MLS BOF

Messaging Layer Security

Chair Slides:

Nick & Sean

IETF 101

NOTE WELL

This is a reminder of IETF policies in effect on various topics such as patents or code of conduct. It is only meant to point you in the right direction. Exceptions may apply. The IETF's patent policy and the definition of an IETF "contribution" and "participation" are set forth in BCP 79; please read it carefully.

As a reminder:

- By participating in the IETF, you agree to follow IETF processes and policies.
- If you are aware that any IETF contribution is covered by patents or patent applications that are owned or controlled by you or your sponsor, you must disclose that fact, or not participate in the discussion.
- As a participant in or attendee to any IETF activity you acknowledge that written, audio, video, and photographic records of meetings may be made public.
- Personal information that you provide to IETF will be handled in accordance with the IETF Privacy Statement.
- As a participant or attendee, you agree to work respectfully with other participants; please contact the ombudsteam (https://www.ietf.org/contact/ombudsteam/) if you have questions or concerns about this.

Definitive information is in the documents listed below and other IETF BCPs. For advice, please talk to WG chairs or ADs:

- BCP 9 (Internet Standards Process)
- BCP 25 (Working Group processes)
- BCP 25 (Anti-Harassment Procedures)
- BCP 54 (Code of Conduct)
- BCP 78 (Copyright)
- BCP 79 (Patents, Participation)
- https://www.ietf.org/privacy-policy/ (Privacy Policy)

Requests

Minute Taker(s)

Jabber Scribe(s)

Sign Blue Sheets

Info

List: https://www.ietf.org/mailman/listinfo/mls

Jabber: mls@jabber.ietf.org

Agenda

5min	Agenda bashing
30min	Problem statement
15min	Architecture
15min	(draft) Protocol
15min	State of formal analysis
30min	Charter text discussion
10min	BOF questions

Problem Statement

Messaging Layer Security

Problem Statement

Context

Lots of secure messaging apps

Some use similar protocols...

... some are quite different

Wildly different levels of analysis

Everyone maintaining their own libraries

Goals

Detailed specifications for an **async group messaging security protocol**

Code that is reusable in multiple contexts

Robust, open security analysis and involvement from the academic community

Non-goal: Application-level interoperability

What do we want?

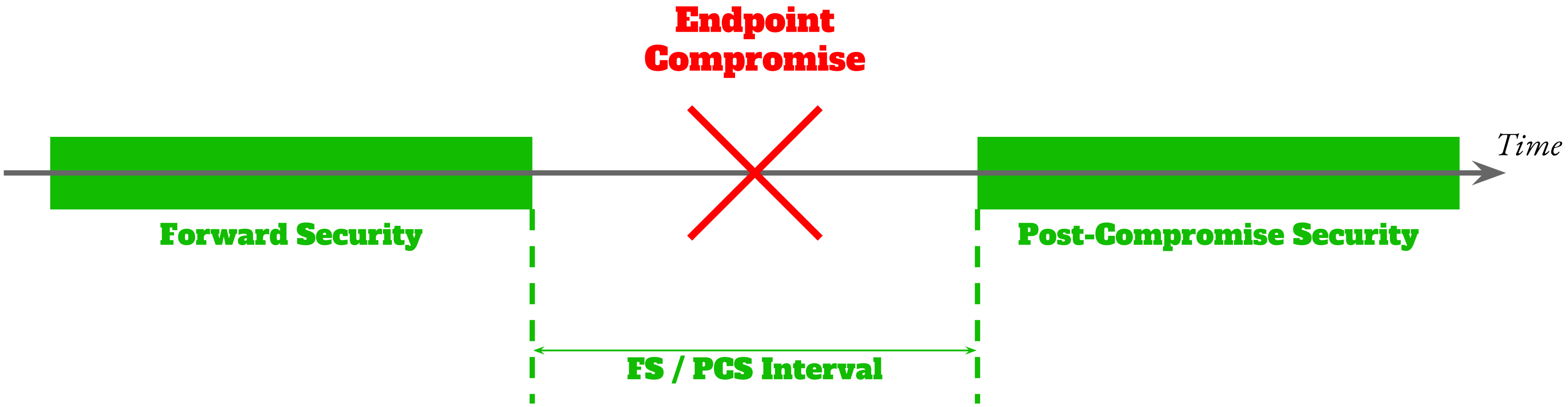
Async - Support sessions where no two participants are online at the same time

