Group Signatures with User-Controlled and Sequential Linkability

Jesus Diaz[1], Anja Lehmann[2]

[1]IBM Research Europe

[2]Hasso-Plattner-Institute

May 10, 2021

Linkability in Group Signatures

Challenge

- Group signatures have a central party that can open/link.

Linkability in Group Signatures

Challenge

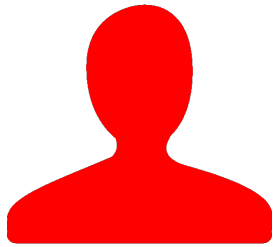
- Group signatures have a central party that can open/link.
- Privacy invasive and needs trust.

Linkability in Group Signatures

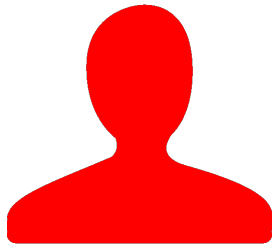
Challenge

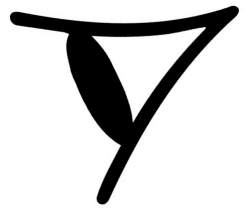
- Group signatures have a central party that can open/link.
- Privacy invasive and needs trust.
- Can we get rid of it?

User-Controlled Linkability in Group Signatures



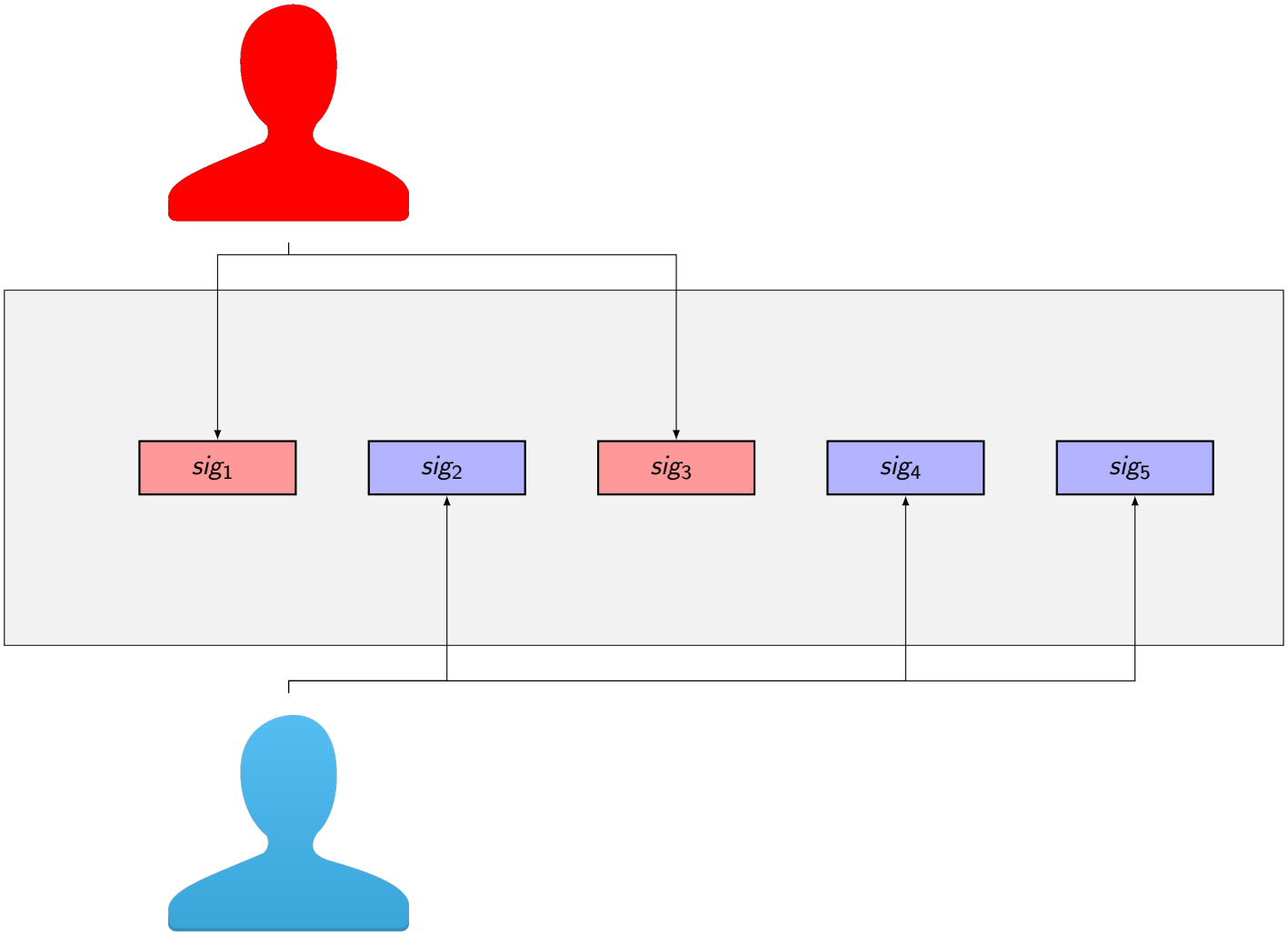
User-Controlled Linkability in Group Signatures



