Security in Times of Surveillance

Are you old enough to buy this?

Zero-Knowledge Age Restriction for GNU Taler

Özgür Kesim

31 May 2024

Code Blau GmbH, FU Berlin, TU Dresden

Prolog

Sponsors

NGI Taler and NGI Pointer programs of the European Commission

NGI TALER

TALER

NGI POINTER

Project *Concrete Contracts* in the *KMU-innovativ* programm

SPONSORED BY THE

Federal Ministry of Education and Research

Who am I

Özgür Kesim,

- security consultant for 20+ years,
- PhD candidate at FU Berlin,
- member of GNU Taler dev-team.

`oec-taler@kesim.org` `@oec@mathstodon.xyz`

What to expect

Deliverable

Present a solution to age restriction and its integration in GNU Taler.

What to expect

Deliverable

Present a solution to age restriction and its integration in GNU Taler.

Drive-By

Show concepts from cryptography by example:

Zero-Knowledge protocol, Security Game and Security Proof

What to expect

Deliverable

Present a solution to age restriction and its integration in GNU Taler.

Drive-By

Show concepts from cryptography by example:

Zero-Knowledge protocol, Security Game and Security Proof

Non-goals

~~Rigorous~~ introduction into GNU Taler

~~Demos~~

Chapters

Introduction

Chapters

Introduction

The quest for a solution to age restriction

Chapters

Introduction

The quest for a solution to age restriction

Integration with GNU Taler

Chapters

Introduction

The quest for a solution to age restriction

Integration with GNU Taler

Discussion & Conclusion

Introduction

Age Restriction in E-commerce

Youth protection

Broad consensus in society about the necessity to protect minors from harmful content.

Also wanted from policy makers:

> *11. Member states should encourage the **use of conditional access tools** by content and service providers in relation to content harmful to minors, **such as age-verification systems**, ...*

From the *Recommendation Rec (2001) 8 of the Committee of Ministers to member states on self-regulation concerning cyber content* of the Council of Europe.

Age restriction in E-commerce

Problem:

Verification of minimum age requirements in e-commerce.

Age restriction in E-commerce

Problem:

Verification of minimum age requirements in e-commerce.

Common solutions:

1. ID Verification
2. Restricted Accounts
3. Attribute-based

Age restriction in E-commerce

Problem:

Verification of minimum age requirements in e-commerce.

Common solutions:

	Privacy
1. ID Verification	bad
2. Restricted Accounts	bad
3. Attribute-based	good

Age restriction in E-commerce

Problem:

Verification of minimum age requirements in e-commerce.

Common solutions:

	Privacy	Ext. authority
1. ID Verification	bad	required
2. Restricted Accounts	bad	required
3. Attribute-based	good	required

Age restriction in E-commerce

Problem:

Verification of minimum age requirements in e-commerce.

Common solutions:

	Privacy	Ext. authority
1. ID Verification	bad	required
2. Restricted Accounts	bad	required
3. Attribute-based	good	required

Principle of subsidiarity is ignored

Principle of Subsidiarity

Functions of government
—such as granting and restricting rights—
should be performed
at the lowest level of authority possible,
as long as they can be performed *adequately*.

Principle of Subsidiarity

Functions of government
—such as granting and restricting rights—
should be performed
at the lowest level of authority possible,
as long as they can be performed *adequately*.

For age-restriction, the lowest level of authority is:

Parents, guardians and caretakers

Our goal

A design and implementation of an age restriction scheme with the following properties:

Our goal

A design and implementation of an age restriction scheme with the following properties:

1. It ties age restriction to the **ability to pay** (not to ID's),

Our goal

A design and implementation of an age restriction scheme with the following properties:

1. It ties age restriction to the **ability to pay** (not to ID's),
2. maintains the **anonymity of buyers**,

Our goal

A design and implementation of an age restriction scheme with the following properties:

1. It ties age restriction to the **ability to pay** (not to ID's),
2. maintains the **anonymity of buyers**,
3. maintains **unlinkability of transactions**,

Our goal

A design and implementation of an age restriction scheme with the following properties:

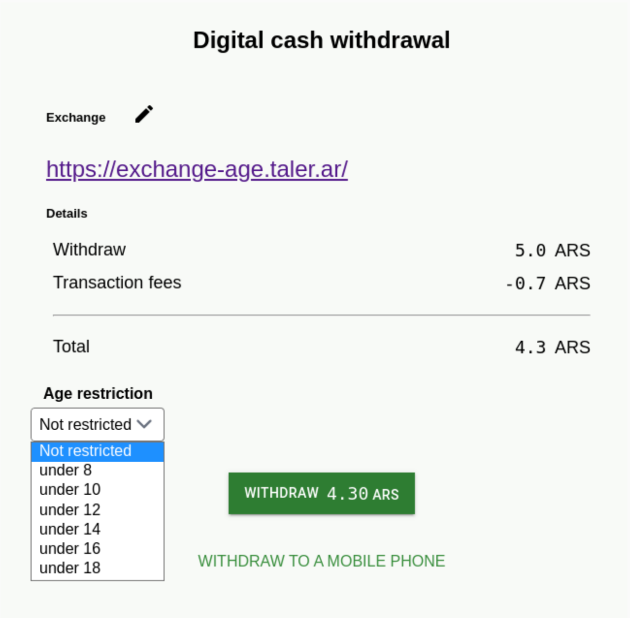
1. It ties age restriction to the **ability to pay** (not to ID's),
2. maintains the **anonymity of buyers**,
3. maintains **unlinkability of transactions**,
4. aligns with the **principle of subsidiarity**,

Our goal

A design and implementation of an age restriction scheme with the following properties:

1. It ties age restriction to the **ability to pay** (not to ID's),
2. maintains the **anonymity of buyers**,
3. maintains **unlinkability of transactions**,
4. aligns with the **principle of subsidiarity**,
5. is **practical and efficient**.

Teaser



Digital cash withdrawal
Exchange
https://exchange-age.taler.ar/
Details
Withdraw 5.0 ARS
Transaction fees -0.7 ARS
Total 4.3 ARS
Age restriction
Not restricted
Not restricted
under 8
under 10
under 12
under 14
under 16
under 18
WITHDRAW 4.30 ARS
WITHDRAW TO A MOBILE PHONE

The quest for a solution to age restriction

A journey through cryptic territory

Basic assumption and ideas

Assumption: Bank accounts are under control of adults/guardians.

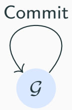
Sketch of scheme, independent of payment service protocol:

Basic assumption and ideas

Assumption: Bank accounts are under control of adults/guardians.

Sketch of scheme, independent of payment service protocol:

1. *Guardians* **commit** to a maximum age

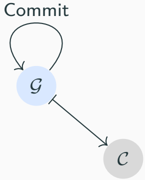


Basic assumption and ideas

Assumption: Bank accounts are under control of adults/guardians.

Sketch of scheme, independent of payment service protocol:

1. *Guardians* **commit** to a maximum age

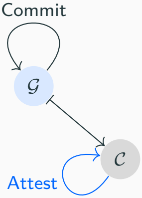


Basic assumption and ideas

Assumption: Bank accounts are under control of adults/guardians.

Sketch of scheme, independent of payment service protocol:

1. *Guardians* **commit** to a maximum age
2. *Minors* **attest** their adequate age

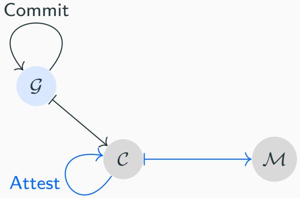


Basic assumption and ideas

Assumption: Bank accounts are under control of adults/guardians.

Sketch of scheme, independent of payment service protocol:

1. *Guardians* **commit** to a maximum age
2. *Minors* **attest** their adequate age

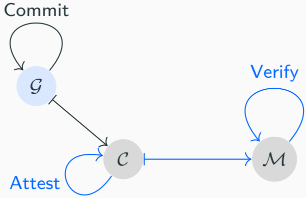


Basic assumption and ideas

Assumption: Bank accounts are under control of adults/guardians.

Sketch of scheme, independent of payment service protocol:

1. *Guardians* **commit** to a maximum age
2. *Minors* **attest** their adequate age
3. *Merchants* **verify** the attestations

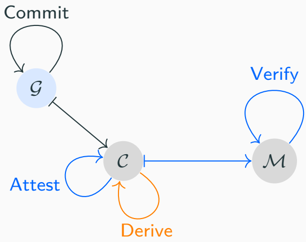


Basic assumption and ideas

Assumption: Bank accounts are under control of adults/guardians.

Sketch of scheme, independent of payment service protocol:

1. *Guardians* **commit** to a maximum age
2. *Minors* **attest** their adequate age
3. *Merchants* **verify** the attestations
4. *Minors* **derive** age commitments from existing ones

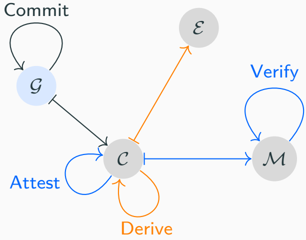


Basic assumption and ideas

Assumption: Bank accounts are under control of adults/guardians.

Sketch of scheme, independent of payment service protocol:

1. *Guardians* **commit** to a maximum age
2. *Minors* **attest** their adequate age
3. *Merchants* **verify** the attestations
4. *Minors* **derive** age commitments from existing ones

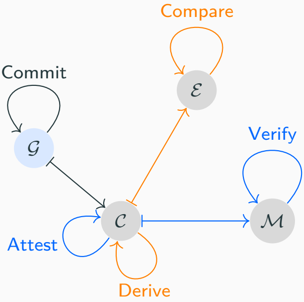


Basic assumption and ideas

Assumption: Bank accounts are under control of adults/guardians.

Sketch of scheme, independent of payment service protocol:

1. *Guardians* **commit** to a maximum age
2. *Minors* **attest** their adequate age
3. *Merchants* **verify** the attestations
4. *Minors* **derive** age commitments from existing ones
5. *Exchanges* **compare** the derived age commitments

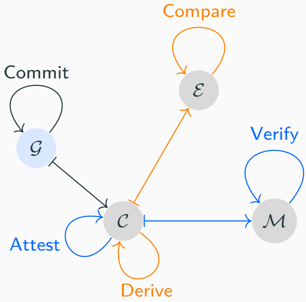


Basic assumption and ideas

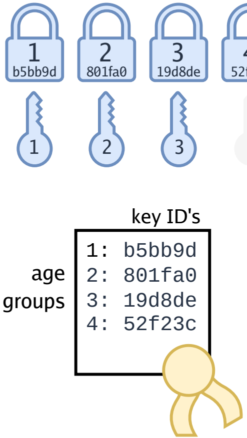
Assumption: Bank accounts are under control of adults/guardians.

Sketch of scheme, independent of payment service protocol:

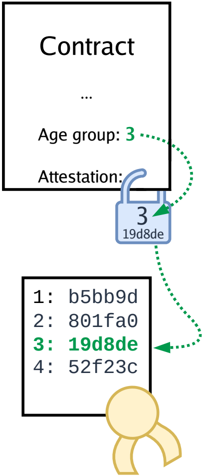
1. *Guardians* **commit** to a maximum age
2. *Minors* **attest** their adequate age
3. *Merchants* **verify** the attestations
4. *Minors* **derive** age commitments from existing ones
5. *Exchanges* **compare** the derived age commitments
6. `GOTO` 2.