Group Messaging - Support large, dynamic groups with efficient scaling

Security Protocol - Modern security properties

- Forward security
- Post-compromise security
- Membership authentication
- Non-goals: Full-time deniability, malleability

Establish intuition:
FS ~> DH
PCS ~> Keep around DH, rotate DH



Prior Art

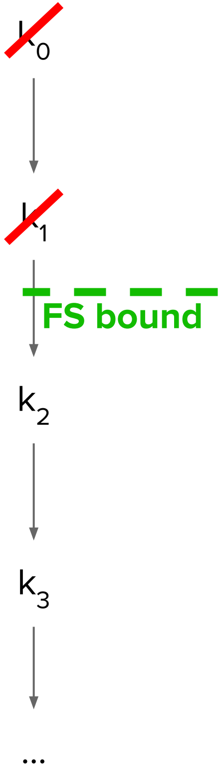
MIKEY	Similar options to S/MIME / OpenPGP
GDOI	Trusted (symmetric) key server
mpOTR, (n+1)sec	No PCS
S/MIME, OpenPGP	Linear scaling, difficult to achieve PCS
Client fanout Signal, Proteus, iMessage, et al.	Linear scaling, but **good async / PCS properties**
Sender Keys FB Messenger, OMEMO, Olm, et al.	Linear scaling, but **good async / PCS properties**

