NEXT GENERATION INTERNET

GNU Taler

Christian Grothoff

29.05.2026

Learning objectives

Introduction to GNU Taler

How does cut-and-choose work?

How to prove protocols secure with cryptographic games?

Programmable money: Age restrictions

What are the future plans for GNU Taler?

Introduction to GNU Taler

GNU Taler

Digital cash, made **socially responsible**.

Privacy-Preserving, Practical, Taxable, Free Software, Efficient

What is Taler?

https://taler.net/en/features.html

Taler is

- a Free/Libre software *payment system* infrastructure project
- … with a surrounding software ecosystem
- … and a company (Taler Systems S.A.) and community that wants to deploy it as widely as possible.

However, Taler is

- *not* a currency
- *not* a long-term store of value
- *not* a network or instance of a system
- *not* decentralized
- *not* based on proof-of-work or proof-of-stake
- *not* a speculative asset / “get-rich-quick scheme”

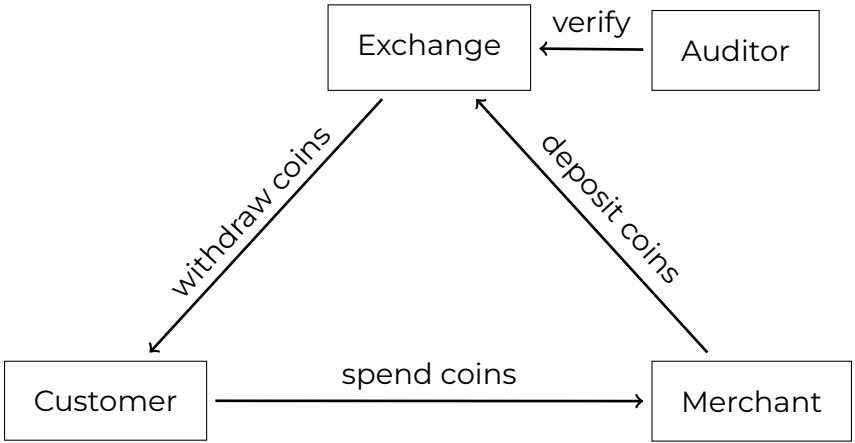
Design goals

… for the GNU Taler payment system

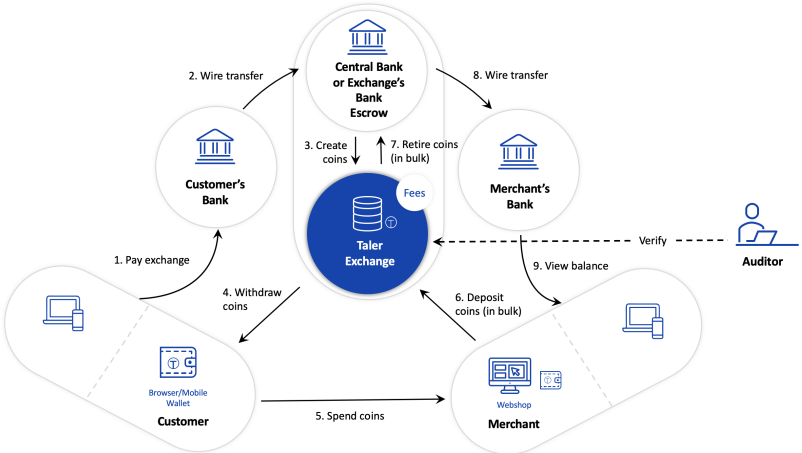
GNU Taler must …

1. … be implemented as **free software**.
2. … protect the **privacy of buyers**.
3. … must enable the state to **tax income** and crack down on illegal business activities.
4. … prevent payment fraud.
5. … only **disclose the minimal amount of information necessary**.
6. … be usable.
7. … be efficient.
8. … avoid single points of failure.
9. … foster **competition**.

Taler overview



Architecture of Taler



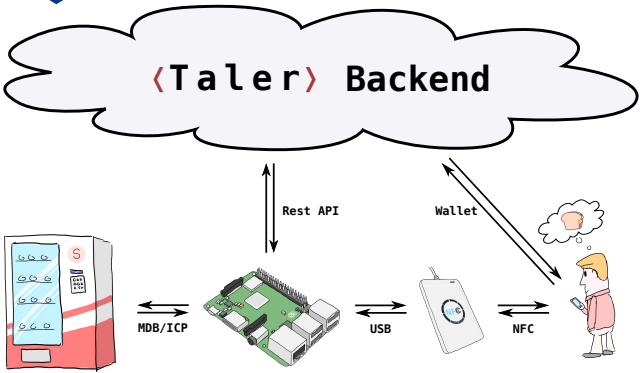
Usability of Taler

`https://demo.taler.net/`

1. Install Web extension.
2. Visit the `bank.demo.taler.net` to withdraw coins.
3. Visit the `shop.demo.taler.net` to spend coins.

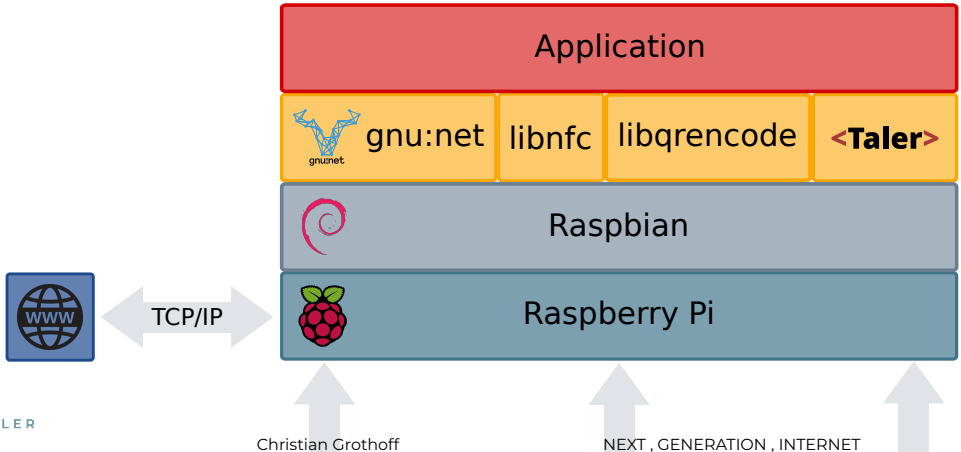
The Taler Snack Machine

Integration of a MDB/ICP to Taler gateway.