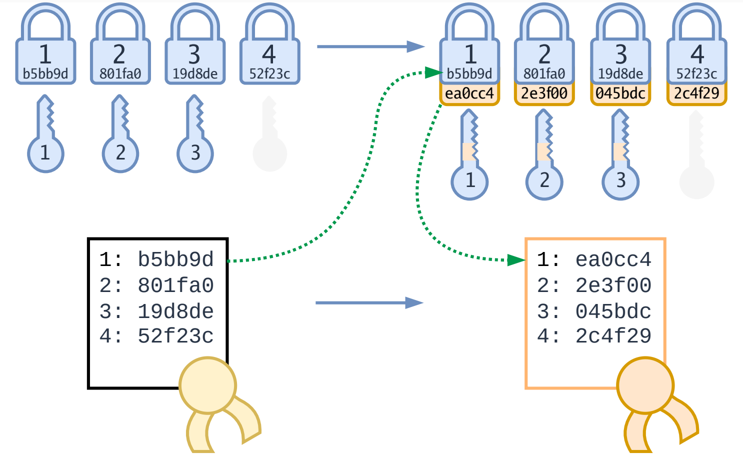
Helpful figure - Commit



Helpful figure - Attest and Verify

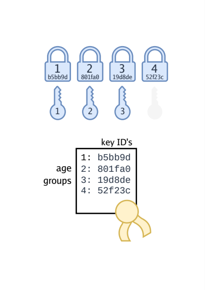


Helpful figure - Derive and Compare

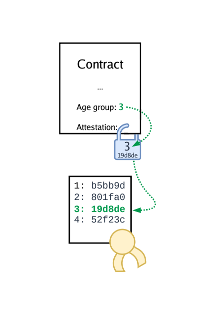


Helpful figure

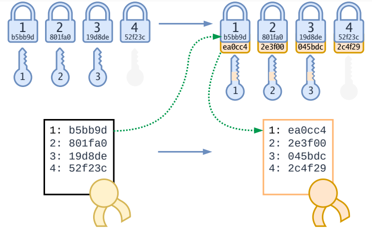
Commit:



Attest and Verify:



Derive and Compare:



Specification of the Function Signatures

Searching for functions

- Commit
- Attest
- Verify
- Derive
- Compare

Specification of the Function Signatures

Searching for functions with the following signatures

$$
\begin{aligned}
&\mathsf{Commit}: && (\mathsf{a},\omega) \mapsto (\mathsf{Q},\mathsf{P}) && \mathbb{N}_{\mathsf{M}} \times \Omega \to \mathbb{O} \times \mathbb{P}, \\
&\mathsf{Attest} \\
&\mathsf{Verify} \\
&\mathsf{Derive} \\
&\mathsf{Compare}
\end{aligned}
$$

Mnemonics:
$\mathbb{O}$ = *c$\mathbb{O}$mmitments*, Q = *Q-mitment* (commitment), $\mathbb{P}$ = *$\mathbb{P}$roofs*,

Specification of the Function Signatures

Searching for functions with the following signatures

$$\begin{aligned} \mathsf{Commit}: \quad & (\mathsf{a}, \omega) \mapsto (\mathsf{Q}, \mathsf{P}) && \mathbb{N}_{\mathsf{M}} \times \Omega \to \mathbb{O} \times \mathbb{P}, \\ \mathsf{Attest}: \quad & (\mathsf{m}, \mathsf{P}) \mapsto \mathsf{T} && \mathbb{N}_{\mathsf{M}} \times \mathbb{P} \to \mathbb{T} \cup \{\bot\}, \\ \mathsf{Verify} \quad & && \\ \mathsf{Derive} \quad & && \\ \mathsf{Compare} \quad & && \end{aligned}$$

Mnemonics:
$\mathbb{O} =$ *c$\mathbb{O}$mmitments*, $\mathsf{Q} =$ *Q-mitment* (commitment), $\mathbb{P} = \mathbb{P}$*roofs*, $\mathsf{P} = \mathsf{P}$*roof*,
$\mathbb{T} =$ *a$\mathbb{T}$testations*, $\mathsf{T} =$ *aTtestation*,

Specification of the Function Signatures

Searching for functions with the following signatures

$$
\begin{aligned}
&\mathsf{Commit}: &(\mathsf{a}, \omega) \mapsto (\mathsf{Q}, \mathsf{P}) \qquad & \mathbb{N}_\mathsf{M} \times \Omega \to \mathbb{O} \times \mathbb{P}, \\
&\mathsf{Attest}: &(\mathsf{m}, \mathsf{P}) \mapsto \mathsf{T} \qquad & \mathbb{N}_\mathsf{M} \times \mathbb{P} \to \mathbb{T} \cup \{\bot\}, \\
&\mathsf{Verify}: &(\mathsf{m}, \mathsf{Q}, \mathsf{T}) \mapsto b \qquad & \mathbb{N}_\mathsf{M} \times \mathbb{O} \times \mathbb{T} \to \mathbb{Z}_2, \\
&\mathsf{Derive} & & \\
&\mathsf{Compare} & &
\end{aligned}
$$

Mnemonics:
$\mathbb{O}$ = *c$\mathbb{O}$mmitments*, Q = *Q-mitment* (commitment), $\mathbb{P}$ = $\mathbb{P}$*roofs*, P = P*roof*,
$\mathbb{T}$ = *a$\mathbb{T}$testations*, T = *a*T*testation*,

Specification of the Function Signatures

Searching for functions with the following signatures

$$\begin{aligned}
&\mathsf{Commit}: && (\mathsf{a}, \omega) \mapsto (\mathsf{Q}, \mathsf{P}) && \mathbb{N}_{\mathsf{M}} \times \Omega \to \mathbb{O} \times \mathbb{P}, \\
&\mathsf{Attest}: && (\mathsf{m}, \mathsf{P}) \mapsto \mathsf{T} && \mathbb{N}_{\mathsf{M}} \times \mathbb{P} \to \mathbb{T} \cup \{\bot\}, \\
&\mathsf{Verify}: && (\mathsf{m}, \mathsf{Q}, \mathsf{T}) \mapsto b && \mathbb{N}_{\mathsf{M}} \times \mathbb{O} \times \mathbb{T} \to \mathbb{Z}_2, \\
&\mathsf{Derive}: && (\mathsf{Q}, \mathsf{P}, \omega) \mapsto (\mathsf{Q}', \mathsf{P}', \beta) && \mathbb{O} \times \mathbb{P} \times \Omega \to \mathbb{O} \times \mathbb{P} \times \mathbb{B}, \\
&\mathsf{Compare}
\end{aligned}$$

Mnemonics:
$\mathbb{O}$ = *c$\mathbb{O}$mmitments*, Q = *Q-mitment* (commitment), $\mathbb{P}$ = $\mathbb{P}$*roofs*, P = P*roof*,
$\mathbb{T}$ = *a$\mathbb{T}$testations*, T = *aTtestation*, $\mathbb{B}$ = $\mathbb{B}$*lindings*, β = β*linding*.

Specification of the Function Signatures

Searching for functions with the following signatures

$$\begin{aligned}
\mathsf{Commit}: &\quad (\mathsf{a}, \omega) \mapsto (\mathsf{Q}, \mathsf{P}) && \mathbb{N}_\mathsf{M} \times \Omega \to \mathbb{O} \times \mathbb{P}, \\
\mathsf{Attest}: &\quad (\mathsf{m}, \mathsf{P}) \mapsto \mathsf{T} && \mathbb{N}_\mathsf{M} \times \mathbb{P} \to \mathbb{T} \cup \{\bot\}, \\
\mathsf{Verify}: &\quad (\mathsf{m}, \mathsf{Q}, \mathsf{T}) \mapsto b && \mathbb{N}_\mathsf{M} \times \mathbb{O} \times \mathbb{T} \to \mathbb{Z}_2, \\
\mathsf{Derive}: &\quad (\mathsf{Q}, \mathsf{P}, \omega) \mapsto (\mathsf{Q}', \mathsf{P}', \beta) && \mathbb{O} \times \mathbb{P} \times \Omega \to \mathbb{O} \times \mathbb{P} \times \mathbb{B}, \\
\mathsf{Compare}: &\quad (\mathsf{Q}, \mathsf{Q}', \beta) \mapsto b && \mathbb{O} \times \mathbb{O} \times \mathbb{B} \to \mathbb{Z}_2,
\end{aligned}$$

Mnemonics:
$\mathbb{O}$ = *c$\mathbb{O}$mmitments*, Q = *Q-mitment* (commitment), $\mathbb{P}$ = $\mathbb{P}$*roofs*, P = P*roof*,
$\mathbb{T}$ = *a$\mathbb{T}$testations*, T = *a*T*testation*, $\mathbb{B}$ = $\mathbb{B}$*lindings*, β = β*linding*.

Specification of the Function Signatures

Searching for functions with the following signatures

$$
\begin{aligned}
\text{Commit} : \quad & (\mathsf{a}, \omega) \mapsto (\mathsf{Q}, \mathsf{P}) && \mathbb{N}_\mathsf{M} \times \Omega \to \mathbb{O} \times \mathbb{P}, \\
\text{Attest} : \quad & (\mathsf{m}, \mathsf{P}) \mapsto \mathsf{T} && \mathbb{N}_\mathsf{M} \times \mathbb{P} \to \mathbb{T} \cup \{\bot\}, \\
\text{Verify} : \quad & (\mathsf{m}, \mathsf{Q}, \mathsf{T}) \mapsto b && \mathbb{N}_\mathsf{M} \times \mathbb{O} \times \mathbb{T} \to \mathbb{Z}_2, \\
\text{Derive} : \quad & (\mathsf{Q}, \mathsf{P}, \omega) \mapsto (\mathsf{Q}', \mathsf{P}', \beta) && \mathbb{O} \times \mathbb{P} \times \Omega \to \mathbb{O} \times \mathbb{P} \times \mathbb{B}, \\
\text{Compare} : \quad & (\mathsf{Q}, \mathsf{Q}', \beta) \mapsto b && \mathbb{O} \times \mathbb{O} \times \mathbb{B} \to \mathbb{Z}_2,
\end{aligned}
$$

with $\Omega, \mathbb{P}, \mathbb{O}, \mathbb{T}, \mathbb{B}$ sufficiently large sets.

Mnemonics:
$\mathbb{O} =$ *c$\mathbb{O}$mmitments*, $\mathsf{Q} =$ *Q-mitment* (commitment), $\mathbb{P} =$ $\mathbb{P}$*roofs*, $\mathsf{P} =$ P*roof*,
$\mathbb{T} =$ *a$\mathbb{T}$testations*, $\mathsf{T} =$ *a*T*testation*, $\mathbb{B} =$ $\mathbb{B}$*lindings*, $\beta =$ β*linding*.

Specification of the Function Signatures

Searching for functions with the following signatures

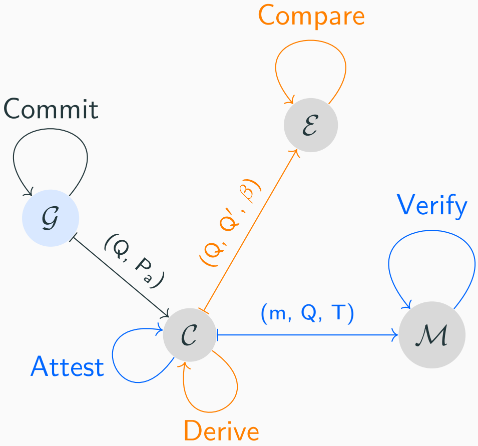
$$\begin{aligned}
\text{Commit}: &\quad (\mathsf{a}, \omega) \mapsto (\mathsf{Q}, \mathsf{P}) && \mathbb{N}_\mathsf{M} \times \Omega \to \mathbb{O} \times \mathbb{P}, \\
\text{Attest}: &\quad (\mathsf{m}, \mathsf{P}) \mapsto \mathsf{T} && \mathbb{N}_\mathsf{M} \times \mathbb{P} \to \mathbb{T} \cup \{\bot\}, \\
\text{Verify}: &\quad (\mathsf{m}, \mathsf{Q}, \mathsf{T}) \mapsto b && \mathbb{N}_\mathsf{M} \times \mathbb{O} \times \mathbb{T} \to \mathbb{Z}_2, \\
\text{Derive}: &\quad (\mathsf{Q}, \mathsf{P}, \omega) \mapsto (\mathsf{Q}', \mathsf{P}', \beta) && \mathbb{O} \times \mathbb{P} \times \Omega \to \mathbb{O} \times \mathbb{P} \times \mathbb{B}, \\
\text{Compare}: &\quad (\mathsf{Q}, \mathsf{Q}', \beta) \mapsto b && \mathbb{O} \times \mathbb{O} \times \mathbb{B} \to \mathbb{Z}_2,
\end{aligned}$$

with $\Omega, \mathbb{P}, \mathbb{O}, \mathbb{T}, \mathbb{B}$ sufficiently large sets.

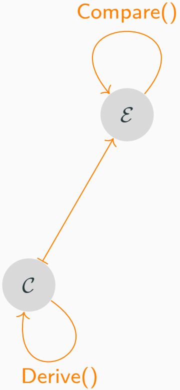
We will define basic and security requirements later.

Mnemonics:
$\mathbb{O} =$ *c$\mathbb{O}$mmitments*, $\mathsf{Q} =$ *Q-mitment* (commitment), $\mathbb{P} = \mathbb{P}$*roofs*, $\mathsf{P} = \mathsf{P}$*roof*,
$\mathbb{T} =$ *a$\mathbb{T}$testations*, $\mathsf{T} =$ *aTtestation*, $\mathbb{B} = \mathbb{B}$*lindings*, $\beta = \beta$*linding*.

