WhatsUpp with Sender Keys?

Analysis, Improvements and Security Proofs

David Balbás[1,2], **Daniel Collins**[3], **Phillip Gajland**[4,5]

27th May 2024

[1]IMDEA Software Institute, Madrid, Spain
[2]Universidad Politécnica de Madrid, Spain
[3]EPFL, Lausanne, Switzerland
[4]Max Planck Institute for Security and Privacy, Bochum, Germany
[5]Ruhr University Bochum, Germany

Group Messaging

- **Messaging protocols** are used by billions daily. Many apps claim **security** + **end-to-end encryption**.

Group Messaging

- **Messaging protocols** are used by billions daily. Many apps claim **security** + **end-to-end encryption**.
- Formal protocol analysis is crucial.

Group Messaging

- **Messaging protocols** are used by billions daily. Many apps claim **security** + **end-to-end encryption**.
- Formal protocol analysis is crucial.
- **MLS**: Lots of theoretical analysis. *Secure, efficient, complex.*

Group Messaging

- **Messaging protocols** are used by billions daily. Many apps claim **security** + **end-to-end encryption**.

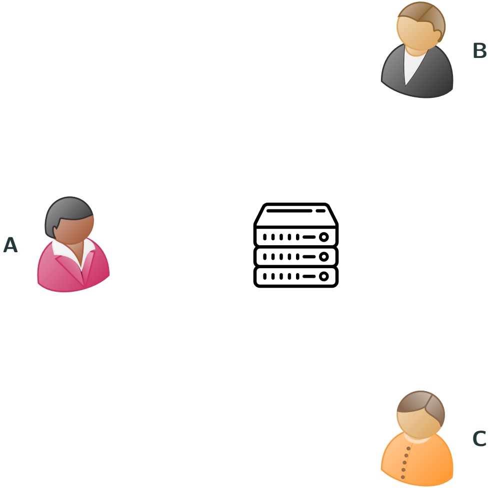
- Formal protocol analysis is crucial.

- **MLS**: Lots of theoretical analysis. *Secure, efficient, complex.*

- **Sender Keys**: WhatsApp, Signal. *No formal analysis so far.*

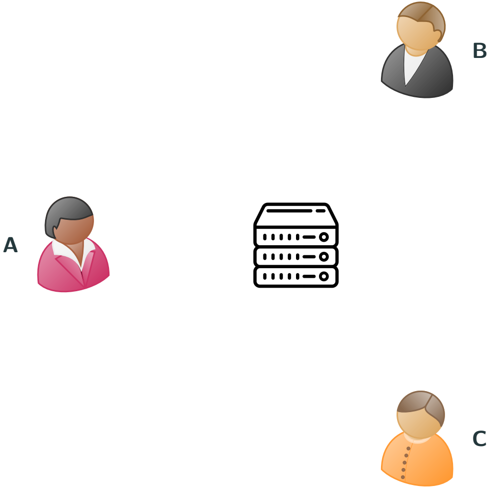
Sender Keys

- Sender Keys is a **simple, efficient** group messaging protocol used in WhatsApp, Signal, Matrix...



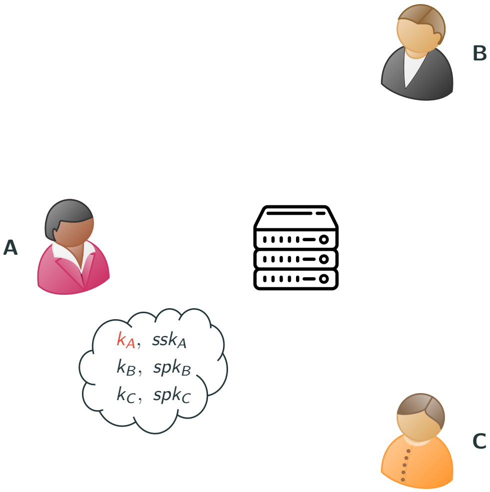
Sender Keys

- Sender Keys is a **simple, efficient** group messaging protocol used in WhatsApp, Signal, Matrix...
- Parties use their own symmetric key k_{ID} to encrypt. **No group key.**



