

WhatsUpp with Sender Keys?

Analysis, Improvements and Security Proofs

David Balbás^{1,2}, **Daniel Collins**³, **Phillip Gajland**^{4,5}

27th May 2024

¹IMDEA Software Institute, Madrid, Spain

²Universidad Politécnica de Madrid, Spain

³EPFL, Lausanne, Switzerland

⁴Max Planck Institute for Security and Privacy, Bochum, Germany

⁵Ruhr University Bochum, Germany

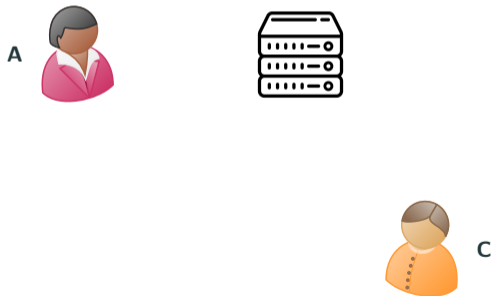
Group Messaging

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



Sender Keys

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



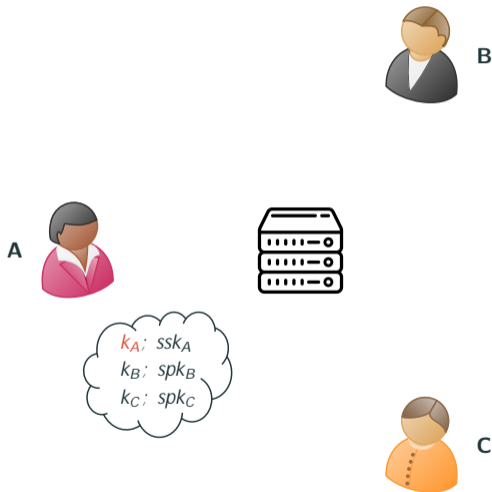
Sender Keys

- 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.**



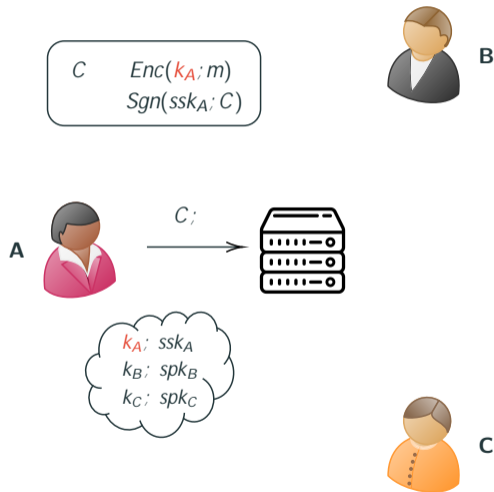
Sender Keys

- 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.**



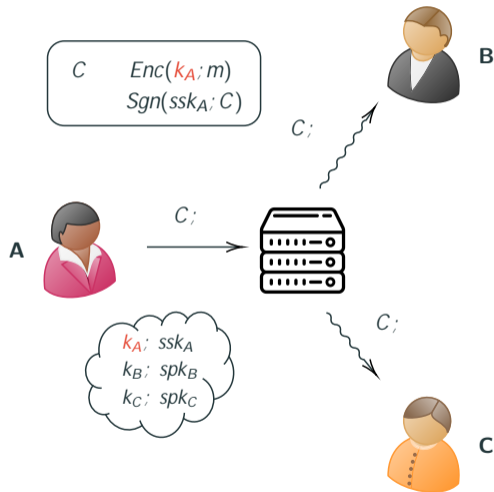
Sender Keys

- 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.**



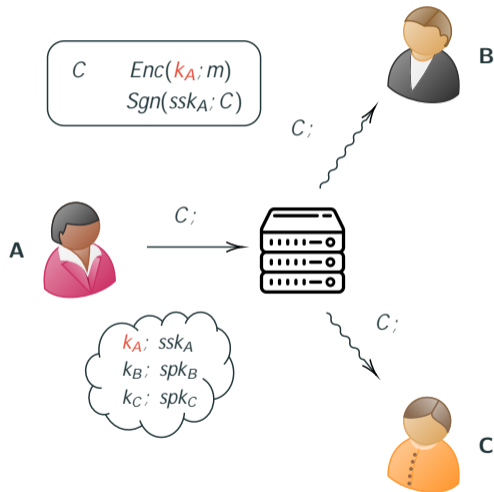
Sender Keys

- 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?