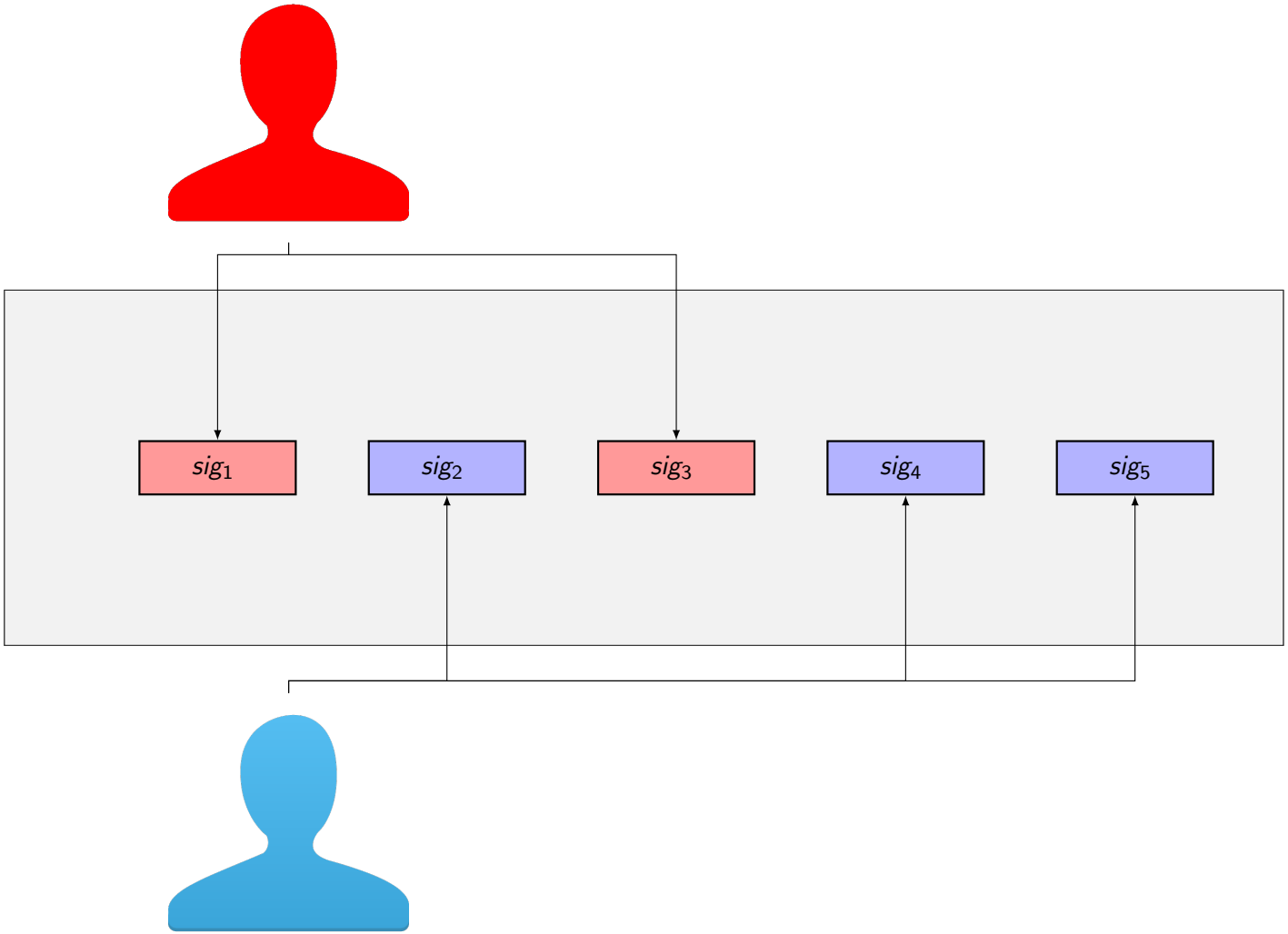


IBM

User-Controlled Linkability in Group Signatures



User-Controlled Linkability in Group Signatures



User-Controlled Linkability in Group Signatures

Syntax

	UCL Group Signatures
Setup	$(gpk, gsk) \leftarrow \mathsf{Setup}(1^\tau)$
Join,Issue	$gsk_i \leftarrow \langle \mathsf{Join}_{gpk}(sk_i), \mathsf{Issue}_{gpk}(gsk) \rangle$
Sign	$\sigma \leftarrow \mathsf{Sign}_{gsk_i}(m)$
Verify	$1/0 \leftarrow \mathsf{Verify}_{gpk}(m, \sigma)$
Link	$\pi \leftarrow \mathsf{Link}_{gsk_i}(\{(m_i, \sigma_i)\}_{i \in [n]})$
VerifyLink	$1/0 \leftarrow \mathsf{VerifyLink}_{gpk}(\pi, \{(m_i, \sigma_i)\}_{i \in [n]})$

User-Controlled Linkability in Group Signatures

Syntax

	UCL Group Signatures
Setup	$(gpk, gsk) \leftarrow \mathsf{Setup}(1^\tau)$
Join,Issue	$gsk_i \leftarrow \langle \mathsf{Join}_{gpk}(sk_i), \mathsf{Issue}_{gpk}(gsk) \rangle$
Sign	$\sigma \leftarrow \mathsf{Sign}_{gsk_i}(m)$
Verify	$1/0 \leftarrow \mathsf{Verify}_{gpk}(m, \sigma)$
Link	$\pi \leftarrow \mathsf{Link}_{gsk_i}(\{(m_i, \sigma_i)\}_{i \in [n]})$
VerifyLink	$1/0 \leftarrow \mathsf{VerifyLink}_{gpk}(\pi, \{(m_i, \sigma_i)\}_{i \in [n]})$