Naïve scheme

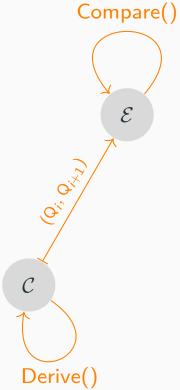


Problem of unlinkability



Simple use of Derive() and Compare() is problematic.

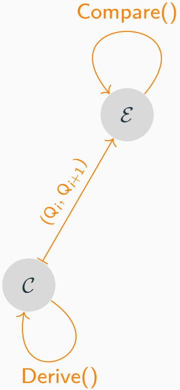
Problem of unlinkability



Simple use of Derive() and Compare() is problematic.

- Calling Derive() iteratively generates sequence $(Q_0, Q_1, \ldots)$ of commitments.

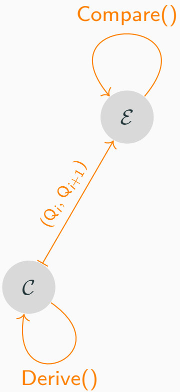
Problem of unlinkability



Simple use of Derive() and Compare() is problematic.

- Calling Derive() iteratively generates sequence $(Q_0, Q_1, \ldots)$ of commitments.
- Exchange calls Compare$(Q_i, Q_{i+1}, \ .\ )$

Problem of unlinkability

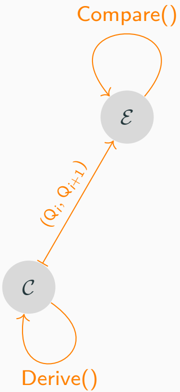


Simple use of Derive() and Compare() is problematic.

- Calling Derive() iteratively generates sequence $(Q_0, Q_1, \ldots)$ of commitments.
- Exchange calls Compare$(Q_i, Q_{i+1},\ .\ )$

$\implies$ Exchange identifies sequence

Problem of unlinkability



Simple use of Derive() and Compare() is problematic.

- Calling Derive() iteratively generates sequence $(Q_0, Q_1, \ldots)$ of commitments.
- Exchange calls Compare$(Q_i, Q_{i+1}, \ . \ )$

$\Longrightarrow$ Exchange identifies sequence

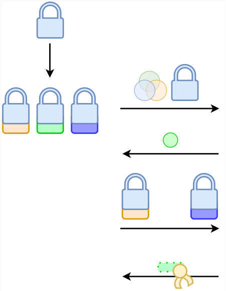
$\Longrightarrow$ **Unlinkability broken**

Achieving Unlinkability

Given Derive() and Compare(), define the cut&choose protocol DeriveCompare$_{\kappa}$ as follows (sketch):

Achieving Unlinkability

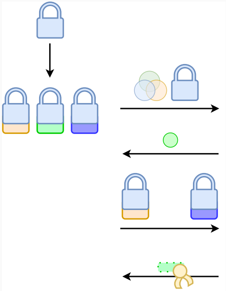
Given Derive() and Compare(), define the cut&choose protocol DeriveCompare$_\kappa$ as follows (sketch):



Achieving Unlinkability

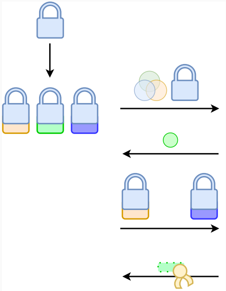
Given Derive() and Compare(), define the cut&choose protocol DeriveCompare$_\kappa$ as follows (sketch):

Let $\kappa \in \mathbb{N}$ (say: $\kappa = 3$)



Achieving Unlinkability

Given Derive() and Compare(), define the cut&choose protocol DeriveCompare$_\kappa$ as follows (sketch):

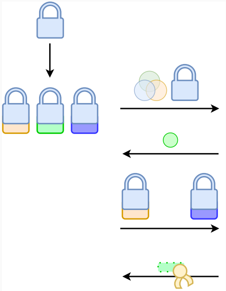


Let $\kappa \in \mathbb{N}$ (say: $\kappa = 3$)

$\mathcal{C}$: 1. generates $(\mathsf{Q}_1, \dots, \mathsf{Q}_\kappa)$ and $(\beta_1, \dots, \beta_\kappa)$ from Q_0 by calling κ times Derive$(\mathsf{Q}_0, \mathsf{P}_0, \omega_i)$

Achieving Unlinkability

Given Derive() and Compare(), define the cut&choose protocol DeriveCompare$_\kappa$ as follows (sketch):



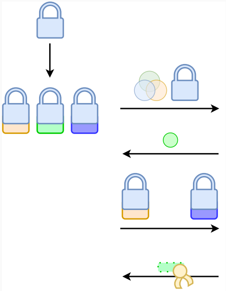
Let $\kappa \in \mathbb{N}$ (say: $\kappa = 3$)

$\mathcal{C}$:

1. generates $(\mathsf{Q}_1, \ldots, \mathsf{Q}_\kappa)$ and $(\beta_1, \ldots, \beta_\kappa)$ from Q_0 by calling κ times Derive$(\mathsf{Q}_0, \mathsf{P}_0, \omega_i)$
2. calculates $h_0 := H\big(H(\mathsf{Q}_1, \beta_1) \parallel \cdots \parallel H(\mathsf{Q}_\kappa, \beta_\kappa)\big)$

Achieving Unlinkability

Given Derive() and Compare(), define the cut&choose protocol DeriveCompare$_\kappa$ as follows (sketch):



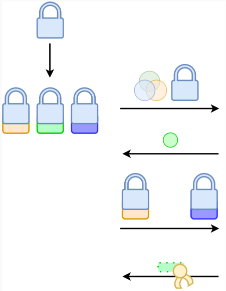
Let $\kappa \in \mathbb{N}$ (say: $\kappa = 3$)

$\mathcal{C}$:

1. generates $(\mathsf{Q}_1, \dots, \mathsf{Q}_\kappa)$ and $(\beta_1, \dots, \beta_\kappa)$ from Q_0 by calling κ times $\mathsf{Derive}(\mathsf{Q}_0, \mathsf{P}_0, \omega_i)$
2. calculates
   $h_0 := H\big(H(\mathsf{Q}_1, \beta_1) \parallel \cdots \parallel H(\mathsf{Q}_\kappa, \beta_\kappa)\big)$
3. sends Q_0 and h_0 to $\mathcal{E}$

Achieving Unlinkability

Given Derive() and Compare(), define the cut&choose protocol DeriveCompare$_\kappa$ as follows (sketch):



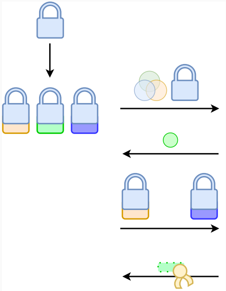
Let $\kappa \in \mathbb{N}$ (say: $\kappa = 3$)

$\mathcal{C}$:

1. generates $(\mathsf{Q}_1, \dots, \mathsf{Q}_\kappa)$ and $(\beta_1, \dots, \beta_\kappa)$ from Q_0 by calling κ times Derive$(\mathsf{Q}_0, \mathsf{P}_0, \omega_i)$
2. calculates $h_0 := H\big(H(\mathsf{Q}_1, \beta_1) \parallel \cdots \parallel H(\mathsf{Q}_\kappa, \beta_\kappa)\big)$
3. sends Q_0 and h_0 to $\mathcal{E}$

Achieving Unlinkability

Given Derive() and Compare(), define the cut&choose protocol DeriveCompare$_\kappa$ as follows (sketch):



Let $\kappa \in \mathbb{N}$ (say: $\kappa = 3$)

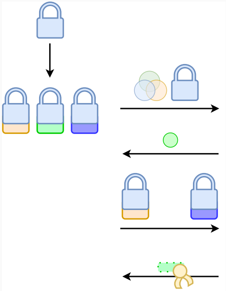
$\mathcal{C}$:
1. generates $(\mathsf{Q}_1, \ldots, \mathsf{Q}_\kappa)$ and $(\beta_1, \ldots, \beta_\kappa)$ from Q_0 by calling κ times Derive$(\mathsf{Q}_0, \mathsf{P}_0, \omega_i)$
2. calculates $h_0 := H\big(H(\mathsf{Q}_1, \beta_1) \parallel \cdots \parallel H(\mathsf{Q}_\kappa, \beta_\kappa)\big)$
3. sends Q_0 and h_0 to $\mathcal{E}$

$\mathcal{E}$:

4. saves Q_0 and h_0 and sends $\mathcal{C}$ random $\gamma \in \{1, \ldots, \kappa\}$

Achieving Unlinkability

Given Derive() and Compare(), define the cut&choose protocol DeriveCompare$_\kappa$ as follows (sketch):



Let $\kappa \in \mathbb{N}$ (say: $\kappa = 3$)

$\mathcal{C}$:

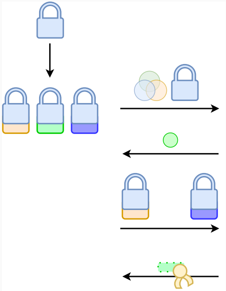
1. generates $(\mathsf{Q}_1, \dots, \mathsf{Q}_\kappa)$ and $(\beta_1, \dots, \beta_\kappa)$ from Q_0 by calling κ times Derive$(\mathsf{Q}_0, \mathsf{P}_0, \omega_i)$
2. calculates $h_0 := H\big(H(\mathsf{Q}_1, \beta_1) \parallel \dots \parallel H(\mathsf{Q}_\kappa, \beta_\kappa)\big)$
3. sends Q_0 and h_0 to $\mathcal{E}$

$\mathcal{E}$:

4. saves Q_0 and h_0 and sends $\mathcal{C}$ random $\gamma \in \{1, \dots, \kappa\}$

Achieving Unlinkability

Given Derive() and Compare(), define the cut&choose protocol DeriveCompare$_\kappa$ as follows (sketch):



Let $\kappa \in \mathbb{N}$ (say: $\kappa = 3$)

$\mathcal{C}$:

1. generates $(\mathsf{Q}_1, \dots, \mathsf{Q}_\kappa)$ and $(\beta_1, \dots, \beta_\kappa)$ from Q_0 by calling κ times Derive$(\mathsf{Q}_0, \mathsf{P}_0, \omega_i)$
2. calculates $h_0 := H(H(\mathsf{Q}_1, \beta_1) \parallel \cdots \parallel H(\mathsf{Q}_\kappa, \beta_\kappa))$
3. sends Q_0 and h_0 to $\mathcal{E}$

$\mathcal{E}$:

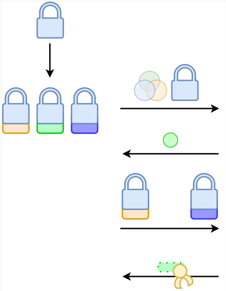
4. saves Q_0 and h_0 and sends $\mathcal{C}$ random $\gamma \in \{1, \dots, \kappa\}$

$\mathcal{C}$:

5. reveals $h_\gamma := H(\mathsf{Q}_\gamma, \beta_\gamma)$ and all (Q_i, β_i), except $(\mathsf{Q}_\gamma, \beta_\gamma)$

Achieving Unlinkability

Given Derive() and Compare(), define the cut&choose protocol DeriveCompare$_\kappa$ as follows (sketch):



Let $\kappa \in \mathbb{N}$ (say: $\kappa = 3$)

$\mathcal{C}$:
1. generates $(\mathsf{Q}_1, \dots, \mathsf{Q}_\kappa)$ and $(\beta_1, \dots, \beta_\kappa)$ from Q_0 by calling κ times Derive$(\mathsf{Q}_0, \mathsf{P}_0, \omega_i)$
2. calculates $h_0 := H\big(H(\mathsf{Q}_1, \beta_1) \parallel \cdots \parallel H(\mathsf{Q}_\kappa, \beta_\kappa)\big)$
3. sends Q_0 and h_0 to $\mathcal{E}$

$\mathcal{E}$:

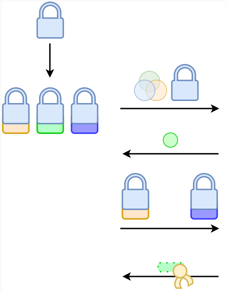
4. saves Q_0 and h_0 and sends $\mathcal{C}$ random $\gamma \in \{1, \dots, \kappa\}$

$\mathcal{C}$:

