Analysis and Improvements of the Sender Keys Protocol for Group Messaging

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

[1]IMDEA Software Institute, Madrid, Spain
[2]EPFL, Lausanne, Switzerland
[3]RUB & MPI-SP, Bochum, Germany

October 21, 2022

RECSI 2022, Santander

European Research Council
Established by the European Commission

WhatsUpp with Sender Keys?

- **Messaging protocols** used by billions daily.
  Commercial solutions claim **security** + **end-to-end encryption**.
- Formal protocol analysis is important. Becomes harder in **groups**.

- **Signal**: Extends Double Ratchet. *Slow; not completely understood.*
- **Telegram**: No end-to-end encryption. *Not ideal.*
- **MLS**: Lots of theoretical analysis. *Secure and efficient but complex.*
- **WhatsApp**: Sender Keys. *No protocol analysis so far.*

WhatsUpp with Sender Keys?

- **Messaging protocols** used by billions daily.
  Commercial solutions claim **security** + **end-to-end encryption**.
- Formal protocol analysis is important. Becomes harder in **groups**.

- **Signal**: Extends Double Ratchet. *Slow; not completely understood.*
- **Telegram**: No end-to-end encryption. *Not ideal.*
- **MLS**: Lots of theoretical analysis. *Secure and efficient but complex.*
- **WhatsApp**: Sender Keys. *No protocol analysis so far.*

Our Contributions

We study the **Sender Keys Protocol** used in WhatsApp groups.

Protocol extracted from WhatsApp's whitepaper + Signal code.

- **Formalization**: Cryptographic primitive, security modelling.
- **Security Analysis**: Issues with concurrency, group membership, recovery from compromise, authentication...
- **Improvements**: Patching our attacks, key updates, securing membership.

Results are preliminary.

Our Contributions

We study the **Sender Keys Protocol** used in WhatsApp groups.

Protocol extracted from WhatsApp's whitepaper + Signal code.

- **Formalization**: Cryptographic primitive, security modelling.
- **Security Analysis**: Issues with concurrency, group membership, recovery from compromise, authentication...
- **Improvements**: Patching our attacks, key updates, securing membership.

Results are preliminary.

Our Contributions

We study the **Sender Keys Protocol** used in WhatsApp groups.

Protocol extracted from WhatsApp's whitepaper + Signal code.

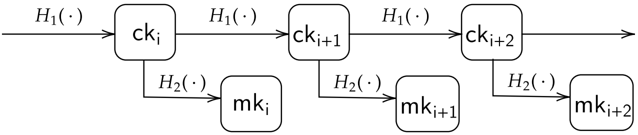
- **Formalization**: Cryptographic primitive, security modelling.
- **Security Analysis**: Issues with concurrency, group membership, recovery from compromise, authentication...
- **Improvements**: Patching our attacks, key updates, securing membership.

Results are preliminary.

Messaging and Sender Keys

Sender Keys: Main Protocol

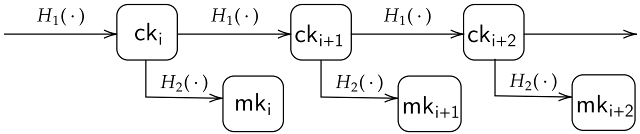
- Every ID $\in \boldsymbol{G}$ owns a symmetric *chain key* $\texttt{ck}_{\text{ID}}$ shared with all members.
- **Sending**: ID encrypts m using a *message key* $\texttt{mk}$ that is deterministically derived from $\texttt{ck}_{\text{ID}}$.



- **Receiving**, members derive $\texttt{mk}$ from $\texttt{ck}_{\text{ID}}$ to decrypt and read m.
- Forward security provided by a fresh $\texttt{mk}$ every time – *symmetric ratchet* using hash functions.
- Additionally, senders *sign application messages*.

