ \ldots\ p[t]$.
4. Recompute MAC on p and compare to `msg_key`; accept if they match, otherwise reject.

Details of Timing Attack on E&M



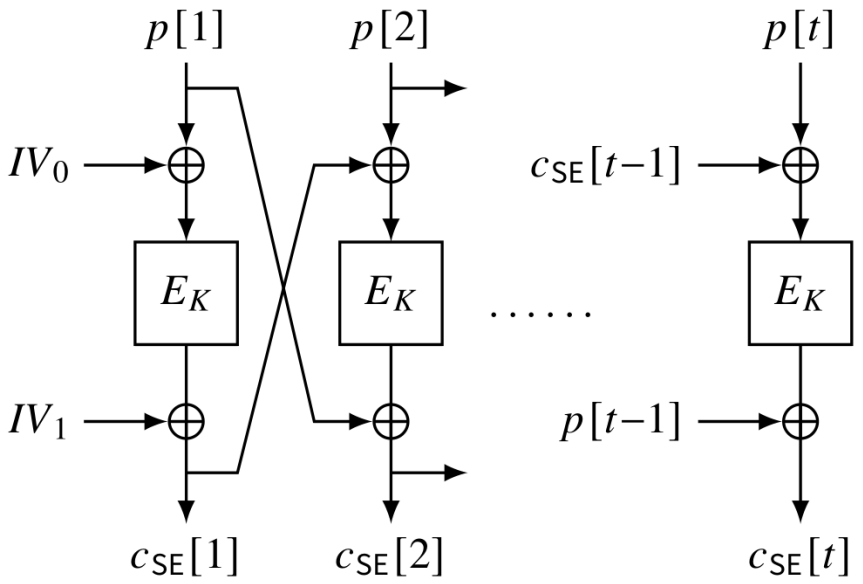


Real-world decryption process:

1. Parse c as (`msg_key`, c_{SE})
2. Use msg_key to rederive IGE key and IV.
3. Run partial IGE decryption on c_{SE} to get partial payload p = p[1] p[2].
4. Sanity check `msg_length` field in p[2]; reject if out of range.
5. Decrypt remaining blocks p[3] p[4] …
6. Recompute MAC on p and compare to `msg_key`; accept if they match, otherwise reject.

Timing side channel!

Details of Timing Attack on E&M



payload p

Random values:	server_salt	64 bits	p[1]
	session_id	64 bits	
Message sequence number:	msg_seq_no	96 bits	
Message length:	msg_length	32 bits	p[2]
Message body:	msg_data	$\leq$ 16 MB	
Random padding:	padding	12-1024 bytes	

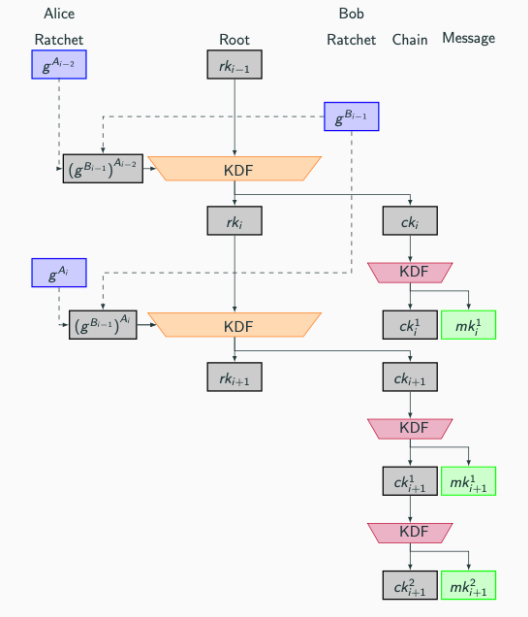
Recovering plaintext from a target block c_{SE}[i]:

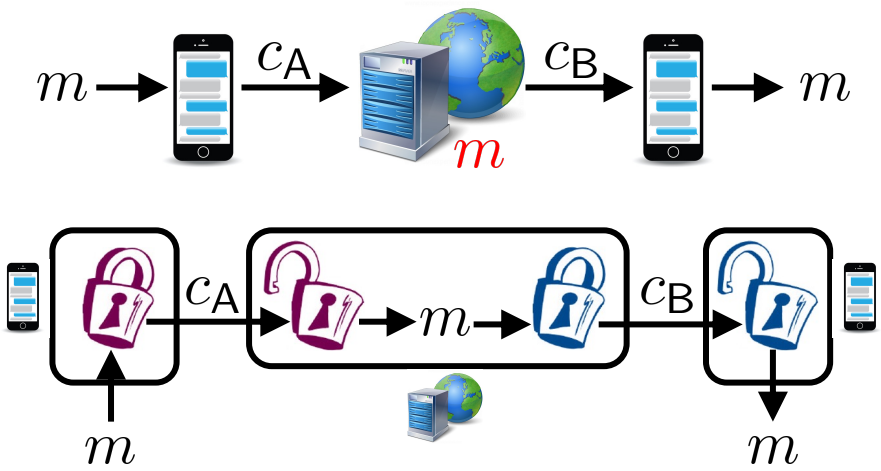
- Suppose blocks p[1] and p[i-1] are known.
- Replace block c_{SE}[2] with c_{SE}[2]' = c_{SE}[i] $\oplus$ p[1] $\oplus$ p[i-1].
- Then c_{SE}[2]' decrypts to p[i] $\oplus$ c_{SE}[i-1] $\oplus$ c_{SE}[1].
- The length sanity check is now done on 32 bits *related* to block p[i].
- Hence timing side channel leaks information on p[i].

Four Attacks **and a Proof?**

- With small tweaks, we obtained a version of MTProto 2.0's secure channel construction that we could prove secure.
- This required new security model for *stateful bidirectional channels* because of lack of independent keying in different directions.
- Standard assumptions needed:
  - Collision resistance of SHA-256 with truncated output
  - One-time IND-CPA security of CBC mode (yields same for IGE)
  - One-time PRF security of SHACAL-1
  - One-time PRF security of the compression function of SHA-256
- New assumptions needed:
  - Related-key PRF security of SHACAL-2 with key leakage
  - Required because of overlapping key bits.

Telegram in Comparison to Signal





Signal

- E2EE by default
- Double-ratchet mechanism
- Fine-grained key derivation for FS and PCS
- Vanilla cryptography: EtM, HMAC, HKDF, etc.

MTProto 2.0 in Telegram

- **Not** E2EE by default
- DH mod p + RSA auth in handshake followed by E&M in MTProto 2.0
- Relatively static keys
- Exotic+old crypto: EE&M, IGE, weird key derivation, mod p DH

What's Next?

Large parts of **Telegram** unstudied:

Secret chats
Key exchange

..., multi-user security, forward secrecy, Telegram Passport, bot APIs, higher-level message processing, control messages, encrypted CDNs, cloud storage, ...

More information at
https://mtpsym.github.io/

Agenda

- Secure Messaging
- The Good: Signal

- The Bad: Bridgefy
- The Ugly: Telegram
- Closing remarks ← We are here

APPLIED CRYPTO GROUP

Closing Remarks

- Secure communications still presents interesting research challenges despite >2000 years of research.
- Studying cryptography deployed at scale: interesting targets, impactful results, identification of research problems not yet solved in practice.
- Don’t roll your own crypto algorithms → don’t roll your own protocol (*Telegram*).
- Don’t roll your own protocol → don’t roll your own protocol integration (*Bridgefy*).
- Enjoy the rest of the summer school.

ETH zürich

APPLIED CRYPTO GROUP

Contact:
Kenny Paterson
Applied Cryptography Group
kenny.paterson@inf.ethz.ch

ETH Zurich
Applied Cryptography Group
Department of Computer Science
Universitätstrasse 6
8092 Zurich, Swizterland

https://appliedcrypto.ethz.ch/

APPLIED CRYPTO GROUP