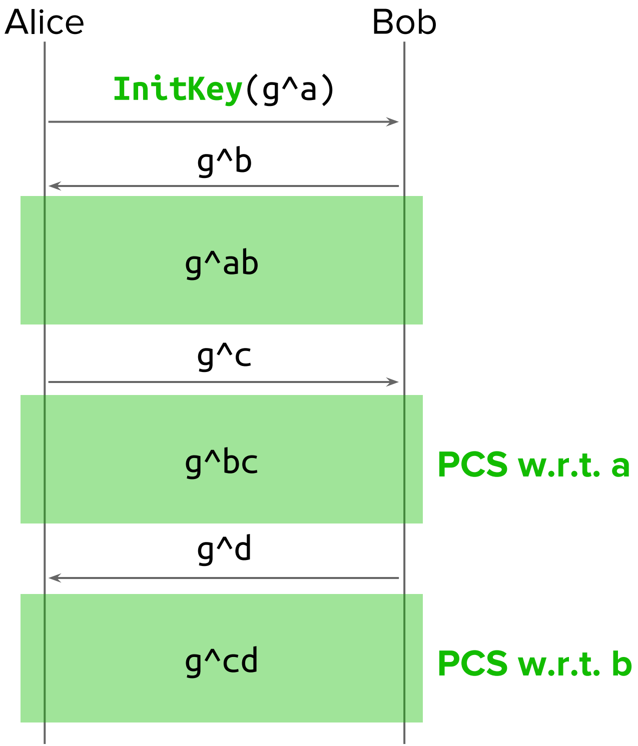
Key Ideas from Prior Art

"**InitKeys**" (or "prekeys") for async

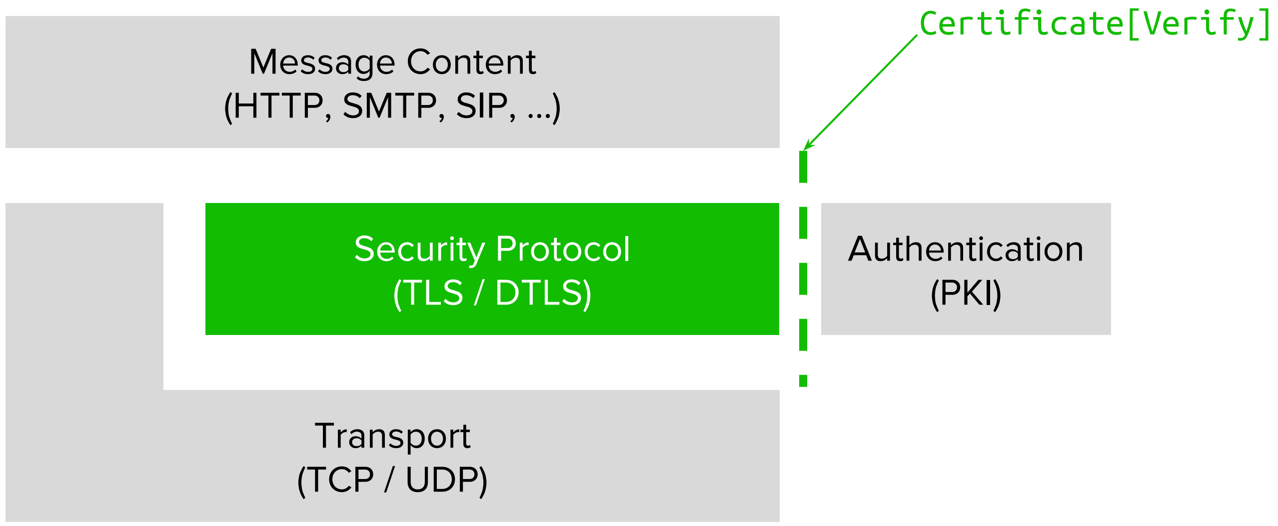
"**Hash ratchets**" forward secrecy

"**DH ratchets**" for PCS





Scope (with analogy to TLS)