Sender Keys: Main Protocol

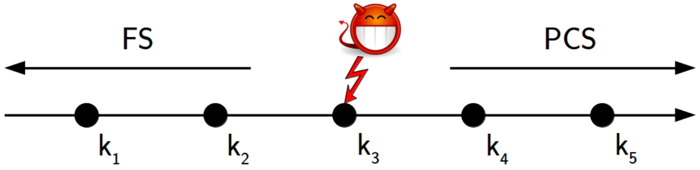
- Every ID $\in \boldsymbol{G}$ owns a symmetric *chain key* $\mathtt{ck}_{\mathrm{ID}}$ shared with all members.
- **Sending**: ID encrypts m using a *message key* $\mathtt{mk}$ that is deterministically derived from $\mathtt{ck}_{\mathrm{ID}}$.



- **Receiving**, members derive $\mathtt{mk}$ from $\mathtt{ck}_{\mathrm{ID}}$ to decrypt and read m.
- Forward security provided by a fresh $\mathtt{mk}$ every time – *symmetric ratchet* using hash functions.
- Additionally, senders *sign application messages*.

What is expected from Sender Keys?

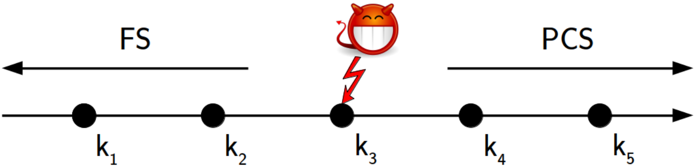
- Correctness, authentication.
- **Forward Security (FS)** - *past* messages safe.
- **Post-Compromise Security (PCS)** - *self-healing*



- Sender Keys does *not aim for strong PCS* in groups.
- **Secure Membership**, namely new users must not read previous messages and old users must not continue reading.

What is expected from Sender Keys?

- Correctness, authentication.
- **Forward Security (FS)** - *past* messages safe.
- **Post-Compromise Security (PCS)** - *self-healing*



- Sender Keys does *not aim for strong PCS* in groups.
- **Secure Membership**, namely new users must not read previous messages and old users must not continue reading.

Sender Keys: Message Exchange

$$\mathbf{A} : (\mathsf{ck}_A^i, \mathsf{ssk}_A, \mathsf{spk}_A, \mathsf{ck}_B^j, \mathsf{spk}_B)$$

$$\mathbf{B} : (\mathsf{ck}_B^j, \mathsf{ssk}_B, \mathsf{spk}_B, \mathsf{ck}_A^i, \mathsf{spk}_A)$$

A sends m_i

$$\mathsf{mk}_A^i \leftarrow H_1(\mathsf{ck}_A^i)$$

$$c_i \xleftarrow{\$} \mathsf{Enc}(\mathsf{mk}_A^i, m_i) \qquad \xrightarrow{(c_i, i, A, \sigma_i)} \qquad \textbf{req } \mathsf{Ver}(\mathsf{spk}_A, \sigma_i) = 1$$

$$\mathsf{ck}_A^{i+1} \leftarrow H_2(\mathsf{ck}_A^i) \qquad\qquad \mathsf{mk}_A^i \leftarrow H_1(\mathsf{ck}_A^i)$$

$$\sigma_i \xleftarrow{\$} \mathsf{Sig}(\mathsf{ssk}_A, (c_i, i, A)) \qquad\qquad m_i \leftarrow \mathsf{Dec}(\mathsf{mk}_A^i, c_i)$$