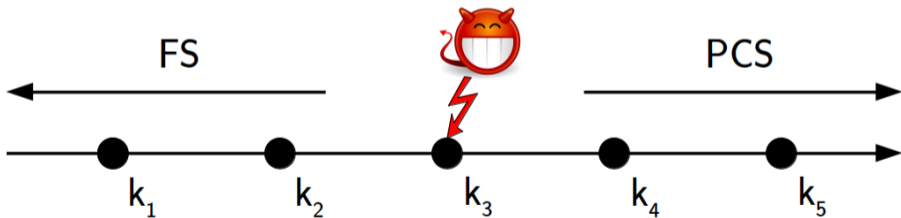
- Correct, authentic, confidential, and asynchronous messaging.
- **Secure membership.**

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, WhatsApp 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*?

- **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.

- **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.

- **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

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

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

- $C \stackrel{s}{\text{Send}}(m;)$

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

- $C \stackrel{\$}{\text{Send}}(m;)$
- $(m; ID ; e; i) \stackrel{\$}{\text{Recv}}(C;)$

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

- $C \stackrel{\$}{\text{Send}}(m;)$
- $(m; ID; e; i) \stackrel{\$}{\text{Recv}}(C;)$
- $T \stackrel{\$}{\text{Exec}}(\text{cmd}; IDs;), \text{cmd} \in \{\text{fcr t}; \text{add}; \text{rem}; \text{updg}\}$

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

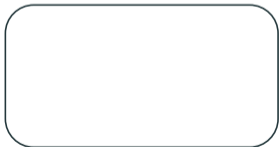
- $C \stackrel{\$}{\text{Send}}(m;)$
- $(m; ID; e; i) \stackrel{\$}{\text{Recv}}(C;)$
- $T \stackrel{\$}{\text{Exec}}(\text{cmd}; IDs;)$, $\text{cmd} \in \{\text{fcrt}; \text{add}; \text{rem}; \text{updg}\}$
- $b \stackrel{\$}{\text{Proc}}(T;)$