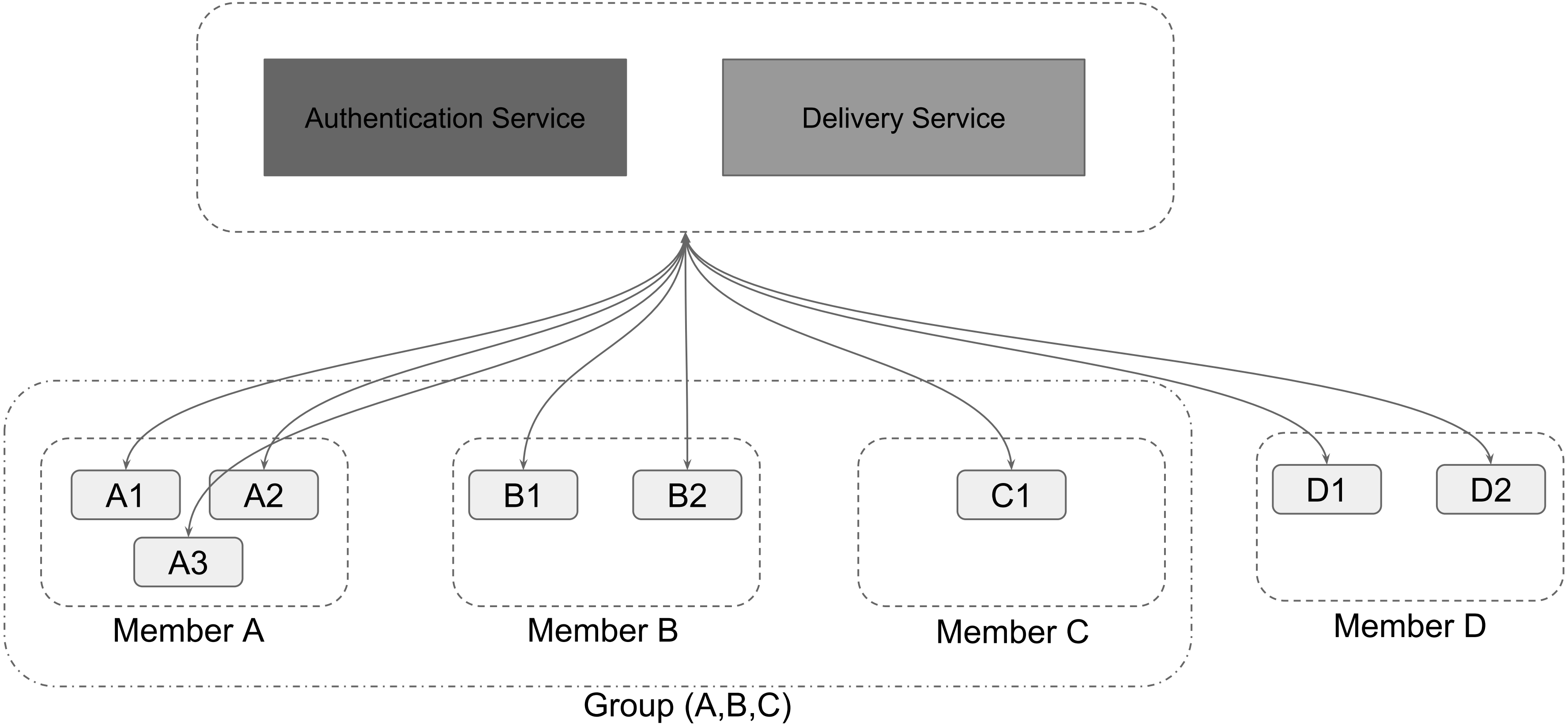


Architecture

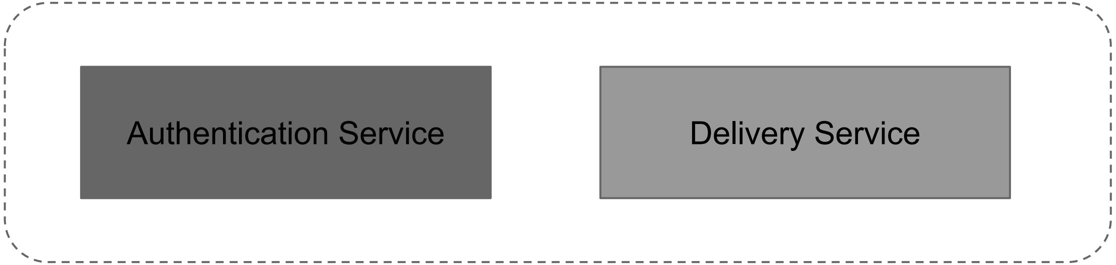
MLS Architecture

emadomara@google.com
@emad_omara

System Overview



System Overview

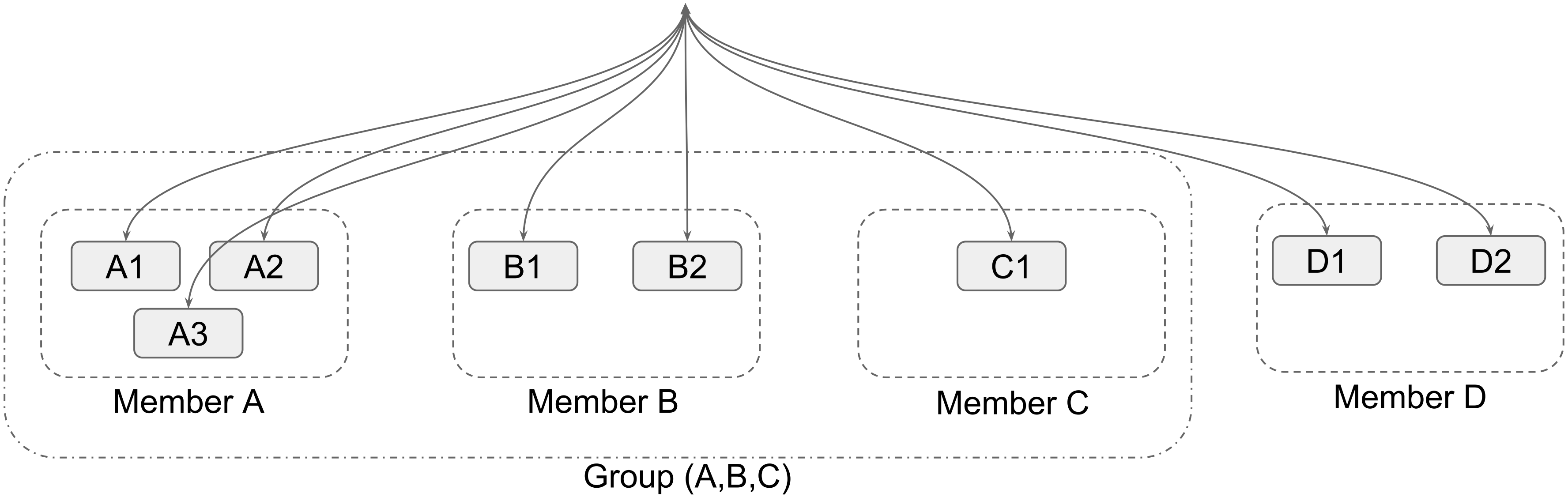


- Stores user ids to identity key mappings

- Distributes and delivers messages and attachments
- Stores initial key materials (initKeys)
- *Stores group membership

System Overview

- Register
- Send message
- Invite member
- Join group
- Add device
- Create group
- Receive message
- Remove member
- Leave group
- Remove device



Functional Requirements

- Scalable
  - Support group size up to 50,000 clients
- Asynchronous
  - All client operations can be performed without waiting for the other clients to be online
- Multiple devices
  - Devices are considered separate clients
  - Restoring history after joining is not allowed by the protocol, but Application can provide that.
- State recovery
  - Lost/Corrupted state must be recovered without affecting the group state.
- Metadata collection
  - AS/DS must only store data required for message delivery
- Federation
  - Multiple implementation should be able to interoperate
- Versioning
  - Support version negotiation

Security Requirements

- Message secrecy, integrity and authentication
  - Only current group member can read messages
  - Messages are only accepted if it was sent by a current group member
  - *Message padding to protect against traffic analysis
- Forward secrecy and post compromise security
- Group membership security
  - Consistent view of group members
  - Added clients can’t read messages sent before joining
  - Removed clients can’t read messages sent after leaving
- Attachments security
- Data origin authentication and *deniability

Security Considerations

- Delivery service compromise