$$\mathsf{ck}_A^{i+1} \leftarrow H_2(\mathsf{ck}_A^{i+1})$$

Sender Keys: Message Exchange

A		B
$\mathbf{A} : (\mathsf{ck}_A^i, \mathsf{ssk}_A, \mathsf{spk}_A, \mathsf{ck}_B^j, \mathsf{spk}_B)$		$\mathbf{B} : (\mathsf{ck}_B^j, \mathsf{ssk}_B, \mathsf{spk}_B, \mathsf{ck}_A^i, \mathsf{spk}_A)$
	A sends m_i	
$\mathsf{mk}_A^i \leftarrow H_1(\mathsf{ck}_A^i)$		
$c_i \xleftarrow{\$} \mathsf{Enc}(\mathsf{mk}_A^i, m_i)$	$\xrightarrow{(c_i, i, A, \sigma_i)}$	**req** $\mathsf{Ver}(\mathsf{spk}_A, \sigma_i) = 1$
$\mathsf{ck}_A^{i+1} \leftarrow H_2(\mathsf{ck}_A^i)$		$\mathsf{mk}_A^i \leftarrow H_1(\mathsf{ck}_A^i)$
$\sigma_i \xleftarrow{\$} \mathsf{Sig}(\mathsf{ssk}_A, (c_i, i, A))$		$m_i \leftarrow \mathsf{Dec}(\mathsf{mk}_A^i, c_i)$
		$\mathsf{ck}_A^{i+1} \leftarrow H_2(\mathsf{ck}_A^{i+1})$
	B sends m_j	
		$\mathsf{mk}_B^j \leftarrow H_1(\mathsf{ck}_B^j)$
req $\mathsf{Ver}(\mathsf{spk}_B, \sigma_j) = 1$	$\xleftarrow{(c_j, j, B, \sigma_j)}$	$c_j \xleftarrow{\$} \mathsf{Enc}(\mathsf{mk}_B^j, m_j)$
$\mathsf{mk}_B^j \leftarrow H_1(\mathsf{ck}_B^j)$		$\mathsf{ck}_B^{j+1} \leftarrow H_2(\mathsf{ck}_B^j)$
$m_j \leftarrow \mathsf{Dec}(\mathsf{mk}_B^j, c_j)$		$\sigma_j \xleftarrow{\$} \mathsf{Sig}(\mathsf{ssk}_B, (c_j, j, B))$
$\mathsf{ck}_B^{j+1} \leftarrow H_2(\mathsf{ck}_B^j)$		

Sender Keys: Key Agreement & Membership

Sender Keys relies on *existing authenticated and confidential two-party channels (2pc)* between all users (strong assumption!).

- If ID joins $\boldsymbol{G}$, it generates new `ck` and `spk` and sends it to everyone in $\boldsymbol{G}$ via 2pc. This is done the first time ID speaks.
- If ID leaves, **everyone deletes keys**, generates fresh ones and restarts the protocol using 2pc. $\mathcal{O}(n^2)$ total communication.

Sender Keys: Key Agreement & Membership

Sender Keys relies on *existing authenticated and confidential two-party channels (2pc)* between all users (strong assumption!).

- If ID joins $\boldsymbol{G}$, it generates new `ck` and `spk` and sends it to everyone in $\boldsymbol{G}$ via 2pc. This is done the first time ID speaks.
- If ID leaves, **everyone deletes keys**, generates fresh ones and restarts the protocol using 2pc. $\mathcal{O}\left(n^2\right)$ total communication.

Security

Primitive and Security Model

A Group Messenger (GM) includes:

- $(C, \gamma') \xleftarrow{\$} Send(m, \gamma)$
- $(m, e, i, \gamma') \leftarrow Recv(C, \gamma)$
- $(T, \gamma') \xleftarrow{\$} Exec(\texttt{cmd}, \boldsymbol{IDs}, \gamma)$
- $\gamma' \leftarrow Proc(T, \gamma)$

We introduce a *message indistinguishability* security game.

Active, adaptive $\mathcal{A}$ that can *forge and inject messages*.

We disallow *'trivial attacks'*: challenge and inject using exposed keys.