Sender Keys: Send & Recv

A



B



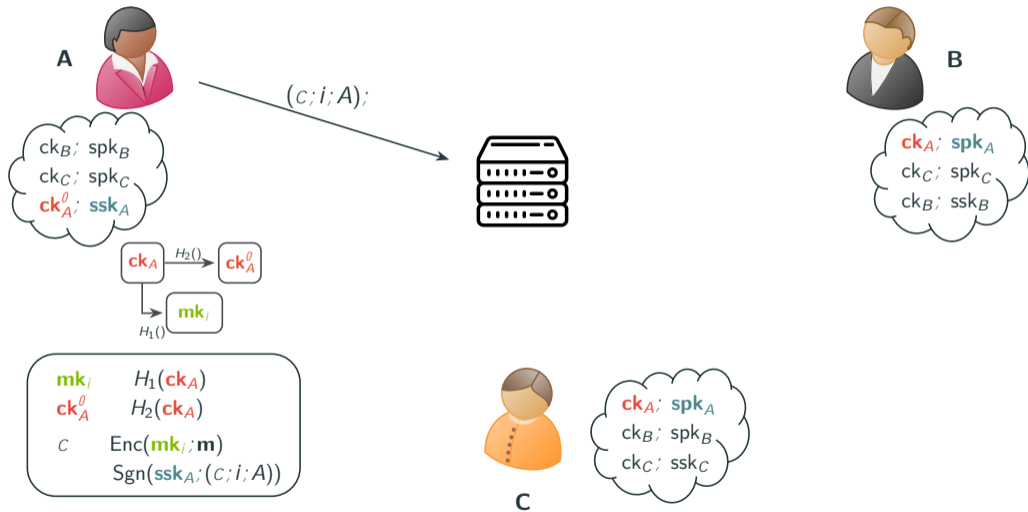
C



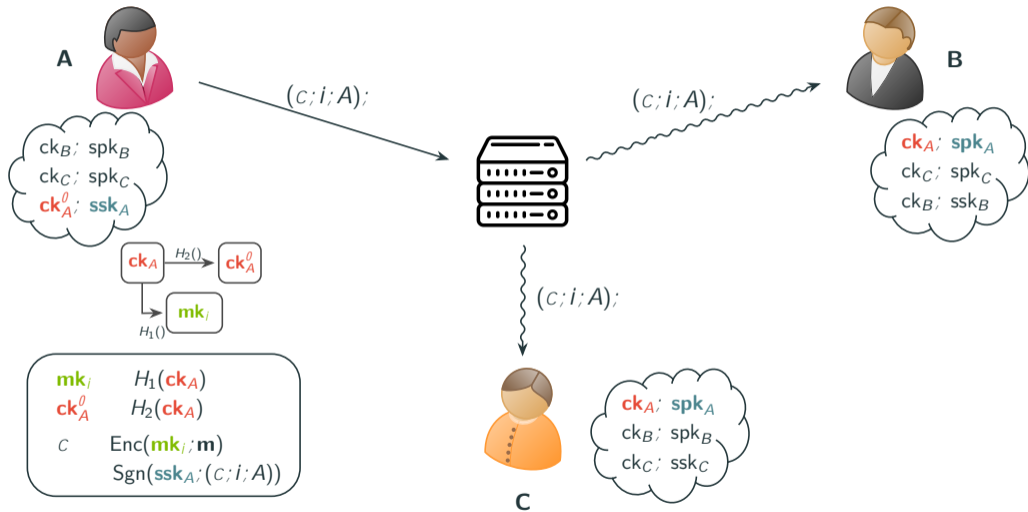
Sender Keys: Send & Recv



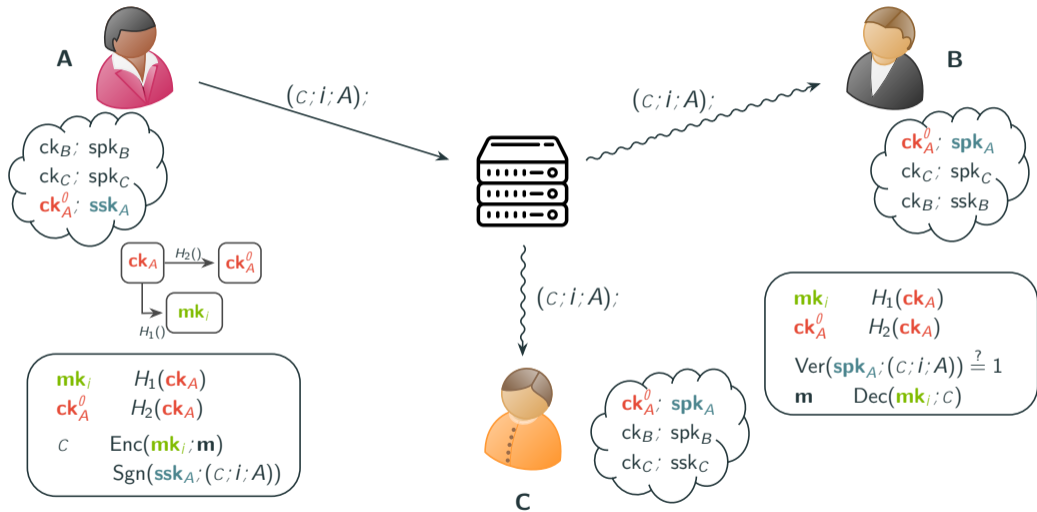
Sender Keys: Send & Recv



Sender Keys: Send & Recv

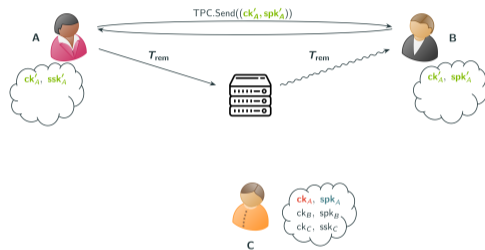


Sender Keys: Send & Recv



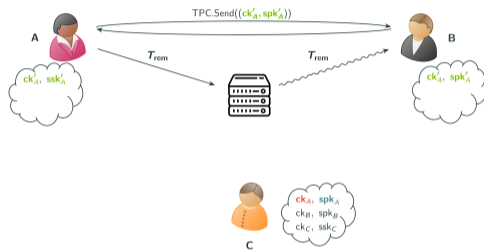
Two-Party Channels

- If C leaves (or someone updates):



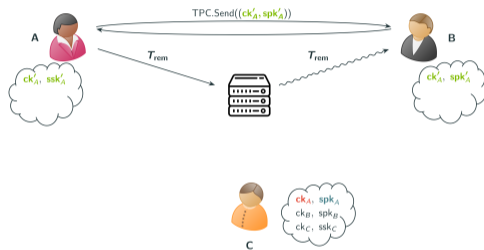
Two-Party Channels

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



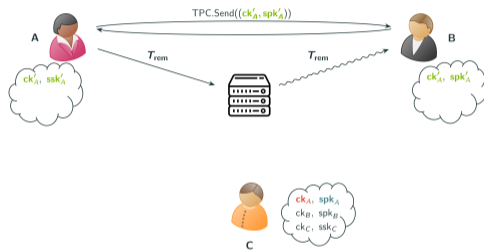
Two-Party Channels

- **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.