5. reveals $h_\gamma := H(\mathsf{Q}_\gamma, \beta_\gamma)$ and all (Q_i, β_i), except $(\mathsf{Q}_\gamma, \beta_\gamma)$

Achieving Unlinkability

Given Derive() and Compare(), define the cut&choose protocol DeriveCompare$_\kappa$ as follows (sketch):



Let $\kappa \in \mathbb{N}$ (say: $\kappa = 3$)

$\mathcal{C}$:
1. generates $(\mathsf{Q}_1, \dots, \mathsf{Q}_\kappa)$ and $(\beta_1, \dots, \beta_\kappa)$ from Q_0 by calling κ times Derive$(\mathsf{Q}_0, \mathsf{P}_0, \omega_i)$
2. calculates $h_0 := H\big(H(\mathsf{Q}_1, \beta_1) \parallel \cdots \parallel H(\mathsf{Q}_\kappa, \beta_\kappa)\big)$
3. sends Q_0 and h_0 to $\mathcal{E}$

$\mathcal{E}$:

4. saves Q_0 and h_0 and sends $\mathcal{C}$ random $\gamma \in \{1, \dots, \kappa\}$

$\mathcal{C}$:

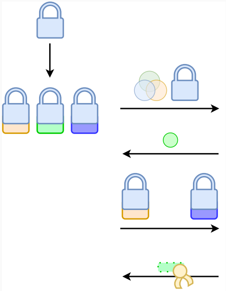
5. reveals $h_\gamma := H(\mathsf{Q}_\gamma, \beta_\gamma)$ and all (Q_i, β_i), except $(\mathsf{Q}_\gamma, \beta_\gamma)$

$\mathcal{E}$:

6. compares h_0 and $H\big(H(\mathsf{Q}_1, \beta_1) \parallel \dots \parallel h_\gamma \parallel \dots \parallel H(\mathsf{Q}_\kappa, \beta_\kappa)\big)$

Achieving Unlinkability

Given Derive() and Compare(), define the cut&choose protocol DeriveCompare$_\kappa$ as follows (sketch):



Let $\kappa \in \mathbb{N}$ (say: $\kappa = 3$)

$\mathcal{C}$:
1. generates $(\mathsf{Q}_1, \dots, \mathsf{Q}_\kappa)$ and $(\beta_1, \dots, \beta_\kappa)$ from Q_0 by calling κ times Derive$(\mathsf{Q}_0, \mathsf{P}_0, \omega_i)$
2. calculates $h_0 := H\big(H(\mathsf{Q}_1, \beta_1) \parallel \cdots \parallel H(\mathsf{Q}_\kappa, \beta_\kappa)\big)$
3. sends Q_0 and h_0 to $\mathcal{E}$

$\mathcal{E}$:

4. saves Q_0 and h_0 and sends $\mathcal{C}$ random $\gamma \in \{1, \dots, \kappa\}$

$\mathcal{C}$:

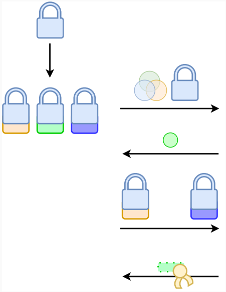
5. reveals $h_\gamma := H(\mathsf{Q}_\gamma, \beta_\gamma)$ and all (Q_i, β_i), except $(\mathsf{Q}_\gamma, \beta_\gamma)$

$\mathcal{E}$:

6. compares h_0 and $H\big(H(\mathsf{Q}_1, \beta_1) \parallel \dots \parallel h_\gamma \parallel \dots \parallel H(\mathsf{Q}_\kappa, \beta_\kappa)\big)$
7. evaluates Compare$(\mathsf{Q}_0, \mathsf{Q}_i, \beta_i)$ for all $i \neq \gamma$.

Achieving Unlinkability

Given Derive() and Compare(), define the cut&choose protocol DeriveCompare$_\kappa$ as follows (sketch):



Let $\kappa \in \mathbb{N}$ (say: $\kappa = 3$)

$\mathcal{C}$:

1. generates $(\mathsf{Q}_1, \ldots, \mathsf{Q}_\kappa)$ and $(\beta_1, \ldots, \beta_\kappa)$ from Q_0 by calling κ times Derive$(\mathsf{Q}_0, \mathsf{P}_0, \omega_i)$
2. calculates $h_0 := H\big(H(\mathsf{Q}_1, \beta_1) \parallel \cdots \parallel H(\mathsf{Q}_\kappa, \beta_\kappa)\big)$
3. sends Q_0 and h_0 to $\mathcal{E}$

$\mathcal{E}$:

4. saves Q_0 and h_0 and sends $\mathcal{C}$ random $\gamma \in \{1, \ldots, \kappa\}$

$\mathcal{C}$:

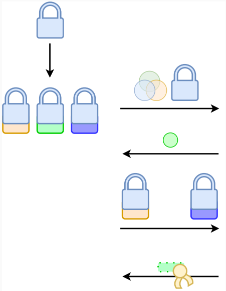
5. reveals $h_\gamma := H(\mathsf{Q}_\gamma, \beta_\gamma)$ and all (Q_i, β_i), except $(\mathsf{Q}_\gamma, \beta_\gamma)$

$\mathcal{E}$:

6. compares h_0 and $H\big(H(\mathsf{Q}_1, \beta_1) \parallel \ldots \parallel h_\gamma \parallel \ldots \parallel H(\mathsf{Q}_\kappa, \beta_\kappa)\big)$
7. evaluates Compare$(\mathsf{Q}_0, \mathsf{Q}_i, \beta_i)$ for all $i \neq \gamma$.

Achieving Unlinkability

Given Derive() and Compare(), define the cut&choose protocol DeriveCompare$_\kappa$ as follows (sketch):



Let $\kappa \in \mathbb{N}$ (say: $\kappa = 3$)

$\mathcal{C}$:

1. generates $(\mathsf{Q}_1, \ldots, \mathsf{Q}_\kappa)$ and $(\beta_1, \ldots, \beta_\kappa)$ from Q_0 by calling κ times Derive$(\mathsf{Q}_0, \mathsf{P}_0, \omega_i)$
2. calculates $h_0 := H(H(\mathsf{Q}_1, \beta_1) \parallel \cdots \parallel H(\mathsf{Q}_\kappa, \beta_\kappa))$
3. sends Q_0 and h_0 to $\mathcal{E}$

$\mathcal{E}$:

4. saves Q_0 and h_0 and sends $\mathcal{C}$ random $\gamma \in \{1, \ldots, \kappa\}$

$\mathcal{C}$:

5. reveals $h_\gamma := H(\mathsf{Q}_\gamma, \beta_\gamma)$ and all (Q_i, β_i), except $(\mathsf{Q}_\gamma, \beta_\gamma)$

$\mathcal{E}$:

6. compares h_0 and $H(H(\mathsf{Q}_1, \beta_1) \parallel \ldots \parallel h_\gamma \parallel \ldots \parallel H(\mathsf{Q}_\kappa, \beta_\kappa))$
7. evaluates Compare$(\mathsf{Q}_0, \mathsf{Q}_i, \beta_i)$ for all $i \neq \gamma$.

If all steps succeed, Q_γ is the new commitment.

Achieving Unlinkability

With DeriveCompare$_\kappa$

- $\mathcal{E}$ learns nothing about Q_γ or $H(\mathsf{Q}_\gamma)$,
- trusts outcome with $\frac{\kappa-1}{\kappa}$ certainty,
- i.e. $\mathcal{C}$ has $\frac{1}{\kappa}$ chance to cheat.

Achieving Unlinkability

With DeriveCompare$_\kappa$

- $\mathcal{E}$ learns nothing about Q_γ or $H(\mathsf{Q}_\gamma)$,
- trusts outcome with $\frac{\kappa-1}{\kappa}$ certainty,
- i.e. $\mathcal{C}$ has $\frac{1}{\kappa}$ chance to cheat.

$\Longrightarrow$ **Gives us unlinkability at the price of (adjustable) uncertainty!**

Achieving Unlinkability

With DeriveCompare$_\kappa$

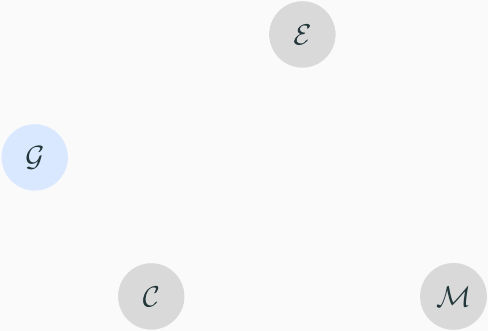
- $\mathcal{E}$ learns nothing about Q_γ or $H(\mathsf{Q}_\gamma)$,
- trusts outcome with $\frac{\kappa-1}{\kappa}$ certainty,
- i.e. $\mathcal{C}$ has $\frac{1}{\kappa}$ chance to cheat.

$\Longrightarrow$ **Gives us unlinkability at the price of (adjustable) uncertainty!**

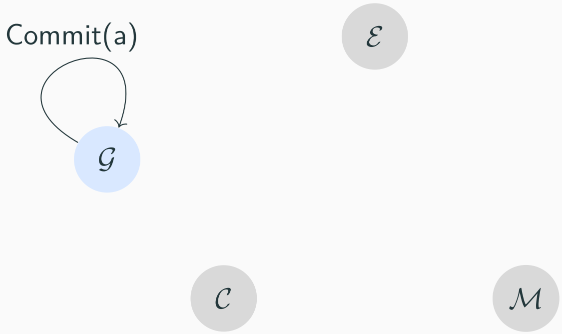
Notes:

- similar to the cut&choose *refresh* protocol in GNU Taler
- still need to define Derive() and Compare().

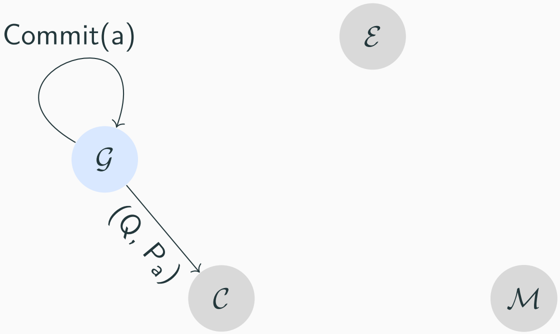
Refined scheme



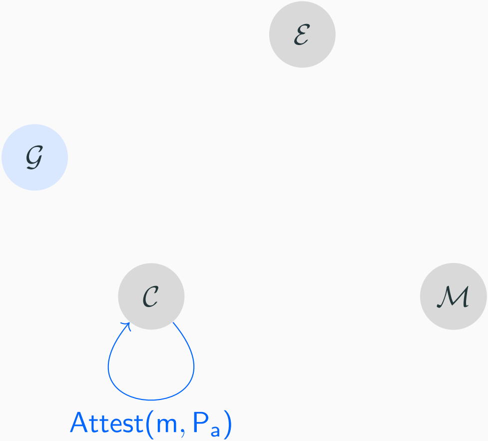
Refined scheme



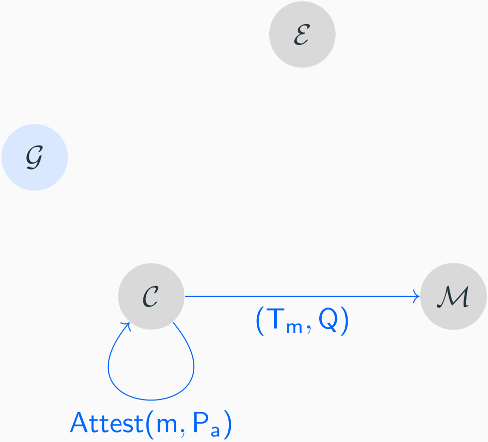
Refined scheme



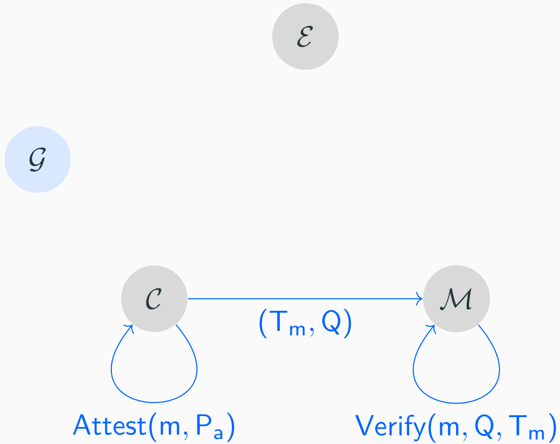
Refined scheme



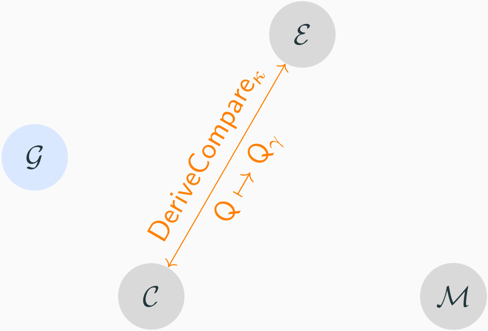
Refined scheme



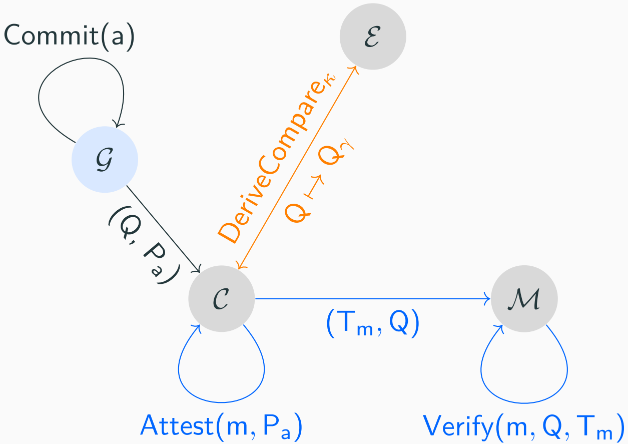
Refined scheme



Refined scheme



Refined scheme



Sensible solutions

Quest for functions should lead to *sensible* solutions.