Game Oracles:

- Create(ID, ***IDs***)
- Challenge(ID, m_0, m_1)
- Send(ID, m)
- Receive(ID, ID′, C)
- Add(ID, ID′)
- Remove(ID, ID′)
- Update(ID)
- Deliver(ID, T)
- Expose(ID)
- ExpMK(ID, e, i)
- Send2PC(ID, ID')
- Receive2PC(ID, ID', e, i)

Primitive and Security Model

A Group Messenger (GM) includes:

- $(C, \gamma') \xleftarrow{\$} Send(m, \gamma)$
- $(m, e, i, \gamma') \leftarrow Recv(C, \gamma)$
- $(T, \gamma') \xleftarrow{\$} Exec(\texttt{cmd}, \boldsymbol{IDs}, \gamma)$
- $\gamma' \leftarrow Proc(T, \gamma)$

We introduce a *message indistinguishability* security game.

Active, adaptive $\mathcal{A}$ that can *forge and inject messages*.

We disallow *'trivial attacks'*: challenge and inject using exposed keys.

Game Oracles:

- `Create`(ID, ***IDs***)
- `Challenge`(ID, m_0, m_1)
- `Send`(ID, m)
- `Receive`(ID, ID′, C)
- `Add`(ID, ID′)
- `Remove`(ID, ID′)
- `Update`(ID)
- `Deliver`(ID, T)
- `Expose`(ID)
- `ExpMK`(ID, e, i)
- `Send2PC`(ID, ID')
- `Receive2PC`(ID, ID', e, i)

Primitive and Security Model

A Group Messenger (GM) includes:

- $(C, \gamma') \xleftarrow{\$} Send(m, \gamma)$
- $(m, e, i, \gamma') \leftarrow Recv(C, \gamma)$
- $(T, \gamma') \xleftarrow{\$} Exec(\texttt{cmd}, \boldsymbol{IDs}, \gamma)$
- $\gamma' \leftarrow Proc(T, \gamma)$

We introduce a *message indistinguishability* security game.

Active, adaptive $\mathcal{A}$ that can *forge and inject messages*.

We disallow *'trivial attacks'*: challenge and inject using exposed keys.

Game Oracles:

- $\texttt{Create}(\text{ID}, \boldsymbol{IDs})$
- $\texttt{Challenge}(\text{ID}, m_0, m_1)$
- $\texttt{Send}(\text{ID}, m)$
- $\texttt{Receive}(\text{ID}, \text{ID}', C)$
- $\texttt{Add}(\text{ID}, \text{ID}')$
- $\texttt{Remove}(\text{ID}, \text{ID}')$
- $\texttt{Update}(\text{ID})$
- $\texttt{Deliver}(\text{ID}, T)$
- $\texttt{Expose}(\text{ID})$
- $\texttt{ExpMK}(\text{ID}, e, i)$
- Send2PC(ID, ID')
- Receive2PC(ID, ID', e, i)

Primitive and Security Model

A Group Messenger (GM) includes:

- $(C, \gamma') \xleftarrow{\$} Send(m, \gamma)$
- $(m, e, i, \gamma') \leftarrow Recv(C, \gamma)$
- $(T, \gamma') \xleftarrow{\$} Exec(\texttt{cmd}, \boldsymbol{IDs}, \gamma)$
- $\gamma' \leftarrow Proc(T, \gamma)$

We introduce a *message indistinguishability* security game.

Active, adaptive $\mathcal{A}$ that can *forge and inject messages*.

We disallow *'trivial attacks'*: challenge and inject using exposed keys.

Game Oracles:

- $\texttt{Create}(\mathsf{ID}, \boldsymbol{IDs})$
- $\texttt{Challenge}(\mathsf{ID}, m_0, m_1)$
- $\texttt{Send}(\mathsf{ID}, m)$
- $\texttt{Receive}(\mathsf{ID}, \mathsf{ID}', C)$
- $\texttt{Add}(\mathsf{ID}, \mathsf{ID}')$
- $\texttt{Remove}(\mathsf{ID}, \mathsf{ID}')$
- $\texttt{Update}(\mathsf{ID})$
- $\texttt{Deliver}(\mathsf{ID}, T)$
- $\texttt{Expose}(\mathsf{ID})$
- $\texttt{ExpMK}(\mathsf{ID}, e, i)$
- Send2PC(ID, ID')
- Receive2PC(ID, ID', e, i)

Two Attacks

Assuming ideal two-party channels, we still find some issues:

Control Messages

These are *not authenticated* and can be forged without any exposure.