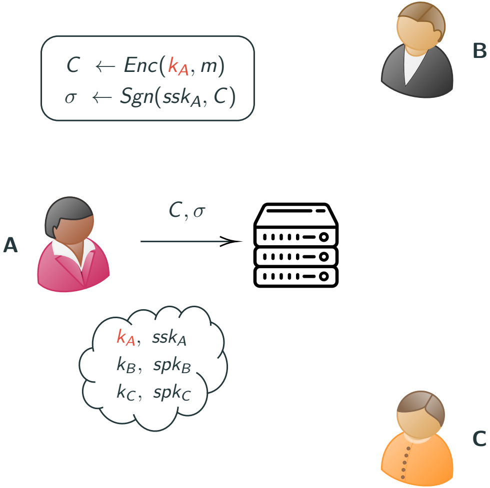
Sender Keys

- Sender Keys is a **simple, efficient** group messaging protocol used in WhatsApp, Signal, Matrix...
- Parties use their own symmetric key k_{ID} to encrypt. **No group key.**



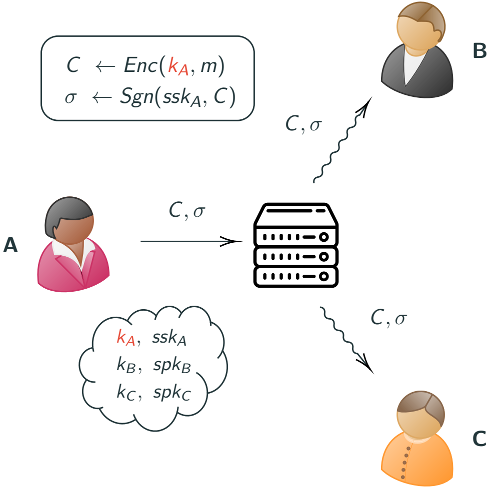
Sender Keys

- Sender Keys is a **simple, efficient** group messaging protocol used in WhatsApp, Signal, Matrix...
- Parties use their own symmetric key k_{ID} to encrypt. **No group key.**



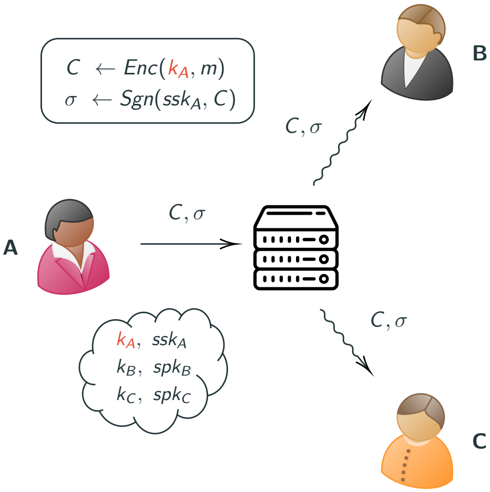
Sender Keys

- Sender Keys is a **simple, efficient** group messaging protocol used in WhatsApp, Signal, Matrix...
- Parties use their own symmetric key k_{ID} to encrypt. **No group key.**



Sender Keys

- Sender Keys is a **simple, efficient** group messaging protocol used in WhatsApp, Signal, Matrix...
- Parties use their own symmetric key k_{ID} to encrypt. **No group key.**
- Parties use *two-party messaging* to share fresh key material.



What is Secure Messaging?

- Correct, authentic, confidential, and asynchronous messaging.

What is Secure Messaging?

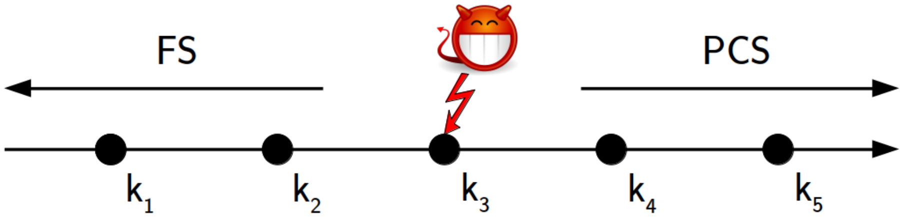
- Correct, authentic, confidential, and asynchronous messaging.
- **Secure membership.**

What is Secure Messaging?

- Correct, authentic, confidential, and asynchronous messaging.
- **Secure membership.**
- **Forward Security (FS):** *past* messages secret after compromise.

What is Secure Messaging?

- Correct, authentic, confidential, and asynchronous messaging.
- **Secure membership.**
- **Forward Security (FS):** *past* messages secret after compromise.
- **Post-Compromise Security (PCS):** *future* messages secret a key refresh.