  - Must not be able to read or inject messages
  - Modified, reordered or replayed messages must be detected by the clients
  - It can mount various DoS attacks.
- Authentication service compromise
  - Can return incorrect identities to the client
  - Can’t be defeated without transparency logging such as KT
- Client compromise
  - Can read and send messages to the group for a period of time
  - It shouldn’t be able to perform DoS attack.
  - Will be defeated once the compromised client updates their key material

(draft) Protocol

Messaging Layer Security

Draft Protocol

Jon Millican

jmillican@fb.com

Protocol Operations

- Group state at each point in time is an “asynchronous ratcheting tree”
- Each participant caches a view of the tree
- Protocol operations update the participants’ view of the tree
  - Group Creation
  - Group-initiated Add
  - User-initiated Add
  - Key Update
  - Remove

Asynchronous Ratcheting Tree

- (Cohn-Gordon et al., 2017, https://eprint.iacr.org/2017/666.pdf)
- Based on a Diffie-Hellman binary key tree.
- Updates to any leaf in logarithmic time.
- Asynchronous operation.
- Proofs of confidentiality of group keys in static groups.
- MLS defines some things that the original paper leaves out of scope:
  - More constraints on tree structure
  - Membership changes.
  - Race conditions.

DH output -> DH key pair

- `Derive-Key-Pair` maps random bit strings to DH key pairs
- Resulting private key known both original private key holders

```
  AB = Derive-Key-Pair(DH(A,B))
 /  \
A    B



e.g.:
  Derive-Key-Pair(X) = X25519-Priv(SHA-256(X))
```

DH Trees

```
             ABCDEFG
            /       \
           /         \
       ABCD           EFGH
       /  \           /  \
     AB    CD       EF    |
    / \   / \      / \    |
   A   B C   D    E   F   G
```