Only users can run Link**!** (on their own sigs.)

Modelling UCL Group Signatures

... without Open

- We do not have open, traditionally crucial for modelling non-frameability and traceability.

Modelling UCL Group Signatures

... without Open

- We do not have open, traditionally crucial for modelling non-frameability and traceability.
  - Non-frame: sigs do not open to a member who did not create them.
  - Trace: sigs open to a valid member.

Modelling UCL Group Signatures

... without Open

- We do not have open, traditionally crucial for modelling non-frameability and traceability.
  - Non-frame: sigs do not open to a member who did not create them.
  - Trace: sigs open to a valid member.
- Leverage *nym* approach from DAA (and GL19 group sigs).

Modelling UCL Group Signatures

the *nym* approach – with Identify helper

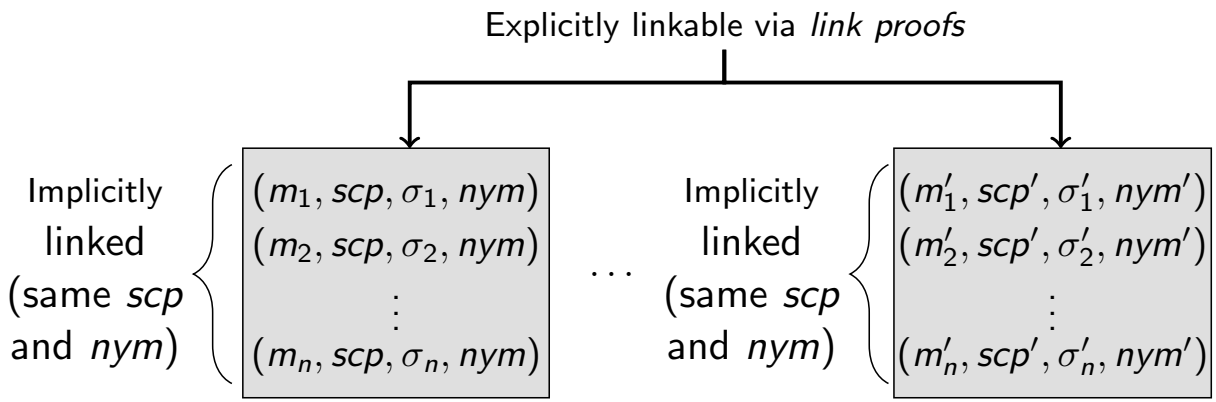
- Signatures are accompanied by a pseudonym, *nym* deterministically produced from a user-chosen scope *scp*.
  - Same *scp* $\implies$ same *nym* $\implies$ same signer.
- This directly gives *implicit* user-controlled linkability.
- *Explicit* user-controlled linkability, achieved via Link.