So, WhatsUpp with Sender Keys?

So, WhatsUpp with Sender Keys?

Can we **formalise** Sender Keys in a *meaningful security model*, considering the aforementioned requirements?

So, WhatsUpp with Sender Keys?

Can we **formalise** Sender Keys in a *meaningful security model*, considering the aforementioned requirements?

What are its main **deficiencies**, and how can we address them *efficiently*?

Our Work

- **Formalization** of Sender Keys in a novel framework.

Our Work

- **Formalization** of Sender Keys in a novel framework.
- **Security model** capturing interaction between group messages and two-party channels.

Our Work

- **Formalization** of Sender Keys in a novel framework.
- **Security model** capturing interaction between group messages and two-party channels.
- **Proof of security** with some restrictions. Identified shortcomings.

Our Work

- **Formalization** of Sender Keys in a novel framework.
- **Security model** capturing interaction between group messages and two-party channels.
- **Proof of security** with some restrictions. Identified shortcomings.
- **Improvements** in *Sender Keys+*: better efficiency and security.

Our Work

- **Formalization** of Sender Keys in a novel framework.
- **Security model** capturing interaction between group messages and two-party channels.
- **Proof of security** with some restrictions. Identified shortcomings.
- **Improvements** in *Sender Keys+*: better efficiency and security.

Concurrent work [Albrecht, Dowling, Jones, S&P 2024] formalizes Matrix, similar conclusions.

Protocol and Syntax

Syntax

A **Group Messenger (GM)** includes:

Syntax

A **Group Messenger (GM)** includes:

- $C \xleftarrow{\$} \mathit{Send}(\mathrm{m}, \gamma)$

Syntax

A **Group Messenger (GM)** includes:

- $C \xleftarrow{\$} \mathit{Send}(\mathrm{m}, \gamma)$
- $(\mathrm{m}, \mathit{ID}^*, e, i) \xleftarrow{\$} \mathit{Recv}(C, \gamma)$

Syntax

A **Group Messenger (GM)** includes:

- $C \xleftarrow{\$} \mathit{Send}(\mathrm{m}, \gamma)$
- $(\mathrm{m}, \mathit{ID}^*, e, i) \xleftarrow{\$} \mathit{Recv}(C, \gamma)$

- $T \xleftarrow{\$} \mathit{Exec}(\mathtt{cmd}, \boldsymbol{\mathit{IDs}}, \gamma)$, $\mathtt{cmd} \in \{\mathtt{crt}, \mathtt{add}, \mathtt{rem}, \mathtt{upd}\}$

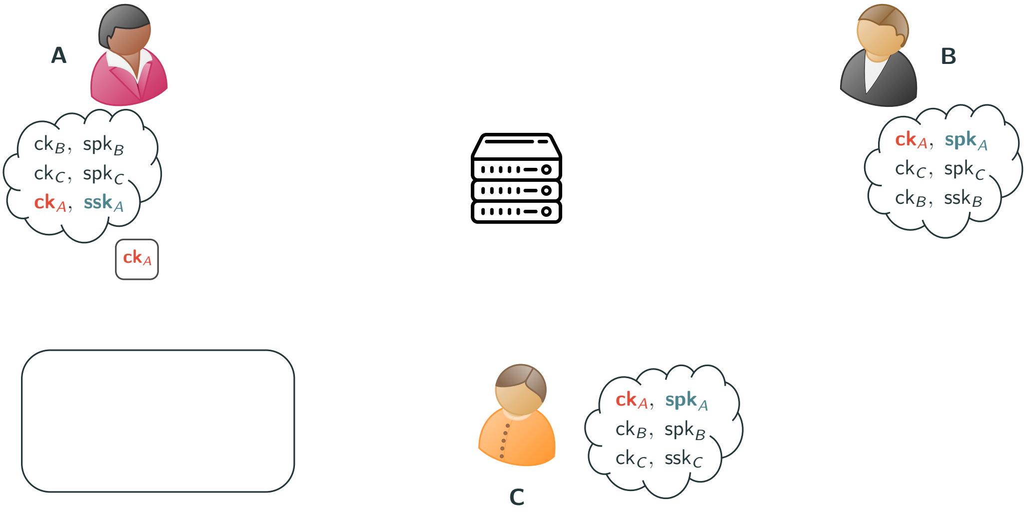
Syntax

A **Group Messenger (GM)** includes:

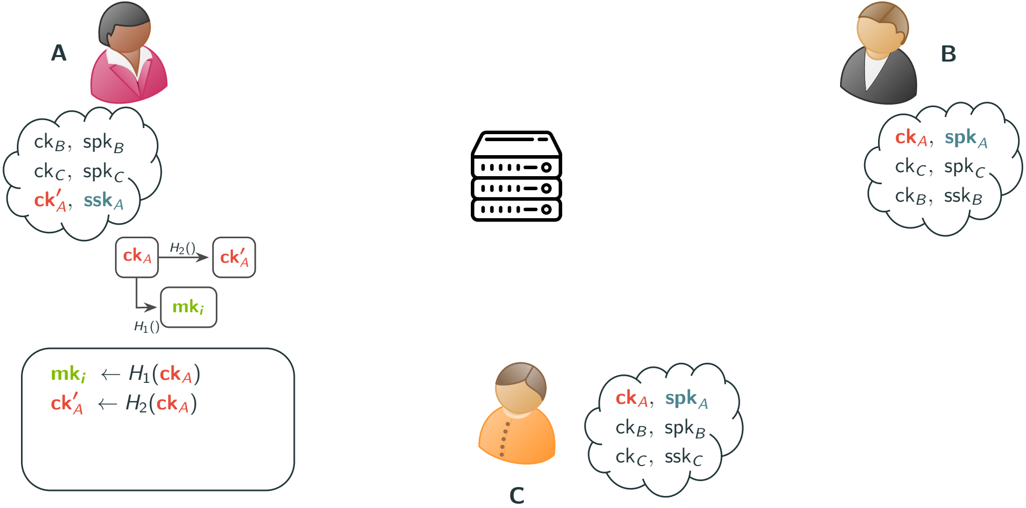
- $C \xleftarrow{\$} \mathit{Send}(\mathrm{m}, \gamma)$
- $(\mathrm{m}, \mathit{ID}^*, e, i) \xleftarrow{\$} \mathit{Recv}(C, \gamma)$

- $T \xleftarrow{\$} \mathit{Exec}(\mathtt{cmd}, \boldsymbol{\mathit{IDs}}, \gamma)$, $\mathtt{cmd} \in \{\mathtt{crt}, \mathtt{add}, \mathtt{rem}, \mathtt{upd}\}$
- $b \xleftarrow{\$} \mathit{Proc}(T, \gamma)$