Sensible solutions

Quest for functions should lead to *sensible* solutions.

F. e. Verify() should not simply always return `true`.

Sensible solutions

Quest for functions should lead to *sensible* solutions.

F. e. Verify() should not simply always return `true`.

We need more requirements.

Requirements

Basic Requirements

Candidate functions

$$(\mathrm{Commit}, \mathrm{Attest}, \mathrm{Verify}, \mathrm{Derive}, \mathrm{Compare})$$

must meet *basic requirements*:

- Existence of attestations
- Efficacy of attestations
- Derivability of commitments and attestations

Basic Requirements

Candidate functions

$$(\text{Commit}, \text{Attest}, \text{Verify}, \text{Derive}, \text{Compare})$$

must meet *basic requirements*:

- Existence of attestations
- Efficacy of attestations
- Derivability of commitments and attestations

More details in the published paper and Appendix.

Security Requirements

Candidate functions must also meet *security requirements*, defined via security games:

Security Requirements

Candidate functions must also meet *security requirements*, defined via security games:

Requirement: Unforgeability of minimum age

Security Requirements

Candidate functions must also meet *security requirements*, defined via security games:

Requirement:	Unforgeability of minimum age
$\leftrightarrow$ **Game:**	Forging an attestation

Security Requirements

Candidate functions must also meet *security requirements*, defined via security games:

Requirement: Unforgeability of minimum age
$\leftrightarrow$ **Game:** Forging an attestation

Requirement: Non-disclosure of age

Security Requirements

Candidate functions must also meet *security requirements*, defined via security games:

Requirement: Unforgeability of minimum age
$\leftrightarrow$ **Game:** Forging an attestation

Requirement: Non-disclosure of age
$\leftrightarrow$ **Game:** Age disclosure by commitment or attestation

Security Requirements

Candidate functions must also meet *security requirements*, defined via security games:

Requirement: Unforgeability of minimum age
$\leftrightarrow$ **Game:** Forging an attestation

Requirement: Non-disclosure of age
$\leftrightarrow$ **Game:** Age disclosure by commitment or attestation

Requirement: Unlinkability of commitments and attestations

Security Requirements

Candidate functions must also meet *security requirements*, defined via security games:

Requirement: Unforgeability of minimum age
$\leftrightarrow$ **Game:** Forging an attestation

Requirement: Non-disclosure of age
$\leftrightarrow$ **Game:** Age disclosure by commitment or attestation

Requirement: Unlinkability of commitments and attestations
$\leftrightarrow$ **Game:** Distinguishing derived commitments and attestations

Security Requirements

Candidate functions must also meet *security requirements*, defined via security games:

Requirement: Unforgeability of minimum age
$\leftrightarrow$ **Game:** Forging an attestation

Requirement: Non-disclosure of age
$\leftrightarrow$ **Game:** Age disclosure by commitment or attestation

Requirement: Unlinkability of commitments and attestations
$\leftrightarrow$ **Game:** Distinguishing derived commitments and attestations

Meeting the security requirements means that adversaries can win those games only with negligible advantage.

Security Requirements

Candidate functions must also meet *security requirements*, defined via security games:

Requirement: Unforgeability of minimum age
$\leftrightarrow$ **Game:** Forging an attestation

Requirement: Non-disclosure of age
$\leftrightarrow$ **Game:** Age disclosure by commitment or attestation

Requirement: Unlinkability of commitments and attestations
$\leftrightarrow$ **Game:** Distinguishing derived commitments and attestations

Meeting the security requirements means that adversaries can win those games only with negligible advantage.

Adversaries are arbitrary polynomial-time algorithms, acting on all relevant input.

Security Requirements

Simplified Example

Game $G_{\mathcal{A}}^{\mathsf{FA}}$: Forging an attest

Security Requirements

Simplified Example

Game $G_{\mathcal{A}}^{\mathsf{FA}}$: Forging an attest

1. $(\mathsf{a}, \omega) \xleftarrow{\$} \mathbb{N}_{\mathsf{M}-1} \times \Omega$

Security Requirements

Simplified Example

Game $G^{\mathsf{FA}}_{\mathcal{A}}$: Forging an attest

1. $(\mathsf{a}, \omega) \xleftarrow{\$} \mathbb{N}_{\mathsf{M}-1} \times \Omega$
2. $(\mathsf{Q}, \mathsf{P}) \leftarrow \mathsf{Commit}(\mathsf{a}, \omega)$

Security Requirements

Simplified Example

Game $G_{\mathcal{A}}^{\mathsf{FA}}$: Forging an attest

1. $(\mathsf{a}, \omega) \xleftarrow{\$} \mathbb{N}_{\mathsf{M}-1} \times \Omega$
2. $(\mathsf{Q}, \mathsf{P}) \leftarrow \mathsf{Commit}(\mathsf{a}, \omega)$
3. $(\mathsf{m}, \mathsf{T}) \leftarrow \mathcal{A}(\mathsf{a}, \mathsf{Q}, \mathsf{P})$

Security Requirements

Simplified Example

Game $G_{\mathcal{A}}^{\mathsf{FA}}$: Forging an attest

1. $(\mathsf{a}, \omega) \xleftarrow{\$} \mathbb{N}_{\mathsf{M}-1} \times \Omega$
2. $(\mathsf{Q}, \mathsf{P}) \leftarrow \mathsf{Commit}(\mathsf{a}, \omega)$
3. $(\mathsf{m}, \mathsf{T}) \leftarrow \mathcal{A}(\mathsf{a}, \mathsf{Q}, \mathsf{P})$
4. Return 0 if $\mathsf{m} \leq \mathsf{a}$

Security Requirements

Simplified Example

Game $G_{\mathcal{A}}^{\mathsf{FA}}$: Forging an attest

1. $(\mathsf{a}, \omega) \xleftarrow{\$} \mathbb{N}_{\mathsf{M}-1} \times \Omega$
2. $(\mathsf{Q}, \mathsf{P}) \leftarrow \mathsf{Commit}(\mathsf{a}, \omega)$
3. $(\mathsf{m}, \mathsf{T}) \leftarrow \mathcal{A}(\mathsf{a}, \mathsf{Q}, \mathsf{P})$
4. Return 0 if $\mathsf{m} \leq \mathsf{a}$
5. Return $\mathsf{Verify}(\mathsf{m}, \mathsf{Q}, \mathsf{T})$

Security Requirements

Simplified Example

Game $G_{\mathcal{A}}^{\mathsf{FA}}$: Forging an attest

1. $(\mathsf{a}, \omega) \xleftarrow{\$} \mathbb{N}_{\mathsf{M}-1} \times \Omega$
2. $(\mathsf{Q}, \mathsf{P}) \leftarrow \mathsf{Commit}(\mathsf{a}, \omega)$
3. $(\mathsf{m}, \mathsf{T}) \leftarrow \mathcal{A}(\mathsf{a}, \mathsf{Q}, \mathsf{P})$
4. Return 0 if $\mathsf{m} \leq \mathsf{a}$
5. Return $\mathsf{Verify}(\mathsf{m}, \mathsf{Q}, \mathsf{T})$

Adversary $\mathcal{A}$ wins the game, if $G_{\mathcal{A}}^{\mathsf{FA}}$ returns 1.

Security Requirements

Simplified Example

Game $G_{\mathcal{A}}^{\mathsf{FA}}$: Forging an attest

1. $(\mathsf{a}, \omega) \xleftarrow{\$} \mathbb{N}_{\mathsf{M}-1} \times \Omega$
2. $(\mathsf{Q}, \mathsf{P}) \leftarrow \mathsf{Commit}(\mathsf{a}, \omega)$
3. $(\mathsf{m}, \mathsf{T}) \leftarrow \mathcal{A}(\mathsf{a}, \mathsf{Q}, \mathsf{P})$
4. Return 0 if $\mathsf{m} \leq \mathsf{a}$
5. Return $\mathsf{Verify}(\mathsf{m}, \mathsf{Q}, \mathsf{T})$

Adversary $\mathcal{A}$ wins the game, if $G_{\mathcal{A}}^{\mathsf{FA}}$ returns 1.

Requirement: Unforgeability of minimum age

$$\underset{\mathcal{A} \in \mathfrak{A}(\mathbb{N}_{\mathsf{M}} \times \mathbb{O} \times \mathbb{P} \to \mathbb{N}_{\mathsf{M}} \times \mathbb{T})}{\forall} : \Pr\Big[G_{\mathcal{A}}^{\mathsf{FA}} = 1\Big] \leq \epsilon$$

Our task

Finding functions

$$(\mathsf{Commit}, \mathsf{Attest}, \mathsf{Verify}, \mathsf{Derive}, \mathsf{Compare})$$

that meet the basic and security requirements.

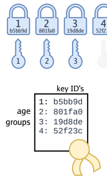
A solution to our quest

Instantiation with ECDSA

We propose a solution based on ECDSA.

Think: One key-pair per age group.

Definition of Commit with ECDSA

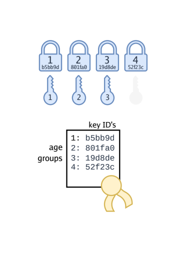


To Commit to age group $a \in \{1, \ldots, M\}$

1. Guardian generates ECDSA-keypairs, one per age group:

$$\langle (q_1, p_1), \ldots, (q_M, p_M) \rangle$$

Definition of Commit with ECDSA



To Commit to age group $\mathsf{a} \in \{1, \ldots, \mathsf{M}\}$

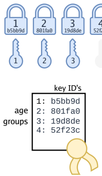
1. Guardian generates ECDSA-keypairs, one per age group:

$$\langle (q_1, p_1), \ldots, (q_\mathsf{M}, p_\mathsf{M}) \rangle$$

2. Guardian then **drops** all private keys p_i for $i > \mathsf{a}$:

$$\Big\langle (q_1, p_1), \ldots, (q_\mathsf{a}, p_\mathsf{a}), (q_{\mathsf{a}+1}, \perp), \ldots, (q_\mathsf{M}, \perp) \Big\rangle$$

Definition of Commit with ECDSA



To Commit to age group $\mathsf{a} \in \{1, \ldots, \mathsf{M}\}$

1. Guardian generates ECDSA-keypairs, one per age group:

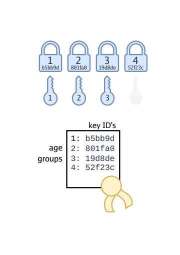
$$\langle (q_1, p_1), \ldots, (q_\mathsf{M}, p_\mathsf{M}) \rangle$$