IBM

Modelling UCL Group Signatures

the *nym* approach – with Identify helper

- Signatures are accompanied by a pseudonym, *nym* deterministically produced from a user-chosen scope *scp*.
  - Same *scp* $\implies$ same *nym* $\implies$ same signer.
- This directly gives *implicit* user-controlled linkability.
- *Explicit* user-controlled linkability, achieved via Link.



Modelling UCL Group Signatures

the *nym* approach – with Identify helper

- Non-frame: Sigs. by different users cannot be linked.
- Trace: All sigs. can be associated via Identify to a valid member.
  - Need to extract secret keys.

Modelling UCL Group Signatures

the *nym* approach – with Identify helper

- Non-frame: Sigs. by different users cannot be linked.
- Trace: All sigs. can be associated via Identify to a valid member.
  - Need to extract secret keys.
- Both properties have to cover implicit and explicit linkability!

UCL Signatures

Instantiation

- BBS+ signatures for the membership credentials.
- Signing requires proving in ZK knowledge of a BBS+ sig.
- Batching technique for efficient linking.

Sequential UCL Signatures

Why?

- Without an opener, users can easily hide data.
- This can be good... but also not good.

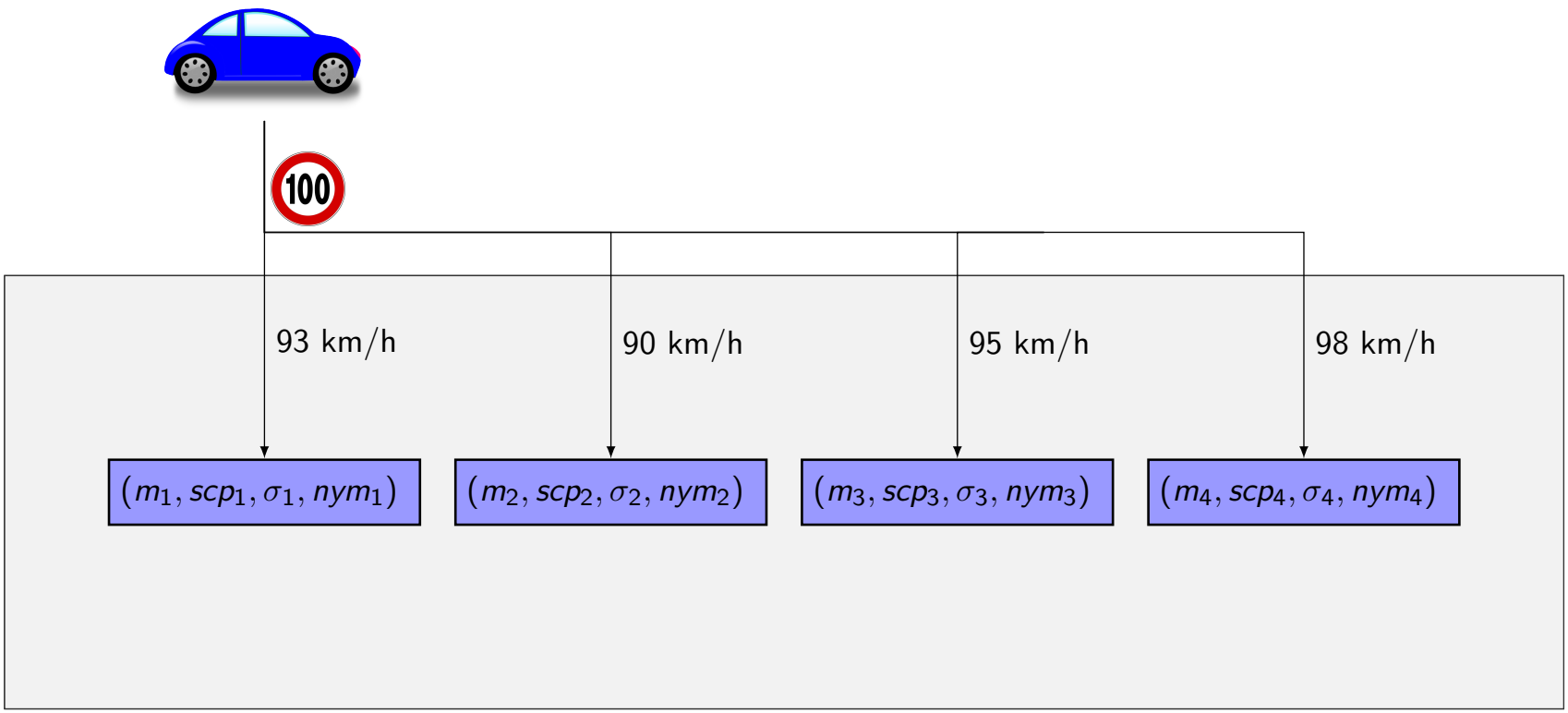
Sequential UCL Signatures

Why?

- Without an opener, users can easily hide data.
- This can be good... but also not good.
- Compromise: sequential linking.
  - For any given sequence of signatures by a same user: link proofs cannot alter order, omit or insert signatures.