Server can add/remove parties on behalf of other users. *Insecure membership* [RMS18, ACDJ22, BCV22].

Sub-Optimal Forward Security

It is possible to *inject* messages using (signature) keys from *before* a state exposure occurs.

Can be mitigated with MACs / refreshing signature keys.

Two Attacks

Assuming ideal two-party channels, we still find some issues:

Control Messages

These are *not authenticated* and can be forged without any exposure.

Server can add/remove parties on behalf of other users. *Insecure membership* [RMS18, ACDJ22, BCV22].

Sub-Optimal Forward Security

It is possible to *inject* messages using (signature) keys from *before* a state exposure occurs.

Can be mitigated with MACs / refreshing signature keys.

An Improvement: PCS Updates

Only 'removes' allow for PCS in Sender Keys. Can we do updates?

- **Naive approach:** ID sends a fresh ck to all users [CHK21]. Problem: only messages encrypted under ck recover security.
- **Alternative idea:** ID sends *fresh randomness* r to all users; $\mathtt{ck}_{\mathsf{ID}'} \leftarrow H(\mathtt{ck}_{\mathsf{ID}'} || r)$ is computed for all ID′s.

We *improve removals* from $O(n^2)$ to $O(n)$ communication.

An Improvement: PCS Updates

Only 'removes' allow for PCS in Sender Keys. Can we do updates?

- **Naive approach:** ID sends a fresh `ck` to all users [CHK21]. Problem: only messages encrypted under `ck` recover security.
- **Alternative idea:** ID sends *fresh randomness* r to all users; $\mathtt{ck}_{\mathsf{ID}'} \leftarrow H(\mathtt{ck}_{\mathsf{ID}'} || r)$ is computed for all ID's.

We *improve removals* from $O(n^2)$ to $O(n)$ communication.

An Improvement: PCS Updates

Only 'removes' allow for PCS in Sender Keys. Can we do updates?

- **Naive approach:** ID sends a fresh $\mathtt{ck}$ to all users [CHK21]. Problem: only messages encrypted under $\mathtt{ck}$ recover security.
- **Alternative idea:** ID sends *fresh randomness* r to all users; $\mathtt{ck}_{\mathsf{ID}'} \leftarrow H(\mathtt{ck}_{\mathsf{ID}'} || r)$ is computed for all ID's.

We *improve removals* from $O(n^2)$ to $O(n)$ communication.

An Improvement: PCS Updates

Only 'removes' allow for PCS in Sender Keys. Can we do updates?

- **Naive approach:** ID sends a fresh $\texttt{ck}$ to all users [CHK21]. Problem: only messages encrypted under $\texttt{ck}$ recover security.
- **Alternative idea:** ID sends *fresh randomness* r to all users; $\texttt{ck}_{\mathsf{ID}'} \leftarrow H(\texttt{ck}_{\mathsf{ID}'} || r)$ is computed for all ID's.

We *improve removals* from $O(n^2)$ to $O(n)$ communication.

Realistic two-party Channels

If ID is removed, assuming secure 2pc:

- Members process removal & erase keys - free from every exposure!
- Generate and send fresh keys over secure 2pc.

Looks great! Keys sent safely, key material erased, exposures resolved. PCS is achieved.

In reality, *New keys sent encrypted...* under **Double ratchet keys!** DR sessions are not 100% safe.

Fine-grained modelling leads to more attacks.

Realistic two-party Channels

If ID is removed, assuming secure 2pc:

- Members process removal & erase keys - free from every exposure!
- Generate and send fresh keys over secure 2pc.