2. Guardian then **drops** all private keys p_i for $i > \mathsf{a}$:

$$\Big\langle (q_1, p_1), \ldots, (q_\mathsf{a}, p_\mathsf{a}), (q_{\mathsf{a}+1}, \perp), \ldots, (q_\mathsf{M}, \perp) \Big\rangle$$

then set

Definition of Commit with ECDSA



To Commit to age group $a \in \{1, \dots, M\}$

1. Guardian generates ECDSA-keypairs, one per age group:

$$\langle (q_1, p_1), \dots, (q_M, p_M) \rangle$$

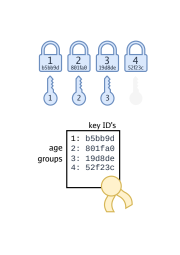
2. Guardian then **drops** all private keys p_i for $i > a$:

$$\left\langle (q_1, p_1), \dots, (q_a, p_a), (q_{a+1}, \perp), \dots, (q_M, \perp) \right\rangle$$

then set

Commitment: $\vec{Q} := (q_1, \ \dots \ \dots \ \dots \ , q_M)$

Definition of Commit with ECDSA



To Commit to age group $\mathsf{a} \in \{1, \ldots, \mathsf{M}\}$

1. Guardian generates ECDSA-keypairs, one per age group:

$$\langle (q_1, p_1), \ldots, (q_\mathsf{M}, p_\mathsf{M}) \rangle$$

2. Guardian then **drops** all private keys p_i for $i > \mathsf{a}$:

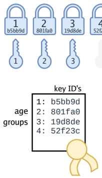
$$\Big\langle (q_1, p_1), \ldots, (q_\mathsf{a}, p_\mathsf{a}), (q_{\mathsf{a}+1}, \bot), \ldots, (q_\mathsf{M}, \bot) \Big\rangle$$

then set

Commitment: $\vec{\mathsf{Q}} := (q_1, \ \ldots \ \ldots \ \ldots \ , q_\mathsf{M})$

Proof: $\vec{\mathsf{P}}_\mathsf{a} := (p_1, \ldots, p_\mathsf{a}, \bot, \ldots, \bot)$

Definition of Commit with ECDSA



To Commit to age group $a \in \{1, \dots, M\}$

1. Guardian generates ECDSA-keypairs, one per age group:

$$\langle (q_1, p_1), \dots, (q_M, p_M) \rangle$$

2. Guardian then **drops** all private keys p_i for $i > a$:

$$\Big\langle (q_1, p_1), \dots, (q_a, p_a), (q_{a+1}, \perp), \dots, (q_M, \perp) \Big\rangle$$

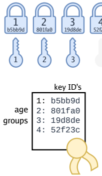
then set

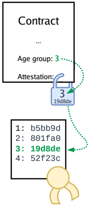
Commitment: $\vec{Q} := (q_1, \quad \dots \quad \dots \quad \dots \quad, q_M)$

Proof: $\vec{P}_a := (p_1, \dots, p_a, \perp, \dots, \perp)$

3. Guardian gives child $\langle \vec{Q}, \vec{P}_a \rangle$

Attest and Verify with ECDSA

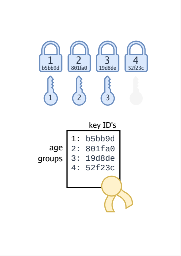


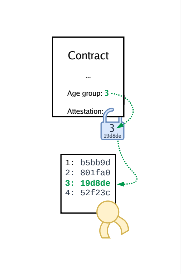


Child has

- ordered public-keys $\vec{Q} = (q_1, \ldots \quad \ldots \quad \ldots, q_M)$,
- (some) private-keys $\vec{P} = (p_1, \ldots, p_a, \perp, \ldots, \perp)$.

Attest and Verify with ECDSA





Child has

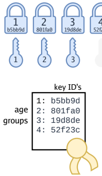
- ordered public-keys $\vec{Q} = (q_1, \dots \quad \dots \quad \dots, q_M)$,
- (some) private-keys $\vec{P} = (p_1, \dots, p_a, \perp, \dots, \perp)$.

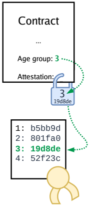
To Attest a minimum age (group) $m \leq a$:

Sign a message with ECDSA using private key p_m.

The signature σ_m is the attestation.

Attest and Verify with ECDSA





Child has

- ordered public-keys $\vec{Q} = (q_1, \ldots \quad \ldots \quad \ldots, q_M)$,
- (some) private-keys $\vec{P} = (p_1, \ldots, p_a, \perp, \ldots, \perp)$.

To Attest a minimum age (group) $m \leq a$:

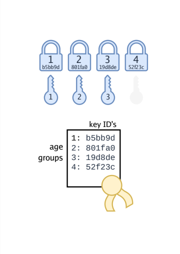
Sign a message with ECDSA using private key p_m.

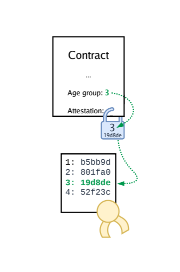
The signature σ_m is the attestation.

Merchant gets

- ordered public-keys $\vec{Q} = (q_1, \ldots, q_M)$
- Signature σ_m

Attest and Verify with ECDSA





Child has

- ordered public-keys $\vec{Q} = (q_1, \dots \quad \dots \quad \dots, q_M)$,
- (some) private-keys $\vec{P} = (p_1, \dots, p_a, \perp, \dots, \perp)$.

To Attest a minimum age (group) $m \leq a$**:**

Sign a message with ECDSA using private key p_m.

The signature σ_m is the attestation.

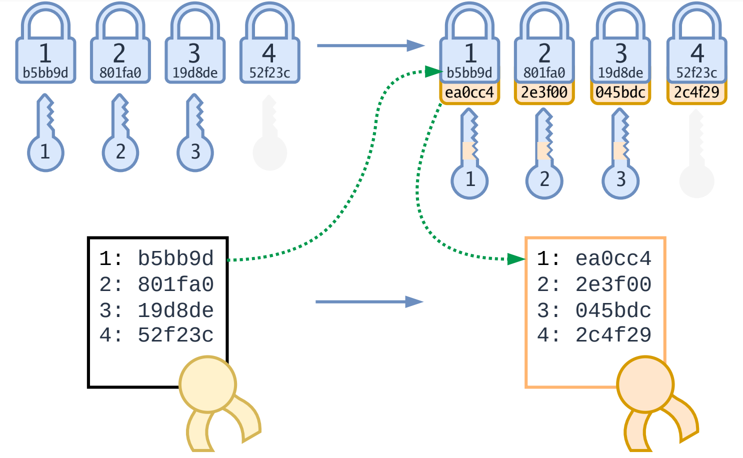
Merchant gets

- ordered public-keys $\vec{Q} = (q_1, \dots, q_M)$
- Signature σ_m

To Verify a minimum age (group) m**:**

Verify the ECDSA-Signature σ_m with public key q_m.

Reminder: Derive and Compare



Derive and Compare with ECDSA

Child has $\vec{\mathrm{Q}} = (q_1, \ldots, q_{\mathrm{M}})$ and $\vec{\mathrm{P}} = (p_1, \ldots, p_{\mathrm{a}}, \perp, \ldots, \perp)$.

Derive and Compare with ECDSA

Child has $\vec{\mathsf{Q}} = (q_1, \ldots, q_{\mathsf{M}})$ and $\vec{\mathsf{P}} = (p_1, \ldots, p_{\mathsf{a}}, \perp, \ldots, \perp)$.

To Derive new $\vec{\mathsf{Q}}'$ and $\vec{\mathsf{P}}'$: Choose random $\beta \in \mathbb{Z}_g$ and calculate

$$\begin{aligned} \vec{\mathsf{Q}}' &= (q'_1, \;\ldots\;\ldots\;\ldots\;, q'_{\mathsf{M}}) && := (\beta * q_1, \ldots\;\ldots, \beta * q_{\mathsf{M}}), \\ \vec{\mathsf{P}}' &= (p'_1, \ldots, p'_{\mathsf{a}}, \perp, \ldots, \perp) && := (\beta p_1, \ldots, \beta p_{\mathsf{a}}, \perp, \ldots, \perp) \end{aligned}$$

Derive and Compare with ECDSA

Child has $\vec{\mathsf{Q}} = (q_1, \ldots, q_{\mathsf{M}})$ and $\vec{\mathsf{P}} = (p_1, \ldots, p_{\mathsf{a}}, \perp, \ldots, \perp)$.

To Derive new $\vec{\mathsf{Q}}'$ and $\vec{\mathsf{P}}'$: Choose random $\beta \in \mathbb{Z}_g$ and calculate

$$\begin{aligned} \vec{\mathsf{Q}}' &= (q'_1, \; \ldots \; \ldots \; \ldots \; , q'_{\mathsf{M}}) && := (\beta * q_1, \ldots \; \ldots, \beta * q_{\mathsf{M}}), \\ \vec{\mathsf{P}}' &= (p'_1, \ldots, p'_{\mathsf{a}}, \perp, \ldots, \perp\,) && := (\beta p_1, \ldots, \beta p_{\mathsf{a}}, \perp, \ldots, \perp\,) \end{aligned}$$

Note:

- $\beta * q_i$ is scalar multiplication on the elliptic curve.
- $p'_i * G = (\beta p_i) * G = \beta * (p_i * G) = \beta * q_i = q'_i$

$\Longrightarrow$ p'_i **actually *is* private key to** q'_i

Derive and Compare with ECDSA

Child has $\vec{Q} = (q_1, \ldots, q_M)$ and $\vec{P} = (p_1, \ldots, p_a, \perp, \ldots, \perp)$.

To Derive new $\vec{Q}'$ and $\vec{P}'$: Choose random $\beta \in \mathbb{Z}_g$ and calculate

$$\begin{aligned}
\vec{Q}' &= (q'_1, \ \ldots \ \ldots \ \ldots \ , q'_M) &&:= (\beta * q_1, \ldots \ \ldots, \beta * q_M), \\
\vec{P}' &= (p'_1, \ldots, p'_a, \perp, \ldots, \perp\ ) &&:= (\beta p_1, \ldots, \beta p_a, \perp, \ldots, \perp\ )
\end{aligned}$$

Note:

- $\beta * q_i$ is scalar multiplication on the elliptic curve.
- $p'_i * G = (\beta p_i) * G = \beta * (p_i * G) = \beta * q_i = q'_i$

$\Longrightarrow$ p'_i **actually *is* private key to** q'_i

Exchange gets $\vec{Q} = (q_1, \ldots, q_M)$, $\vec{Q}' = (q'_1, \ldots, q'_M)$ and β

To Compare, calculate: $(\beta * q_1, \ldots, \beta * q_M) \stackrel{?}{=} (q'_1, \ldots, q'_M)$

Instantiation with ECDSA

Functions (Commit, Attest, Verify, Derive, Compare)
as defined in the instantiation with ECDSA

- meet the basic requirements,
- also meet all security requirements.

Security proofs by reduction, details are in the paper.

Example: Proof of Unforgeability

Proof by reduction:

Game $G^{\mathsf{FA}}_{\mathcal{A}}$: Forging an attest

1. $(\mathsf{a}, \omega) \xleftarrow{\$} \mathbb{N}_{\mathsf{M}-1} \times \Omega$
2. $(\mathsf{Q}, \mathsf{P}) \leftarrow \mathsf{Commit}(\mathsf{a}, \omega)$
3. $(\mathsf{m}, \mathsf{T}) \leftarrow \mathcal{A}(\mathsf{a}, \mathsf{Q}, \mathsf{P})$
4. Return 0 if $\mathsf{m} \leq \mathsf{a}$
5. Return $\mathsf{Verify}(\mathsf{m}, \mathsf{Q}, \mathsf{T})$

Requirement:

$$\underset{\mathcal{A}}{\forall} : \Pr\Big[G^{\mathsf{FA}}_{\mathcal{A}} = 1\Big] \leq \epsilon$$

Example: Proof of Unforgeability

Game $G_{\mathcal{A}}^{\mathsf{FA}}$: Forging an attest

1. $(\mathsf{a}, \omega) \xleftarrow{\$} \mathbb{N}_{\mathsf{M}-1} \times \Omega$
2. $(\mathsf{Q}, \mathsf{P}) \leftarrow \mathsf{Commit}(\mathsf{a}, \omega)$
3. $(\mathsf{m}, \mathsf{T}) \leftarrow \mathcal{A}(\mathsf{a}, \mathsf{Q}, \mathsf{P})$
4. Return 0 if $\mathsf{m} \leq \mathsf{a}$
5. Return $\mathsf{Verify}(\mathsf{m}, \mathsf{Q}, \mathsf{T})$

Requirement:

$$\underset{\mathcal{A}}{\forall} : \Pr\left[G_{\mathcal{A}}^{\mathsf{FA}} = 1\right] \leq \epsilon$$

Proof by reduction:

1. Adversary wins if $1 = \mathsf{Verify}(\mathsf{m}, \mathsf{Q}, \mathsf{T})$.

Example: Proof of Unforgeability

Game $G_{\mathcal{A}}^{\mathsf{FA}}$: Forging an attest

1. $(\mathsf{a}, \omega) \xleftarrow{\$} \mathbb{N}_{\mathsf{M}-1} \times \Omega$
2. $(\mathsf{Q}, \mathsf{P}) \leftarrow \mathsf{Commit}(\mathsf{a}, \omega)$
3. $(\mathsf{m}, \mathsf{T}) \leftarrow \mathcal{A}(\mathsf{a}, \mathsf{Q}, \mathsf{P})$
4. Return 0 if $\mathsf{m} \leq \mathsf{a}$
5. Return $\mathsf{Verify}(\mathsf{m}, \mathsf{Q}, \mathsf{T})$

Requirement:

$$\underset{\mathcal{A}}{\forall} : \Pr\left[G_{\mathcal{A}}^{\mathsf{FA}} = 1\right] \leq \epsilon$$

Proof by reduction:

1. Adversary wins if $1 = \mathsf{Verify}(\mathsf{m}, \mathsf{Q}, \mathsf{T})$.
2. That means: σ was a valid ECDSA-signature, validated with q_m.