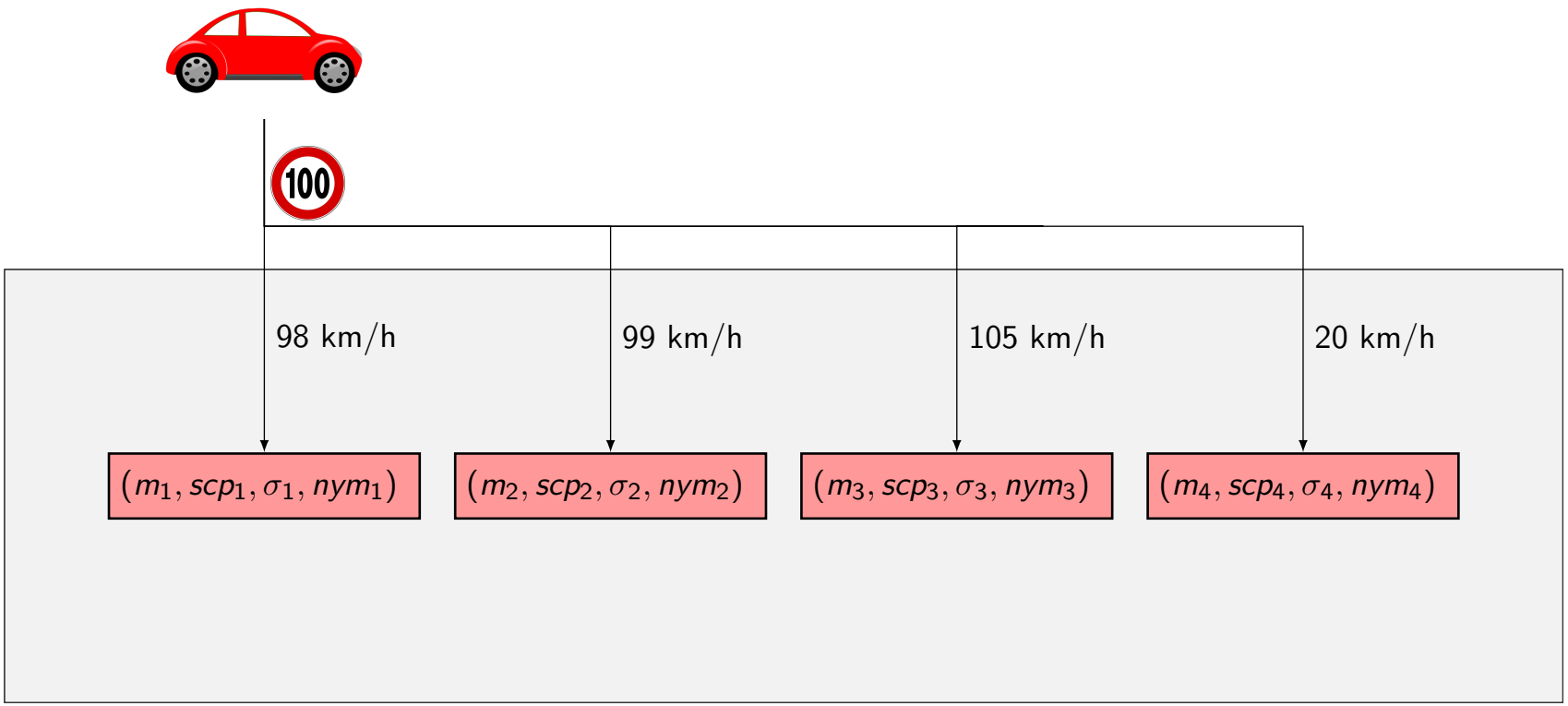
Sequential UCL Signatures

Why?