Sender Keys: Send & Recv

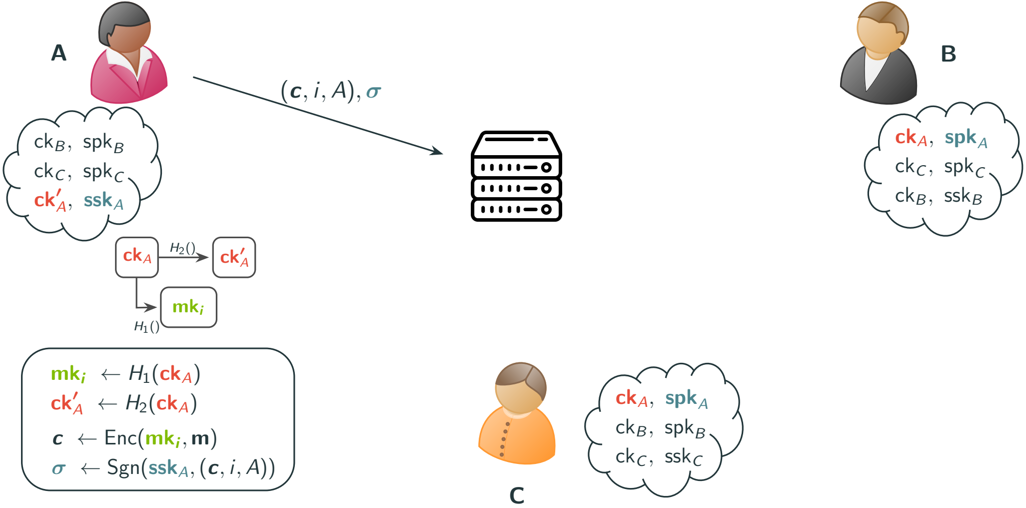


Sender Keys: Send & Recv

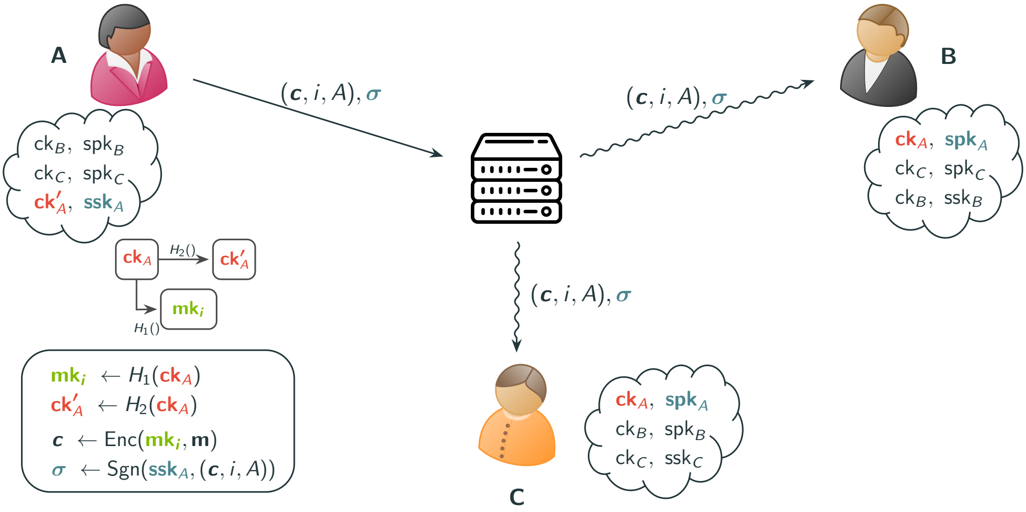


$$mk_i \leftarrow H_1(ck_A)$$
$$ck'_A \leftarrow H_2(ck_A)$$

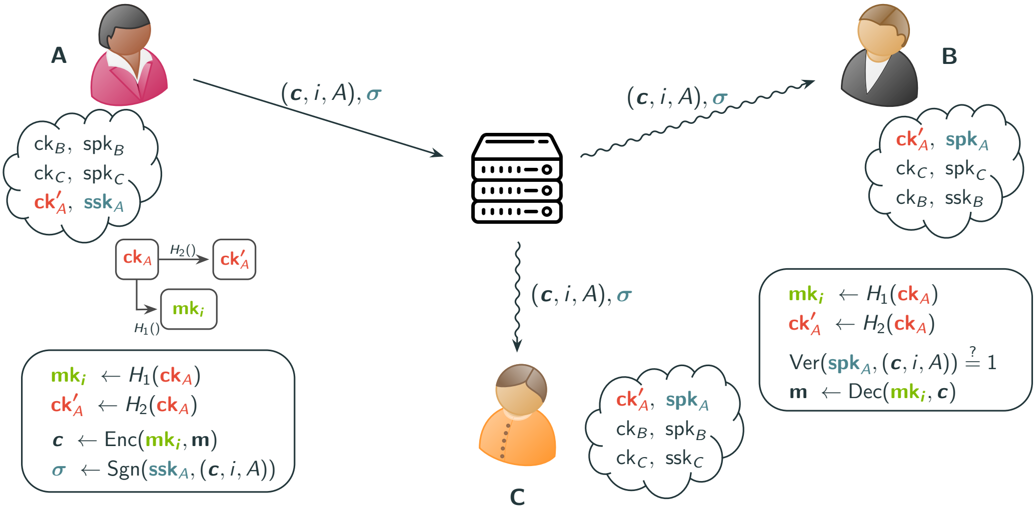
Sender Keys: Send & Recv



Sender Keys: Send & Recv

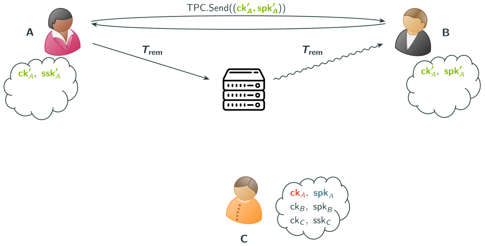


Sender Keys: Send & Recv



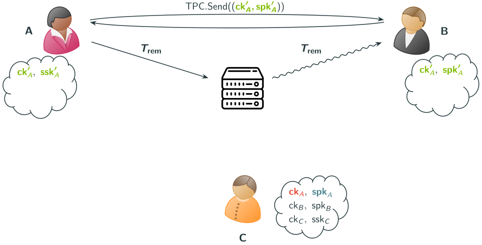
Two-Party Channels

- **If C leaves** (or someone updates):