Looks great! Keys sent safely, key material erased, exposures resolved. PCS is achieved.

In reality, *New keys sent encrypted...* under **Double ratchet keys!** DR sessions are not 100% safe.

Fine-grained modelling leads to more attacks.

Realistic two-party Channels

If ID is removed, assuming secure 2pc:

- Members process removal & erase keys - free from every exposure!
- Generate and send fresh keys over secure 2pc.

Looks great! Keys sent safely, key material erased, exposures resolved. PCS is achieved.

In reality, *New keys sent encrypted...* under **Double ratchet keys!** DR sessions are not 100% safe.

Fine-grained modelling leads to more attacks.

Final Remarks

Conclusions and Future Work

Takeaways:

- *Analysis*: Formalization, weaknesses, comparison to other protocols, concurrency.
- *Improvements*: Update options (even if strong PCS impossible), efficiency, security.

Work in progress:
Complete analysis with *realistic two-party channels*, further improvements.

Conclusions and Future Work

Takeaways:

- *Analysis*: Formalization, weaknesses, comparison to other protocols, concurrency.
- *Improvements*: Update options (even if strong PCS impossible), efficiency, security.

Work in progress:
Complete analysis with *realistic two-party channels*, further improvements.

Conclusions and Future Work

Takeaways:

- *Analysis*: Formalization, weaknesses, comparison to other protocols, concurrency.
- *Improvements*: Update options (even if strong PCS impossible), efficiency, security.

Work in progress:

Complete analysis with *realistic two-party channels*, further improvements.

So, WhatsUpp with Sender Keys?

¡Gracias!

Slides (and more!) at:
`davidbalbas.github.io`

Appendix: Single- vs multi-key

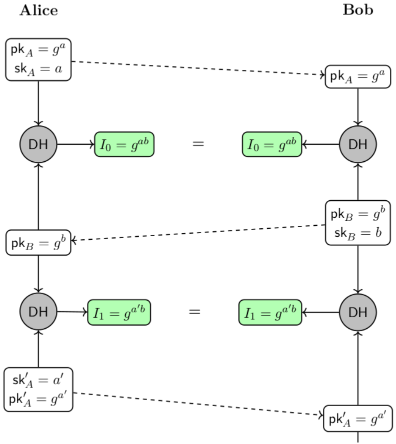
Each ID has a different `ck`, so the state has $O(n)$ secret material at all times. We observe:

- Security comparable if all users have *the same chain key* (single-key group). Everyone knows all secret material except signature keys.
- Different chain keys are essentially useful for *concurrency*.
- One could envision trade-offs depending on how active users are. A central server could be employed to help.

For example, in MLS a common group secret is agreed (with possible PCS). Then, $O(n)$ application keys are derived from it to improve concurrency.

PCS after users leave

In the Double Ratchet, we require a full roundtrip for state exposure recovery (hence for PCS).



An attack vector

1. ID is *exposed*. Then $\mathcal{A}$ knows its DR keys with everyone.
2. Someone leaves the group. Users erase their old keys.
3. ID' sends a new sender key to ID via their DR.
4. $\mathcal{A}$ can read the key despite ID's updated pk.

This raises **open questions:**

- Can we *improve* this mechanism?
- Do we need a fresh, *interactive* key exchange?
- Can PCS be recovered at all (just by sending keys)?
- What is the *exact* security we get?

Current approach: simplified modelling of underlying channels

Security in ideal model

We disallow 'trivial attacks' when party ID is exposed:

- Cannot inject (via `Receive`) previous messages and future until ID is removed;
- Cannot `Challenge` using keys learnt from ID until removal of ID′/update.
- Game only delivers honestly generated control messages.

Then, we can prove security assuming ideal 2pc. Some remarks:

- Message keys $\mathtt{mk}_i$ can be exposed independently, never affecting other keys.
- *Assuming perfect 2pc*, users recover from exposure after a removal.