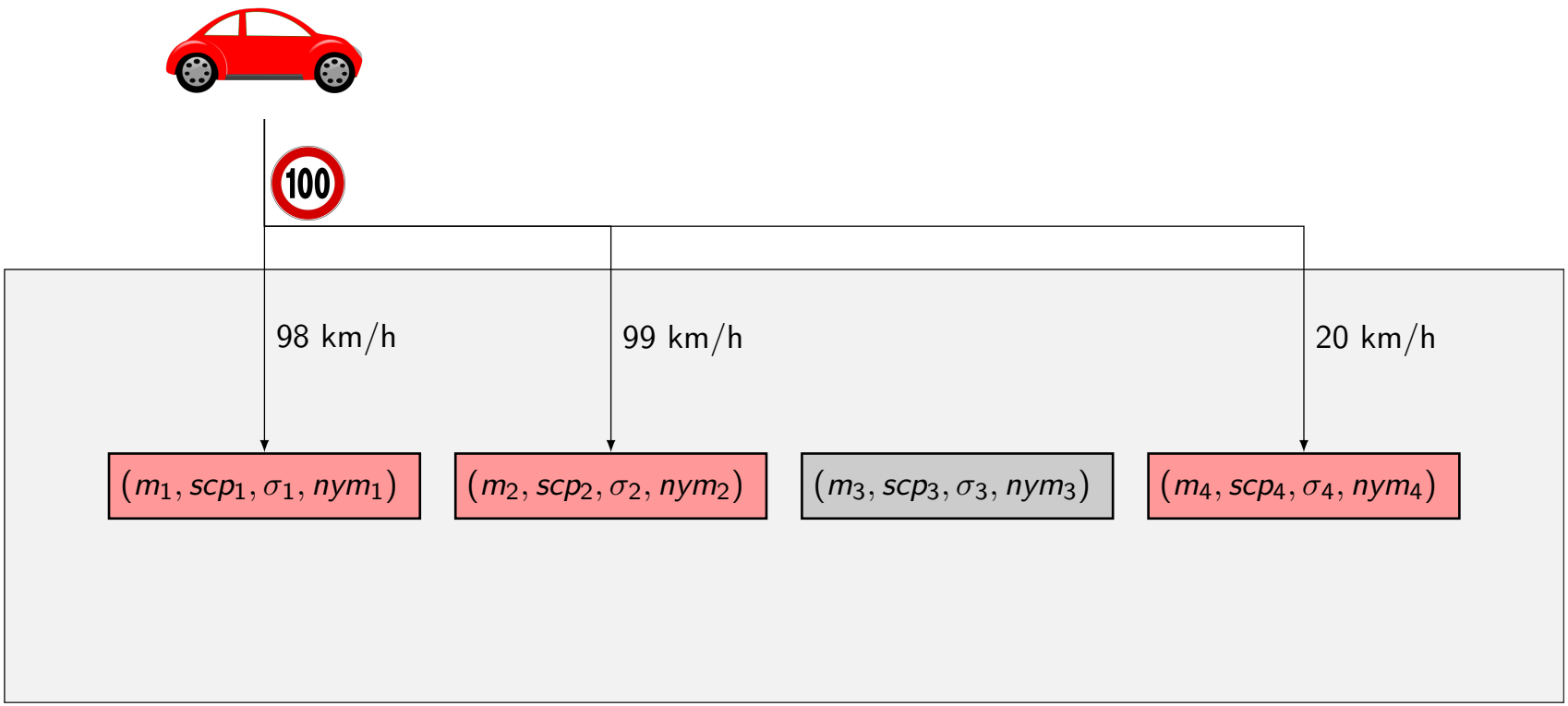
Sequential UCL Signatures

Why?