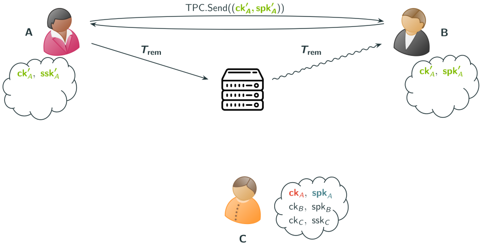
Two-Party Channels

- **If C leaves** (or someone updates):
  - A, B process removal & erase keys.
  - Fresh keys sent over secure 2PC.



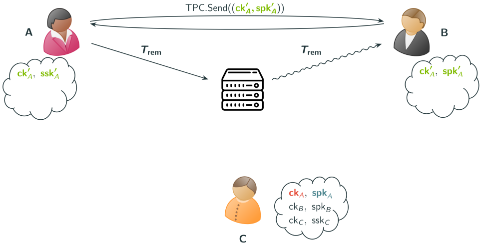
Two-Party Channels

- **If C leaves** (or someone updates):
  - A, B process removal & erase keys.
  - Fresh keys sent over secure 2PC.
- In reality, *New keys sent encrypted...* under **Double Ratchet keys!** [MP16]
- A compromise also affects 2PC keys.



Two-Party Channels

- **If C leaves** (or someone updates):
  - A, B process removal & erase keys.
  - Fresh keys sent over secure 2PC.
- In reality, *New keys sent encrypted…* under **Double Ratchet keys!** [MP16]
- A compromise also affects 2PC keys.

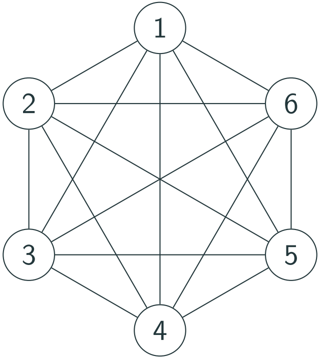


Modelling 2PC

We model *two-party channels as a primitive* 2PC.

Two-Party Channels

Two-party channels only refresh (i.e. achieve PCS) if users interact.



However, some two-party chats are often stale...

Proving Security

Security Model

We introduce a *message indistinguishability* security game $\textbf{M-IND}_{\textbf{C}}$.

Security Model

We introduce a *message indistinguishability* security game $\mathbf{M\text{-}IND_C}$.

- Adaptive $\mathcal{A}$ can *forge and inject messages*.

Security Model

We introduce a *message indistinguishability* security game $\mathbf{M\text{-}IND_C}$.

- Adaptive $\mathcal{A}$ can *forge and inject messages*.
- Users can be *exposed* at any time (capturing FS, PCS).

Security Model

We introduce a *message indistinguishability* security game $\mathbf{M\text{-}IND_C}$.

- Adaptive $\mathcal{A}$ can *forge and inject messages*.
- Users can be *exposed* at any time (capturing FS, PCS).
- **$\mathcal{A}$ wins if:**

Security Model

We introduce a *message indistinguishability* security game $\mathbf{M\text{-}IND_C}$.

- Adaptive $\mathcal{A}$ can *forge and inject messages*.
- Users can be *exposed* at any time (capturing FS, PCS).
- **$\mathcal{A}$ wins if:**
  - breaks semantic security, or
  - forges a message
  - in a *clean*/safe execution under C.

Security Model

We introduce a *message indistinguishability* security game $\mathbf{M\text{-}IND_C}$.

- Adaptive $\mathcal{A}$ can *forge and inject messages*.
- Users can be *exposed* at any time (capturing FS, PCS).
- **$\mathcal{A}$ wins if:**
  - breaks semantic security, or
  - forges a message
  - in a *clean*/safe execution under C.
- Clean execution: no trivial attacks.

Security Model

We introduce a *message indistinguishability* security game $\mathbf{M\text{-}IND_C}$.