Part	Role
Root	Group Key
Direct Path	Update
Copath	Add
Frontier	Add

```
leaf + copath -> root
frontier = copath(next)
```

Group Evolution

```
               +-> Msg Secret                +-> Msg Secret                +-> Msg Secret
               |                             |                             |
... --> KDF -+-> Init Secret --> KDF -+-> Init Secret --> KDF -+-> Init Secret -->
...
         ^                             ^                             ^
         |                             |                             |
       Update                        Update                        Update
       Secret                        Secret                        Secret
         ^                             ^                             ^
         |                             |                             |
        Tree                          Tree                          Tree
        Root                          Root                          Root
```

Operation 0: Create group

- Can be created directly.
- Can be created by starting with an one-member group, then doing add operations.
- Current draft does the latter, so there’s no protocol for creation.
- ART paper specifies the former, but we don’t use in the draft yet.

```
              ABCDEFG
             /       \
            /         \
        ABCD           EFGH
       /    \         /    \
     AB      CD     EF      |
    /  \    /  \   /  \     |
   A    B  C    D E    F    G
```

Operation 1: Group-Initiated Add

```
struct {
    UserInitKey init_key;
} GroupAdd;

// Pre-published UserInitKey for
// asynchronicity

// NB: Add Key has implications
// for removals; “double join”
```

```
             ABCD
            /    \
        AB          CD
       /  \        /  \
      A    B    C      D
                      / \
             Add Key     Init
```

Operation 2: User-Initiated Add

```
struct {
    DHPublicKey add_path<1..2^16-1>;
} UserAdd;

// Pre-published frontier in
// GroupInitKey for asynchronicity
```

```
        ABCD
       /    \
     AB      CD
    /  \    /  \
   A    B  C    D
```

Operation 3: Key Update (for PCS)

```
struct {
    DHPublicKey
ratchetPath<1..2^16-1>;
} Update;

// This approach to confidentiality
// is proved in [ART]
```

```
          ABCD
        /      \
     AB          CD
   /    \       /   \
  A      B    C      D
```

Operation 4: Remove

```
struct {
    uint32 deleted;
    DHPublicKey path<1..2^16-1>;
} Delete;

// To lock out, update to a key the
// deleted node doesn’t know

// “Double join” issues similar to
// GroupAdd
```

```
        AXCD
       /    \
    AX        CD
   /  \      /  \
  A    X    C    D
```

Open Issues

- Tuning up, proving FS and PCS properties of the operations
  - … especially Add, Remove
- Logistical details, especially around Remove
- Message sequencing
- Message protection, transcript integrity
- Authentication
  - Current draft has a very basic scheme, needs elaboration
  - Deniable authentication?
- *Attachments

Summary

- Group keys derived from an Asynchronous Ratcheting Tree
- Group operations update the tree
  - Creation
  - Group-Initiated Add
  - User-Initiated Add
  - Update
  - Remove
- Several open issues to address in the WG

Formal Analysis

the ART of analysing MLS

Katriel Cohn-Gordon
University of Oxford

people involved

- Karthik (HACL*, miTLS, F*)
- Benjamin (F*, NSS)
- Cas (Tamarin, TLS 1.3)
- Katriel (Signal, PCS)
- ...

similar projects

- F*
- miTLS
- TLS 1.3: the swamp
- 5G-ENSUUUURE
- ...

1

On Ends-to-Ends Encryption

Asynchronous Group Messaging with Strong Security Guarantees

Abstract—**In the past few years secure messaging has become mainstream, with over a billion active users of end-to-end encryption protocols through apps such as WhatsApp, Signal, Facebook Messenger, Google Allo, Wire and many more. While these users' two-party communications now enjoy very strong security guarantees, it turns out that many of these apps provide, without notifying the users, a weaker property for *group* messaging: an adversary who compromises a single group member can intercept communications indefinitely.**

One reason for this discrepancy in security guarantees, despite the large body of work on group key agreement, is that most existing protocol designs are fundamentally *synchronous*, and thus cannot be used in the asynchronous world of mobile communications. In this paper we show that this is not necessary, presenting a design for a tree-based group key exchange protocol in which no two parties ever need to be online at the same time, which we call *Asynchronous Ratcheting Tree (ART)*. ART achieves strong security guarantees, in particular including post-compromise security.