Example: Proof of Unforgeability

Game $G_{\mathcal{A}}^{\mathsf{FA}}$: Forging an attest

1. $(\mathsf{a}, \omega) \xleftarrow{\$} \mathbb{N}_{\mathsf{M}-1} \times \Omega$
2. $(\mathsf{Q}, \mathsf{P}) \leftarrow \mathsf{Commit}(\mathsf{a}, \omega)$
3. $(\mathsf{m}, \mathsf{T}) \leftarrow \mathcal{A}(\mathsf{a}, \mathsf{Q}, \mathsf{P})$
4. Return 0 if $\mathsf{m} \leq \mathsf{a}$
5. Return $\mathsf{Verify}(\mathsf{m}, \mathsf{Q}, \mathsf{T})$

Requirement:

$$\underset{\mathcal{A}}{\forall} : \Pr\left[G_{\mathcal{A}}^{\mathsf{FA}} = 1\right] \leq \epsilon$$

Proof by reduction:

1. Adversary wins if $1 = \mathsf{Verify}(\mathsf{m}, \mathsf{Q}, \mathsf{T})$.
2. That means: σ was a valid ECDSA-signature, validated with q_m.
3. But adversary does not have the private key p_m to q_m.

Example: Proof of Unforgeability

Game $G_{\mathcal{A}}^{\mathsf{FA}}$: Forging an attest

1. $(\mathsf{a}, \omega) \xleftarrow{\$} \mathbb{N}_{\mathsf{M}-1} \times \Omega$
2. $(\mathsf{Q}, \mathsf{P}) \leftarrow \mathsf{Commit}(\mathsf{a}, \omega)$
3. $(\mathsf{m}, \mathsf{T}) \leftarrow \mathcal{A}(\mathsf{a}, \mathsf{Q}, \mathsf{P})$
4. Return 0 if $\mathsf{m} \leq \mathsf{a}$
5. Return $\mathsf{Verify}(\mathsf{m}, \mathsf{Q}, \mathsf{T})$

Requirement:

$$\underset{\mathcal{A}}{\forall} : \Pr\left[G_{\mathcal{A}}^{\mathsf{FA}} = 1\right] \leq \epsilon$$

Proof by reduction:

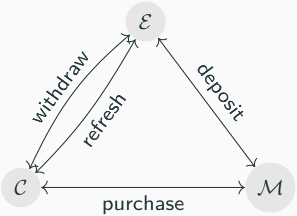
1. Adversary wins if $1 = \mathsf{Verify}(\mathsf{m}, \mathsf{Q}, \mathsf{T})$.
2. That means: σ was a valid ECDSA-signature, validated with q_m.
3. But adversary does not have the private key p_m to q_m.

$\Longrightarrow$ So winning this game would require to existentially forge the signature, which is negligible.

Integration with GNU Taler

GNU Taler

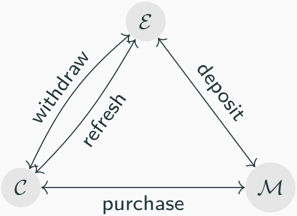
https://www.taler.net



- Protocol suite for online payment services
- Based on Chaum's blind signatures
- Taxable, efficient, free software
- Allows for change and refund
- Privacy preserving: anonymous and unlinkable payments

GNU Taler

https://www.taler.net



- Protocol suite for online payment services
- Based on Chaum's blind signatures
- Taxable, efficient, free software
- Allows for change and refund
- Privacy preserving: anonymous and unlinkable payments

- Coins are public-/private key-pairs (C_p, c_s).
- Exchange blindly signs $H(C_p)$ with denomination key d_p:

$$\beta(\sigma_p) = \text{BlindSign}\big(\beta\left(H(C_p)\right), d_p\big)$$

- Verification:

$$1 \quad \stackrel{?}{=} \quad \text{SigCheck}\big(H(C_p), D_p, \sigma_p\big)$$

(D_p = public key of denomination and σ_p = signature)

Integration with GNU Taler

Binding age restriction to coins

To bind an age commitment Q to a coin C_p, instead of blindly signing $H(C_p)$

$$\beta(\sigma_p) = \text{BlindSign}\big(\beta\,(H(C_p))\,, d_p\big)$$

$\mathcal{E}$ now blindly signs $H(C_p \parallel H(\text{Q}))$

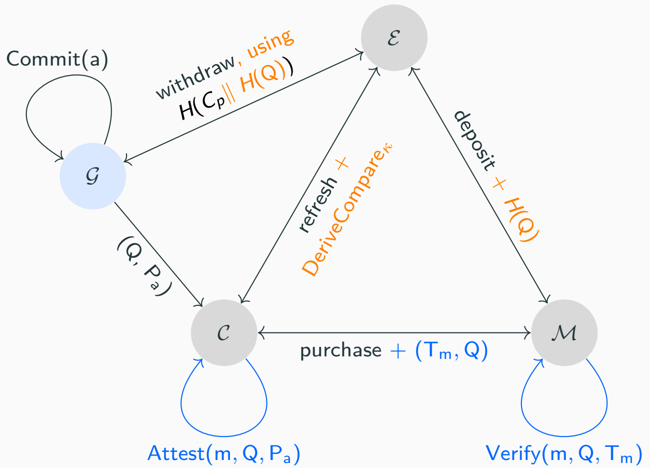
$$\beta(\sigma_p) = \text{BlindSign}\big(\beta\,(H(C_p \parallel H(\text{Q})))\,, d_p\big)$$

Therefore, verfication of a coin now requires $H(\text{Q})$, too:

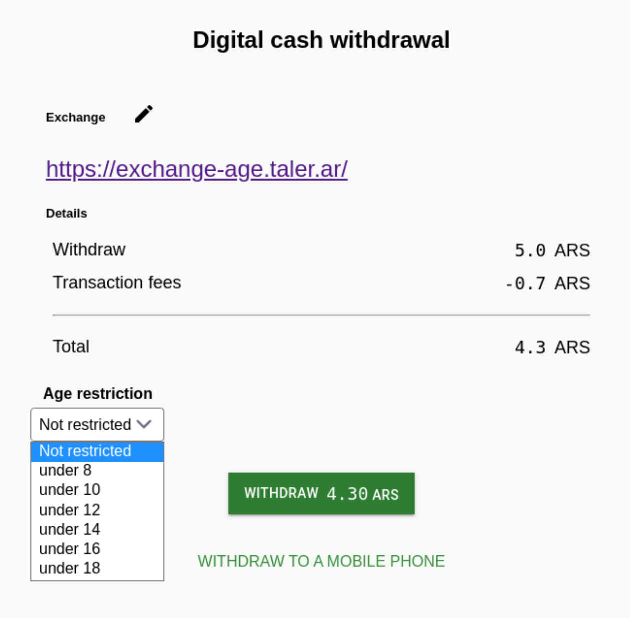
$$1 \stackrel{?}{=} \text{SigCheck}\big(H\,(C_p \parallel H(\text{Q}))\,, D_p, \sigma_p\big)$$

Integration with GNU Taler

Integrated schemes



Age restriction in the wallet



Discussion & Conclusion

Discussion

Technical aspects and challenges

- Our solution can in principle be used with any token-based payment scheme

Discussion

Technical aspects and challenges

- Our solution can in principle be used with any token-based payment scheme

  However, GNU Taler best aligned with our design goals (security, privacy and efficiency).

Discussion

Technical aspects and challenges

- Our solution can in principle be used with any token-based payment scheme

  However, GNU Taler best aligned with our design goals (security, privacy and efficiency).
- Subsidiarity requires bank accounts being owned by adults.

Discussion

Technical aspects and challenges

- Our solution can in principle be used with any token-based payment scheme

  However, GNU Taler best aligned with our design goals (security, privacy and efficiency).
- Subsidiarity requires bank accounts being owned by adults. However, scheme can be adapted

Discussion

Technical aspects and challenges

- Our solution can in principle be used with any token-based payment scheme

  However, GNU Taler best aligned with our design goals (security, privacy and efficiency).
- Subsidiarity requires bank accounts being owned by adults.

  However, scheme can be adapted
  - Know-Your-Customer (KYC) provides age information

Discussion

Technical aspects and challenges

- Our solution can in principle be used with any token-based payment scheme

  However, GNU Taler best aligned with our design goals (security, privacy and efficiency).
- Subsidiarity requires bank accounts being owned by adults.

  However, scheme can be adapted
  - Know-Your-Customer (KYC) provides age information
  - Parents can set age on a long-term wallet of a child

Discussion

Technical aspects and challenges

- Our solution can in principle be used with any token-based payment scheme

  However, GNU Taler best aligned with our design goals (security, privacy and efficiency).
- Subsidiarity requires bank accounts being owned by adults.

  However, scheme can be adapted
  - Know-Your-Customer (KYC) provides age information
  - Parents can set age on a long-term wallet of a child
  - cut&choose protocol `age-withdraw` implemented

Discussion

Legal aspects and applicability

- The scheme only makes sense when cheating can be discouraged, f.e. economically

Discussion

Legal aspects and applicability

- The scheme only makes sense when cheating can be discouraged, f.e. economically
- There will be limits where the scheme is considered acceptable.

Discussion

Legal aspects and applicability

- The scheme only makes sense when cheating can be discouraged, f.e. economically
- There will be limits where the scheme is considered acceptable.
- Our scheme offers an alternative to identity management systems (IMS), where applicable

Discussion

Potential for misuse

- Instead of age groups, couldn't the scheme encode *arbitrary* semantics?

Discussion

Potential for misuse

- Instead of age groups, couldn't the scheme encode *arbitrary* semantics?
- Yes, but *implementation* in GNU Taler only allows for arithmetic comparison.

Discussion

Potential for misuse

- Instead of age groups, couldn't the scheme encode *arbitrary* semantics?
- Yes, but *implementation* in GNU Taler only allows for arithmetic comparison.
- Note that augmented coin material (coin + age commitment) remains fungible.

Discussion

Potential for misuse

- Instead of age groups, couldn't the scheme encode *arbitrary* semantics?
- Yes, but *implementation* in GNU Taler only allows for arithmetic comparison.
- Note that augmented coin material (coin + age commitment) remains fungible.

- Problem of tainting already exists with denomitations keys:

Discussion

Potential for misuse

- Instead of age groups, couldn't the scheme encode *arbitrary* semantics?
- Yes, but *implementation* in GNU Taler only allows for arithmetic comparison.
- Note that augmented coin material (coin + age commitment) remains fungible.

- Problem of tainting already exists with denomitations keys:
- They could be used to seperate the anonymity set of users.

Discussion

Potential for misuse

- Instead of age groups, couldn't the scheme encode *arbitrary* semantics?
- Yes, but *implementation* in GNU Taler only allows for arithmetic comparison.
- Note that augmented coin material (coin + age commitment) remains fungible.

- Problem of tainting already exists with denomitations keys:
- They could be used to seperate the anonymity set of users.

- GNU Taler defines the role of an *Auditor*:

Discussion

Potential for misuse

- Instead of age groups, couldn't the scheme encode *arbitrary* semantics?
- Yes, but *implementation* in GNU Taler only allows for arithmetic comparison.
- Note that augmented coin material (coin + age commitment) remains fungible.