- Adaptive $\mathcal{A}$ can *forge and inject messages*.
- Users can be *exposed* at any time (capturing FS, PCS).
- **$\mathcal{A}$ wins if:**
  - breaks semantic security, or
  - forges a message
  - in a *clean*/safe execution under C.
- Clean execution: no trivial attacks.

Oracles:

- $\texttt{Create}(ID, \boldsymbol{IDs})$
- $\texttt{Challenge}(ID, \mathrm{m}_0, \mathrm{m}_1)$
- $\texttt{Send}(ID, \mathrm{m})$
- $\texttt{Receive}(ID, C)$
- $\texttt{Add}/\texttt{Remove}(ID, ID')$
- $\texttt{Update}(ID)$
- $\texttt{Deliver}(ID, T)$
- $\texttt{Expose}(ID)$

Main theorem

Security of Sender Keys (informal)

Assume

- SymEnc is a IND-CPA symmetric encryption scheme.
- Sig is a SUF-CMA signature scheme.
- H is a PRG.
- 2PC is a 2PC-IND_{Δ} two-party channels scheme for PCS bound $\Delta > 0$.

Then Sender Keys is $\text{M-IND}_{C(\Delta)}$ secure in our model w.r.t. a weak predicate C.

Main theorem

Security of Sender Keys (informal)

Assume

- SymEnc is a IND-CPA symmetric encryption scheme.
- Sig is a SUF-CMA signature scheme.
- H is a PRG.
- 2PC is a 2PC-IND_{Δ} two-party channels scheme for PCS bound $\Delta > 0$.

Then Sender Keys is $\mathsf{M\text{-}IND}_{\mathsf{C}(\Delta)}$ secure in our model w.r.t. a weak predicate C.

Conclusion: The core of the protocol has *no fundamental flaws*. But it still presents some drawbacks.

Limitations

- **Slow healing** due to two-party channels, inefficient updates.

Limitations

- **Slow healing** due to two-party channels, inefficient updates.
- **Total ordering** of control messages required.

Limitations

- **Slow healing** due to two-party channels, inefficient updates.
- **Total ordering** of control messages required.
- No **authentication** for control messages.

Limitations

- **Slow healing** due to two-party channels, inefficient updates.
- **Total ordering** of control messages required.
- No **authentication** for control messages.
- **Weak forward security** for authentication.

Sender Keys+

- **Slow healing** due to two-party channels, *~~in~~efficient updates.*
- **Total ordering** of control messages required.
- ~~No~~ **authentication** *for control messages*.
- **~~Weak~~ forward security** *for authentication*.

We propose and formalize **Sender Keys+** *as a practical, improved alternative*!

Sub-Optimal Forward Security

Let $\boldsymbol{G} = \{ID_1, ID_2\}$. Then $\mathcal{A}$ queries:

Sub-Optimal Forward Security

Let $\boldsymbol{G} = \{ID_1, ID_2\}$. Then $\mathcal{A}$ queries:

- $q_1 = \texttt{Send}(ID_1, \mathrm{m})$ generates C encrypted under mk and signed under ssk_1.

Sub-Optimal Forward Security

Let $\boldsymbol{G} = \{ID_1, ID_2\}$. Then $\mathcal{A}$ queries:

- $q_1 = \texttt{Send}(ID_1, \mathrm{m})$ generates C encrypted under mk and signed under ssk_1.
- $q_2 = \texttt{Expose}(ID_1)$, where $\mathcal{A}$ obtains ssk_1, but not mk.

Sub-Optimal Forward Security

Let $\boldsymbol{G} = \{ID_1, ID_2\}$. Then $\mathcal{A}$ queries:

- $q_1 = \texttt{Send}(ID_1, \mathrm{m})$ generates C encrypted under mk and signed under ssk_1.
- $q_2 = \texttt{Expose}(ID_1)$, where $\mathcal{A}$ obtains ssk_1, but not mk.
- $\mathcal{A}$ replaces the symmetric ciphertext in C and signs under ssk_1 to create C'.

Sub-Optimal Forward Security

Let $\boldsymbol{G} = \{ID_1, ID_2\}$. Then $\mathcal{A}$ queries:

- $q_1 = \texttt{Send}(ID_1, \mathrm{m})$ generates C encrypted under mk and signed under ssk_1.
- $q_2 = \texttt{Expose}(ID_1)$, where $\mathcal{A}$ obtains ssk_1, but not mk.
- $\mathcal{A}$ replaces the symmetric ciphertext in C and signs under ssk_1 to create C'.
- $q_3 = \texttt{Receive}(ID_2, ID_1, C')$, which is a successful injection.