We give a computational security proof for ART's core design as well as a proof-of-concept implementation, showing that ART scales efficiently even to large groups. Our results show that strong security guarantees for group messaging are achievable even in the modern, asynchronous setting, without resorting to using inefficient point-to-point communications for large groups. By building on standard and well-studied constructions, our hope is that many existing solutions can be applied while still respecting the practical constraints of mobile devices.

I. Introduction

The level of security offered by secure messaging systems has improved substantially over recent years; for example, WhatsApp now provides end-to-end encryption for its billion active users, based on Open Whisper Systems' Signal Protocol [34, 45], and the Guardian publishes Signal contact details for its investigative journalism teams [21]. An important constraint of modern messaging systems, compared to related protocols such as those used for key exchange, is that they must allow for *asynchronous communication*: Alice must be able to send a message to Bob even if Bob is currently offline. Typically, the encrypted message is temporarily stored on a (possibly untrusted) server, to be delivered to Bob once he comes online again.

Asynchronicity means that standard solutions to achieve perfect forward secrecy (PFS), such as a Diffie–Hellman (DH) key exchange, do not apply directly. This has driven the development of novel techniques to achieve PFS without interaction, e.g., using sets of "prekeys" [33] that Bob uploads to a server, essentially serving as precomputed DH keys, or by using puncturable encryption [20].

Moreover, some modern messaging protocols offer a property called Post-Compromise Security (PCS) [12], sometimes referred to as "future secrecy" or "self-healing". For PCS, even after Alice's device is entirely compromised by an adversary, she may later be able to establish secure communications with others after a single unintercepted exchange. PCS limits the scope of a compromise, forcing an adversary to act as a permanent active man-in-the-middle if they wish to exploit knowledge of a long-term key. Thus far, PCS-style properties have only been proven for point-to-point protocols [11], and they are only achievable by stateful protocols [12].

In practice however, point-to-point communication does not suffice for real-world messaging applications, in which group and multi-device messaging are often important features. In theory, it is easy to solve this: Alice uses the point-to-point protocol with each of her communication partners. However, as group sizes become larger, this leads to inefficient systems in which the bandwidth and computational cost for sending a message grows linearly with the group size (as each recipient gets their own, differently encrypted, copy of the message). In many real-world scenarios, this inefficiency can be problematic, especially in areas with restricted bandwidth or high data costs (e.g., 2G networks in the developing world). The 2015 State of Connectivity report by internet.org [22] lists affordability of mobile data as one of the four major barriers to global connectivity, with a developing-world average monthly data use of just 255 MB/device.

Instead of using a point-to-point protocol with each group member, a theoretical alternative is to use a group protocol [7, 8, 14, 24, 25, 26, 27, 28, 31, 37, 38]. These typically use tree structures based on DH keys to combine the participants' individual keys into a group key. This reduces both the computational effort and bandwidth required to send a message, as the sender sends only one copy of each message encrypted under the group key. However, such protocols are in general not asynchronous, and do not consider PCS—they do not make any guarantees after the adversary completely compromises a participant.

Synchronicity of existing group protocols, among other considerations, means that modern messaging protocols which provide PCS for two-party communications generally drop this guarantee for their group messaging implementations without notifying the users. For example, WhatsApp, Facebook Messenger and the Signal app have mechanisms to achieve PCS for two-party communications, but for conversations containing three or more devices they use a simpler key-transport mechanism ("sender keys") which does not achieve PCS [16, 45]. Indeed, in all three systems, an adversary who fully compromises a single group member can indefinitely and passively read future communications in that group (though

analysis status

- properties fairly well understood
  - secrecy, authentication
  - agreement on members
  - post-compromise security
- ART construction is new but has some early formal analysis
  - *On Ends-to-Ends Encryption* https://ia.cr/2017/666
  - Katriel, Cas, Luke, Kevin, Jon
- full group protocol: more to do!