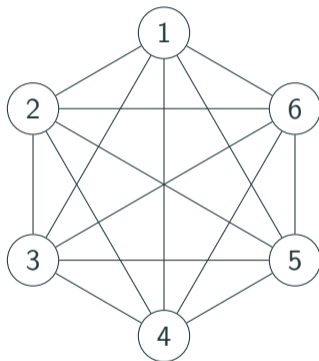


Modelling 2PC

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 **M-IND_C**.

Security Model

We introduce a *message indistinguishability* security game **M-IND_C**.

- Adaptive A can *forge and inject messages*.

Security Model

We introduce a *message indistinguishability* security game **M-IND_C**.

- Adaptive 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 **M-IND_C**.

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

Security Model

We introduce a *message indistinguishability* security game **M-IND_C**.

- Adaptive A can *forge and inject messages*.
- Users can be *exposed* at any time (capturing FS, PCS).
- 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 **M-IND_C**.

- Adaptive A can *forge and inject messages*.
- Users can be *exposed* at any time (capturing FS, PCS).
- 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 **M-IND_C**.

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

Oracles:

- Create($ID; IDs$)
- Challenge($ID; m_0; m_1$)
- Send($ID; m$)
- Receive($ID; C$)
- Add=Remove($ID; ID^0$)
- Update(ID)
- Deliver($ID; T$)
- Expose(ID)

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 $2PC\text{-IND}_{\Delta}$ two-party channels scheme for PCS bound $\Delta > 0$.

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

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 $2PC\text{-IND}_{\Delta}$ two-party channels scheme for PCS bound $\Delta > 0$.

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

Conclusion: The core of the protocol has *no fundamental aws*. 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, *inefficient 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 $G = fID_1; ID_2g$. Then A queries:

Sub-Optimal Forward Security

Let $G = fID_1; ID_2g$. Then A queries:

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

Sub-Optimal Forward Security

Let $G = fID_1; ID_2g$. Then A queries:

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

Sub-Optimal Forward Security

Let $G = fID_1; ID_2g$. Then A queries:

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

Sub-Optimal Forward Security

Let $G = fID_1; ID_2g$. Then A queries:

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

Sub-Optimal Forward Security

Let $G = fID_1; ID_2g$. Then A queries:

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

q_3 attempts to inject a message with keys from *before* exposure \Rightarrow should be allowed.

Can occur naturally e.g. if ID_2 is offline when m is first sent.

Sub-Optimal Forward Security

Let $G = fID_1; ID_2g$. Then A queries:

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

q_3 attempts to inject a message with keys from *before* exposure \Rightarrow 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

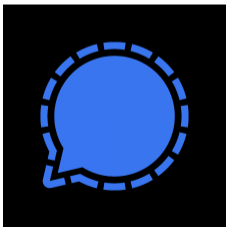
- In WhatsApp, **control messages** are *not signed, so a network adversary can forge them without exposing any party!*
- Server can add/remove parties on behalf of other users:
 - Burgle into the group attack [RMS18].
- Signal provides more protection but less than if signatures were used.

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

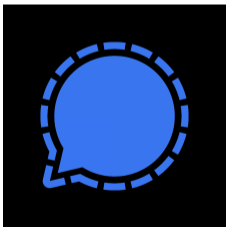
- Contrary to some folklore, Signal uses Sender Keys whenever possible!



[**matrix**]

Sender Keys in Practice

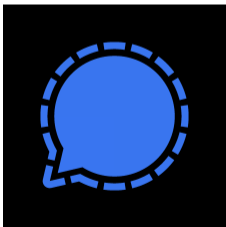
- Contrary to some folklore, Signal uses Sender Keys whenever possible!
- WhatsApp (resp. Signal) supports 1024 (resp. 256) member groups.



[**matrix**]

Sender Keys in Practice

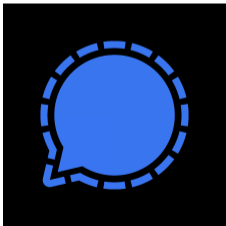
- Contrary to some folklore, Signal uses Sender Keys whenever possible!
- WhatsApp (resp. Signal) supports 1024 (resp. 256) member groups.
- Signal protects privacy more than WhatsApp (sealed sender, private groups...).



[**matrix**]

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.



[**matrix**]

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.

i a. cr/2023/1385

- 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!

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