Sub-Optimal Forward Security

Let $\boldsymbol{G} = \{ID_1, ID_2\}$. Then $\mathcal{A}$ queries:

- $q_1 = \texttt{Send}(ID_1, \mathrm{m})$ generates C encrypted under mk and signed under ssk_1.
- $q_2 = \texttt{Expose}(ID_1)$, where $\mathcal{A}$ obtains ssk_1, but not mk.
- $\mathcal{A}$ replaces the symmetric ciphertext in C and signs under ssk_1 to create C'.
- $q_3 = \texttt{Receive}(ID_2, ID_1, C')$, which is a successful injection.

q_3 attempts to inject a message with keys from *before* exposure $\implies$ should be allowed.
Can occur naturally e.g. if ID_2 is offline when m is first sent.

Sub-Optimal Forward Security

Let $\boldsymbol{G} = \{ID_1, ID_2\}$. Then $\mathcal{A}$ queries:

- $q_1 = \texttt{Send}(ID_1, \mathrm{m})$ generates C encrypted under mk and signed under ssk_1.
- $q_2 = \texttt{Expose}(ID_1)$, where $\mathcal{A}$ obtains ssk_1, but not mk.
- $\mathcal{A}$ replaces the symmetric ciphertext in C and signs under ssk_1 to create C'.
- $q_3 = \texttt{Receive}(ID_2, ID_1, C')$, which is a successful injection.

q_3 attempts to inject a message with keys from *before* exposure $\implies$ should be allowed.
Can occur naturally e.g. if ID_2 is offline when m is first sent.

Can be prevented with a MAC.

Unsigned Control Messages

- In WhatsApp, **control messages** are *not signed, so a network adversary can forge them without exposing any party!*

Unsigned Control Messages

- In WhatsApp, **control messages** are *not signed, so a network adversary can forge them without exposing any party!*
- Server can add/remove parties on behalf of other users:
  - Burgle into the group attack [RMS18].

Unsigned Control Messages

- In WhatsApp, **control messages** are *not signed, so a network adversary can forge them without exposing any party!*
- Server can add/remove parties on behalf of other users:
  - Burgle into the group attack [RMS18].
- Signal provides more protection but less than if signatures were used.

Unsigned Control Messages

- In WhatsApp, **control messages** are *not signed, so a network adversary can forge them without exposing any party!*
- Server can add/remove parties on behalf of other users:
    - Burgle into the group attack [RMS18].
- Signal provides more protection but less than if signatures were used.
- **Solution:** *sign control messages*!

Sender Keys in Practice

- Contrary to some folklore, Signal uses Sender Keys whenever possible!

Sender Keys in Practice

- Contrary to some folklore, Signal uses Sender Keys whenever possible!
- WhatsApp (resp. Signal) supports 1024 (resp. 256) member groups.

Sender Keys in Practice

- Contrary to some folklore, Signal uses Sender Keys whenever possible!
- WhatsApp (resp. Signal) supports 1024 (resp. 256) member groups.
- Signal protects privacy more than WhatsApp (sealed sender, private groups...).

Sender Keys in Practice

- Contrary to some folklore, Signal uses Sender Keys whenever possible!
- WhatsApp (resp. Signal) supports 1024 (resp. 256) member groups.
- Signal protects privacy more than WhatsApp (sealed sender, private groups...).
- Matrix uses Sender Keys but does not ratchet symmetric keys.

Final Remarks

Takeaways

- Sender Keys is used by WhatsApp and Signal.

Takeaways

- Sender Keys is used by WhatsApp and Signal.
- Sender Keys is *provably secure* but in a *weak model*.

Takeaways

- Sender Keys is used by WhatsApp and Signal.
- Sender Keys is *provably secure* but in a *weak model*.
- *Sender Keys+*: improved security and efficiency.

Takeaways

- Sender Keys is used by WhatsApp and Signal.
- Sender Keys is *provably secure* but in a *weak model*.
- *Sender Keys+*: improved security and efficiency.

Thank you!

ia.cr/2023/1385

danielpatcollins@gmail.com
phillip.gajland@mpi-sp.org