Sequential UCL Signatures

Why?



Sequential UCL Signatures

Syntax

	UCL Group Signatures
Setup	$(gpk, gsk) \leftarrow \mathsf{Setup}(1^\tau)$
Join,Issue	$gsk_i \leftarrow \langle \mathsf{Join}_{gpk}(sk_i), \mathsf{Issue}_{gpk}(gsk) \rangle$
Sign	$\sigma \leftarrow \mathsf{Sign}_{gsk_i}(m, seq)$
Verify	$1/0 \leftarrow \mathsf{Verify}_{gpk}(m, \sigma)$
Seq. Link	$\pi_s \leftarrow \mathsf{SeqLink}_{gsk_i}([(m_i, \sigma_i)]_{i \in [n]})$
Verify Seq. Link	$1/0 \leftarrow \mathsf{VerifySeqLink}_{gpk}(\pi_s, [(m_i, \sigma_i)]_{i \in [n]})$

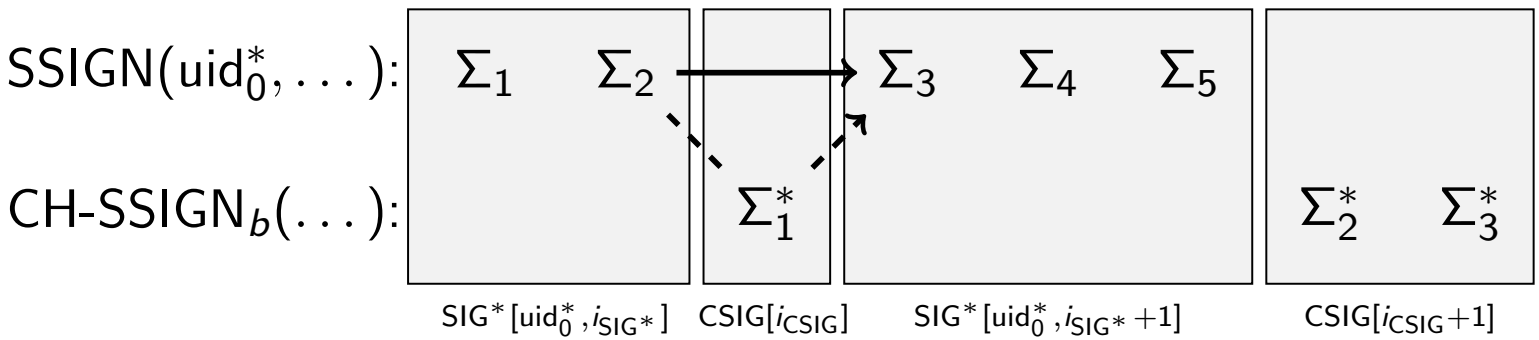
* *scp* and *nym* ommitted for readability!

Modelling Sequential UCL Group Signatures

- **Non-frameability** and **traceability** as UCL signatures.
- For **anonymity** we need to prevent (more) trivial wins by adversaries exploiting order info!
  - Cumbersome, but otherwise similar anonymity notion.

Modelling Sequential UCL Group Signatures

- **Non-frameability** and **traceability** as UCL signatures.
- For **anonymity** we need to prevent (more) trivial wins by adversaries exploiting order info!
  - Cumbersome, but otherwise similar anonymity notion.



$$\Sigma_i = (m_i, scp_i, \sigma_i, nym_i)$$
$$\Sigma_i^* = (m_i^*, scp_i^*, \sigma_i^*, nym_i^*)$$

Modelling Sequential UCL Group Signatures

Sequentiality Property

- Captures that no honest-then-corrupt user can:
  - Swap, omit or insert signatures in a previous sequence.
- Two-phase game:
  - **Choose phase**: $\mathcal{A}$ commits to a sequence of signatures (maybe both honest and dishonest), and picks a target user (honest up to the second phase).
  - **Forge phase**: $\mathcal{A}$ has to forge a sequential proof that includes:
    - At least one honest signature by the target user (of which $\mathcal{A}$ receives the secret key).
    - Signatures contained in the previously committed set.
  - $\mathcal{A}$ wins if he produces a valid sequential proof that alters the order in the sequence committed to in the choose phase.

Sequential UCL Signatures

Instantiation

- We rely on an *append-only bulletin board*.
  - Trusted to verify all signatures before appending.
  - But cannot open/link!