- Problem of tainting already exists with denomitations keys:
- They could be used to seperate the anonymity set of users.

- GNU Taler defines the role of an *Auditor*:
- a seperate entity to supervise the operation of the exchange.

Conclusion

Age restriction is a technical, ethical and legal challenge.

Conclusion

Age restriction is a technical, ethical and legal challenge.

Existing solutions are

- without strong protection of privacy or
- based on identity management systems (IMS)

Conclusion

Age restriction is a technical, ethical and legal challenge.

Existing solutions are

- without strong protection of privacy or
- based on identity management systems (IMS)

Our scheme offers an option that

- aligns with subsidiarity
- preserves privacy
- is efficient
- and an alternative to IMS

Thank you!

Questions?

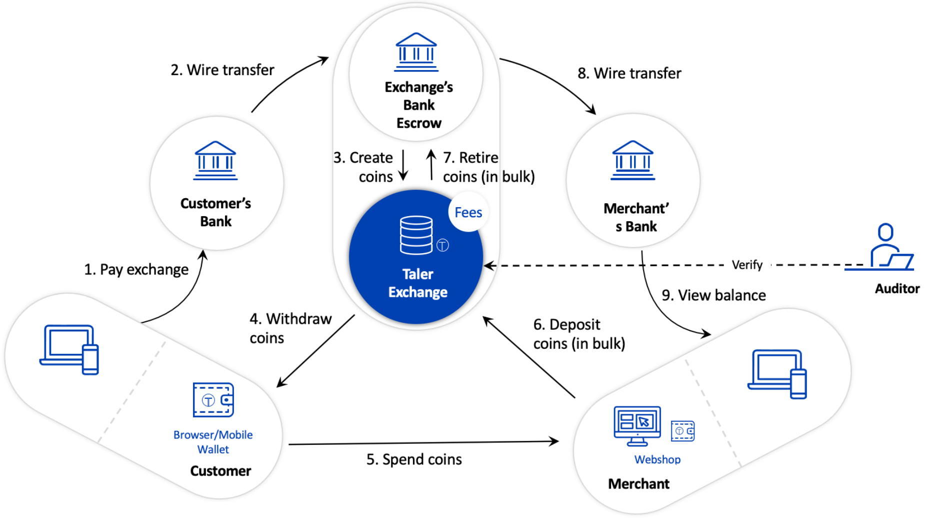
`oec-taler@kesim.org`

`@oec@mathstodon.xyz`

Interested in GNU Taler?

Intro:	`https://taler.net`
Learn:	`https://docs.taler.net`
Develop:	`https://git.taler.net`, `https://bugs.taler.net`
Connect:	`https://ich.taler.net`
NGI Taler:	`https://ngi.taler.net`

Taler Overview



Basic Requirements - Details

back to Basic Requirements

Existence of attestations

$$\underset{\substack{a \in \mathbb{N}_M \\ \omega \in \Omega}}{\forall} : \mathrm{Commit}(a, \omega) =: (Q, P) \implies \mathrm{Attest}(m, Q, P) = \begin{cases} T \in \mathbb{T}, \text{ if } m \leq a \\ \perp \text{ otherwise} \end{cases}$$

Basic Requirements - Details

back to Basic Requirements

Existence of attestations

$$\underset{\substack{a \in \mathbb{N}_M \\ \omega \in \Omega}}{\forall} : \mathrm{Commit}(a, \omega) =: (Q, P) \implies \mathrm{Attest}(m, Q, P) = \begin{cases} T \in \mathbb{T}, & \text{if } m \leq a \\ \perp & \text{otherwise} \end{cases}$$

Efficacy of attestations

$$\mathrm{Verify}(m, Q, T) = \begin{cases} 1, \text{if } \underset{P \in \mathbb{P}}{\exists} : \mathrm{Attest}(m, Q, P) = T \\ 0 \text{ otherwise} \end{cases}$$

$$\forall_{n \leq a} : \mathrm{Verify}\big(n, Q, \mathrm{Attest}(n, Q, P)\big) = 1.$$

...

Basic Requirements - Details

back to Basic Requirements

Existence of attestations

$$\bigforall_{\substack{a \in \mathbb{N}_M \\ \omega \in \Omega}} : \mathrm{Commit}(a, \omega) =: (Q, P) \implies \mathrm{Attest}(m, Q, P) = \begin{cases} T \in \mathbb{T}, & \text{if } m \le a \\ \perp & \text{otherwise} \end{cases}$$

Efficacy of attestations

$$\mathrm{Verify}(m, Q, T) = \begin{cases} 1, \text{if } \displaystyle\exists_{P \in \mathbb{P}} : \mathrm{Attest}(m, Q, P) = T \\ 0 \text{ otherwise} \end{cases}$$

$$\forall_{n \le a} : \mathrm{Verify}\big(n, Q, \mathrm{Attest}(n, Q, P)\big) = 1.$$

...

Derivability of commitments and attestations ...

Basic Requirements - Details

back to Basic Requirements

Existence of attestations

$$\underset{\substack{a \in \mathbb{N}_M \\ \omega \in \Omega}}{\forall} : \mathrm{Commit}(a, \omega) =: (Q, P) \implies \mathrm{Attest}(m, Q, P) = \begin{cases} T \in \mathbb{T}, & \text{if } m \leq a \\ \bot & \text{otherwise} \end{cases}$$

Efficacy of attestations

$$\mathrm{Verify}(m, Q, T) = \begin{cases} 1, \text{if } \underset{P \in \mathbb{P}}{\exists} : \mathrm{Attest}(m, Q, P) = T \\ 0 \text{ otherwise} \end{cases}$$

$$\forall_{n \leq a} : \mathrm{Verify}\big(n, Q, \mathrm{Attest}(n, Q, P)\big) = 1.$$

...

Derivability of commitments and attestations ...

Basic Requirements - Details

back to Basic Requirements

Existence of attestations

$$\underset{\substack{a \in \mathbb{N}_M \\ \omega \in \Omega}}{\forall} : \mathsf{Commit}(a, \omega) =: (Q, P) \implies \mathsf{Attest}(m, Q, P) = \begin{cases} T \in \mathbb{T}, & \text{if } m \leq a \\ \perp & \text{otherwise} \end{cases}$$

Efficacy of attestations

$$\mathsf{Verify}(m, Q, T) = \begin{cases} 1, \text{if } \underset{P \in \mathbb{P}}{\exists} : \mathsf{Attest}(m, Q, P) = T \\ 0 \text{ otherwise} \end{cases}$$

$$\forall_{n \leq a} : \mathsf{Verify}\big(n, Q, \mathsf{Attest}(n, Q, P)\big) = 1.$$

...

Derivability of commitments and attestations ...

More details in the published paper.

Reminder: RSA blind signature

In RSA, a public key (e, N) and private key (d, N) have the property

$$x^{ed} = x \mod N$$

Reminder: RSA blind signature

In RSA, a public key (e, N) and private key (d, N) have the property

$$x^{ed} = x \mod N$$

Bob (B) creates a blind signature of a message m for Alice (A):

back to *taler* or *binding*

Reminder: RSA blind signature

In RSA, a public key (e, N) and private key (d, N) have the property

$$x^{ed} = x \mod N$$

Bob (B) creates a blind signature of a message m for Alice (A):

A:
- chooses random integer b

back to *taler* or *binding*

Reminder: RSA blind signature

In RSA, a public key (e, N) and private key (d, N) have the property

$$x^{ed} = x \mod N$$

Bob (B) creates a blind signature of a message m for Alice (A):

A:
- chooses random integer b
- calculates $m' := m * b^e$ *(blinding)*

back to *taler* or *binding*

Reminder: RSA blind signature

In RSA, a public key (e, N) and private key (d, N) have the property

$$x^{ed} = x \mod N$$

Bob (B) creates a blind signature of a message m for Alice (A):

A:
- chooses random integer b
- calculates $m' := m * b^e$ *(blinding)*
- sends m' to B.

back to *taler* or *binding*

Reminder: RSA blind signature

In RSA, a public key (e, N) and private key (d, N) have the property

$$x^{ed} = x \mod N$$

Bob (B) creates a blind signature of a message m for Alice (A):

A:
- chooses random integer b
- calculates $m' := m * b^e$ *(blinding)*
- sends m' to B.

back to *taler* or *binding*

Reminder: RSA blind signature

In RSA, a public key (e, N) and private key (d, N) have the property

$$x^{ed} = x \mod N$$

Bob (B) creates a blind signature of a message m for Alice (A):

A:
- chooses random integer b
- calculates $m' := m * b^e$ *(blinding)*
- sends m' to B.

B:
- signs m', by calculating $\sigma' := (m')^d \mod N$ *(B doesn't learn m)*

back to *taler* or *binding*

Reminder: RSA blind signature

In RSA, a public key (e, N) and private key (d, N) have the property

$$x^{ed} = x \mod N$$

Bob (B) creates a blind signature of a message m for Alice (A):

A:
- chooses random integer b
- calculates $m' := m * b^e$ *(blinding)*
- sends m' to B.

B:
- signs m', by calculating $\sigma' := (m')^d \mod N$ *(B doesn't learn m)*
- sends σ' to A.

back to *taler* or *binding*

Reminder: RSA blind signature

In RSA, a public key (e, N) and private key (d, N) have the property

$$x^{ed} = x \mod N$$

Bob (B) creates a blind signature of a message m for Alice (A):

A:
- chooses random integer b
- calculates $m' := m * b^e$ *(blinding)*
- sends m' to B.

B:
- signs m', by calculating $\sigma' := (m')^d \mod N$ *(B doesn't learn m)*
- sends σ' to A.

  Note: $(m')^d = (m * b^e)^d = m^d * b^{ed} = m^d * b \mod N$

back to *taler* or *binding*

Reminder: RSA blind signature

In RSA, a public key (e, N) and private key (d, N) have the property

$$x^{ed} = x \mod N$$

Bob (B) creates a blind signature of a message m for Alice (A):

A:
- chooses random integer b
- calculates $m' := m * b^e$ *(blinding)*
- sends m' to B.

B:
- signs m', by calculating $\sigma' := (m')^d \mod N$ *(B doesn't learn m)*
- sends σ' to A.

  Note: $(m')^d = (m * b^e)^d = m^d * b^{ed} = m^d * b \mod N$

back to *taler* or *binding*

Reminder: RSA blind signature

In RSA, a public key (e, N) and private key (d, N) have the property

$$x^{ed} = x \mod N$$

Bob (B) creates a blind signature of a message m for Alice (A):

A:
- chooses random integer b
- calculates $m' := m * b^e$ *(blinding)*
- sends m' to B.

B:
- signs m', by calculating $\sigma' := (m')^d \mod N$ *(B doesn't learn m)*
- sends σ' to A.

  Note: $(m')^d = (m * b^e)^d = m^d * b^{ed} = m^d * b \mod N$

A:
- unblinds σ' by calculating

$$\sigma := \sigma' * b^{-1} (= m^d)$$

back to *taler* or *binding*

Reminder: RSA blind signature

In RSA, a public key (e, N) and private key (d, N) have the property

$$x^{ed} = x \mod N$$

Bob (B) creates a blind signature of a message m for Alice (A):

A:
- chooses random integer b
- calculates $m' := m * b^e$ *(blinding)*
- sends m' to B.

B:
- signs m', by calculating $\sigma' := (m')^d \mod N$ *(B doesn't learn m)*
- sends σ' to A.

  Note: $(m')^d = (m * b^e)^d = m^d * b^{ed} = m^d * b \mod N$

A:
- unblinds σ' by calculating

$$\sigma := \sigma' * b^{-1} (= m^d)$$

$\implies$ σ is a valid RSA signature to message m.

back to *taler* or *binding*

Instantiation with Edx25519

But... isn't ECDSA considered to be difficult to implement correctly?

Instantiation with Edx25519

But... isn't ECDSA considered to be difficult to implement correctly?

We also formally define another signature scheme, Edx25519:

- based on EdDSA (Bernstein et al.),
- generates compatible signatures,
- allows for key derivation from both, private and public keys, independently and
- is already in use in GNUnet.

Current implementation of age restriction in GNU Taler uses Edx25519.

Related Work

- Current privacy-perserving systems all based on attribute-based credentials (Koning et al., Schanzenbach et al., Camenisch et al., Au et al.)
- Attribute-based approach lacks support:
  - Complex for consumers and retailers
  - Requires trusted additional authority

- Other approaches tie age-restriction to ability to pay ("debit cards for kids")
  - Advantage: mandatory to payment process
  - Not privacy friendly