a bit more on the formal analysis

Theorem VI.1. *Let n_P, n_S and n_T denote bounds on the number of parties, sessions and stages in the security experiment respectively. Under the decisional DH assumption, where ι is instantiated as a random oracle, the success probability of any ppt adversary against the key indistinguishability game of our protocol is bounded above by*

$$\frac{1}{2} + \frac{\binom{\mathsf{n}_\mathsf{P}\mathsf{n}_\mathsf{S}\mathsf{n}_\mathsf{T}}{2}}{q} + \gamma(\mathsf{n}_\mathsf{P}\mathsf{n}_\mathsf{S}{\mathsf{n}_\mathsf{T}}^2)^\gamma\left(\epsilon_{DDH} + {}^1/_q\right) + \mathsf{negl}(\lambda)$$

where ϵ_{DDH} bounds the advantage of a PPT adversary against the decisional DH game.

going forward

- precise definitions of the properties we would like
  - interactions with “practical” constraints such as recovery from lost devices
  - general enough to cover different use cases

- proofs for the whole system
  - authentication
  - malicious insiders
  - adding and removing people

- verified implementations in F^*?

tl;dr

no proofs yet, but early work on ART and we're still going :)

Charter Text

Several Internet applications have a need for group key establishment and message protection protocols with the following properties:

- Asynchronicity - Keys can be established without any two participants being online at the same time
- Forward secrecy - Full compromise of a node at a point in time does not reveal past group keys
- Post-compromise security - Full compromise of a node at a point in time does not reveal future group keys
- Membership Authentication - Each participant can verify the set of members in the group
- Message Authentication - Each message has an authenticated sender
- Scalability - Resource requirements that have good scaling in the size of the group (preferably sub-linear)

Several widely-deployed applications have developed their own protocols to meet these needs. While these protocols are similar, no two are close enough to interoperate. As a result, each application vendor has had to maintain their own protocol stack and independently build trust in the quality of the protocol. The primary goal of this working group is to develop a standard messaging security protocol so that applications can share code, and so that there can be shared validation of the protocol (as there has been with TLS 1.3).

It is not a goal of this group to enable interoperability between messaging applications beyond the key establishment, authentication, and confidentiality services. Full interoperability would require alignment at many different layers beyond security, e.g., standard message transport and application semantics. The focus of this work is to develop a messaging security layer that different applications can adapt to their own needs.

In developing this protocol, we will draw on lessons learned from several prior message-oriented security protocols, in addition to the proprietary messaging security protocols deployed within existing applications:
- S/MIME | OpenPGP | Off the Record | Signal

The intent of this working group is to follow the pattern of TLS 1.3, with specification, implementation, and verification proceeding in parallel. By the time we arrive at RFC, we hope to have several interoperable implementations as well as a thorough security analysis.

The specifications developed by this working group will be based on pre-standardization implementation and deployment experience, and generalizing the design described in:

- draft-omara-mls-architecture
- draft-barnes-mls-protocol

Note that consensus is required both for changes to the current protocol mechanisms and retention of current mechanisms. In particular, because something is in the initial document set does not imply that there is consensus around the feature or around how it is specified.

Milestones:

May 2018 Initial working group documents for architecture and key management

Sept 2018 Initial working group document adopted for message protection

Jan 2019 Submit architecture document to IESG as Informational

Jun 2019 Submit key management protocol to IESG as Proposed Standard

Sept 2019 Submit message protection protocol to IESG as Proposed Standard

Scoping Questions

Should the IETF do the work?

Does the scope sound reasonable?

Are the boundaries presented suitable for a security analysis?

Do we agree that the application layer interface is the correct place to enable visibility requirements should they exist?

Do the documents presented represent a good starting point?

Is this proposal flexible enough for the common use cases of secure messaging applications?

BOF Questions

Successful BOF Questions

➔➔➔➔➔➔➔➔

Is the problem sufficiently understood?

Is the problem tractable?

Is this the right place to address “the problem”?

Who is willing to author specs?

Who is willing to review specs?

Backup