Sequential UCL Signatures

Instantiation

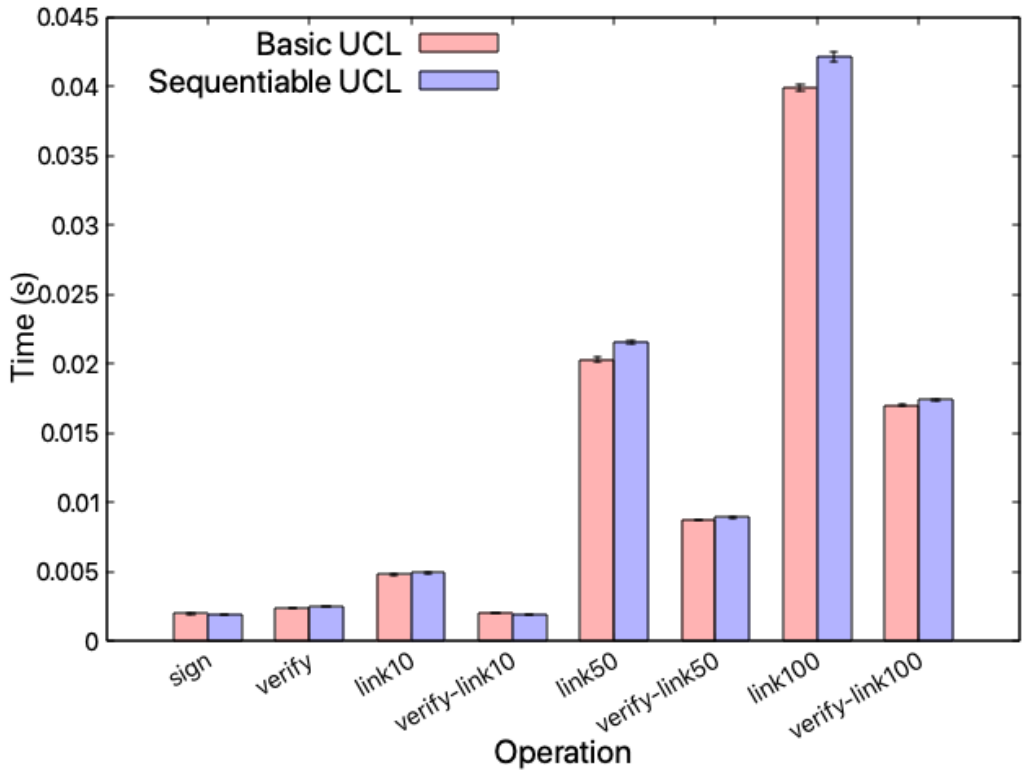
On top of UCL signatures:

- $seq_i = (\mathsf{H}(x_i), \mathsf{H}(x_i \oplus x_{i-1}), n_i)$.
  - $x_i = \mathsf{PRF}(k, n_i)$.
- To sequentially link sig_{i-1} and sig_i: Reveal (x_{i-1}, x_i).

[Sequential] UCL Signatures

Evaluation

Signature	$4\mathbb{G}_1 + 1\mathsf{H} + 5\mathbb{Z}_p + 3\mathsf{H}$
Linkability Proof (s sigs.)	$1\mathsf{H} + 1\mathbb{Z}_p + s\mathbb{Z}_p$



More in the paper.

[Sequential] UCL Signatures

Implementation

https://github.com/IBM/libgroupsig

- Core library in C.
- Wrapper for Python.
- Wrapper for Java.
- Wrapper for NodeJS.

[Sequential] UCL Signatures

Implementation

Supports:
BBS04, KTY04, PS16, GL19, KLAP20,
DL21... and more schemes coming!

https://github.com/IBM/libgroupsig

- Core library in C.
- Wrapper for Python.
- Wrapper for Java.
- Wrapper for NodeJS.

[Sequential] UCL Signatures

Demo

1. `$ docker pull jdiazvico/sucl:latest`
2. `$ docker run -p 5000:5000 jdiazvico/sucl`
3. Access `http://127.0.0.1:5000` on your browser.

[Sequential] UCL Signatures

Further work

- Proof of **not** linked signatures.
  - Blacklisting.
- Security against initially corrupt users?
- Batched verification.

Thanks!

Questions?

@JesusDiazVico

Work supported by the EU's H2020 under
Grant Agreement Number 76 8953 (ICT4CART).