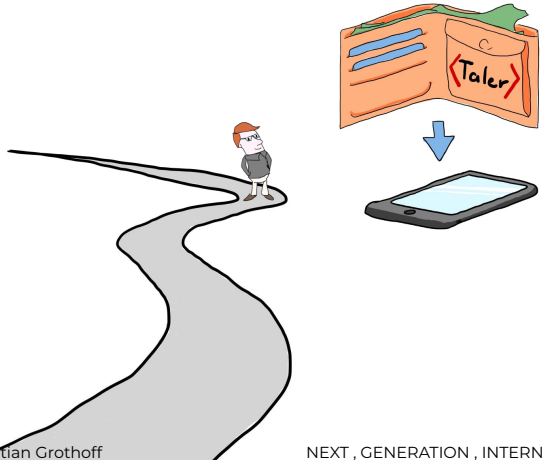


by M. Boss and D. Hofer

Software architecture for the Taler Snack Machine

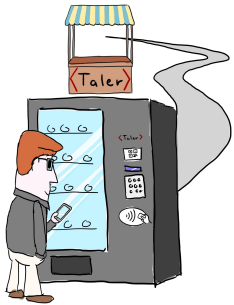


Exercise: Install App on Smartphone

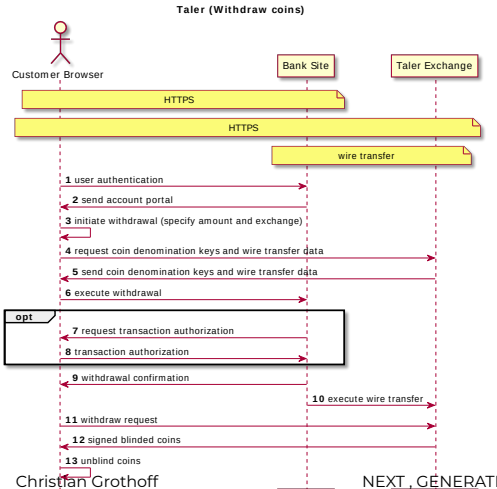


Exercise: Withdraw e-cash

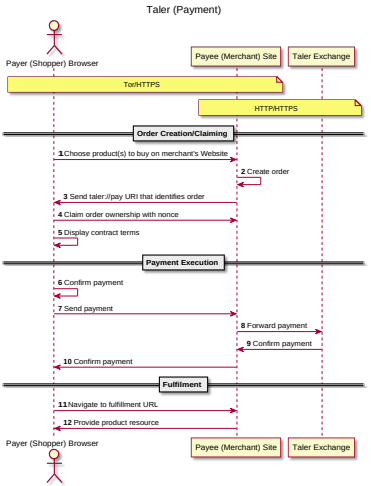
Exercise: Use machine!



Withdrawing coins on the Web



Payment processing with blind signatures



Part II: How does cut-and-choose work?

Taxability

We say Taler is taxable because:

- Merchant's income is visible from deposits.
- Hash of contract is part of deposit data.
- State can trace income and enforce taxation.

Limitations:

- withdraw loophole
- *sharing* coins among family and friends

Other contemporary payment systems have similar limitations on identification, and thus these limitations should not be a legal issue.

Giving change

It would be inefficient to pay EUR 100 with 1 cent coins!

- Denomination key represents value of a coin.
- Exchange may offer various denominations for coins.
- Wallet may not have exact change!
- Usability requires ability to pay given sufficient total funds.

Giving change

It would be inefficient to pay EUR 100 with 1 cent coins!

- Denomination key represents value of a coin.
- Exchange may offer various denominations for coins.
- Wallet may not have exact change!
- Usability requires ability to pay given sufficient total funds.

Key goals:

- maintain unlinkability
- maintain taxability of transactions

Giving change

It would be inefficient to pay EUR 100 with 1 cent coins!

- Denomination key represents value of a coin.
- Exchange may offer various denominations for coins.
- Wallet may not have exact change!
- Usability requires ability to pay given sufficient total funds.

Key goals:

- maintain unlinkability
- maintain taxability of transactions

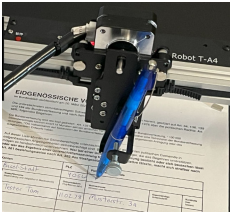
Method:

- Contract can specify to only pay *partial value* of a coin.
- Exchange allows wallet to obtain *unlinkable change* for remaining coin value.

Unique Signatures

- Some public key operations depend on a nonce or “random” value
  - Example: ElGamal (encryption), DSA/ECDSA (signing)
  - \+ same plaintext, different ciphertext
  - \- security may break on nonce-reuse
- Generating the nonce deterministically by hashing all inputs (see also: Fiat-Shamir transformation) can make these algorithms **deterministic**
  - Example: EdDSA
- If only one form of a valid signature exists and the verifier can check this, a signature is **unique**

Unique signatures:



Verifiable Random Functions

Micali, Rabin, & Vadhan (1999) proposed verifiable random functions.

Let M be some input.

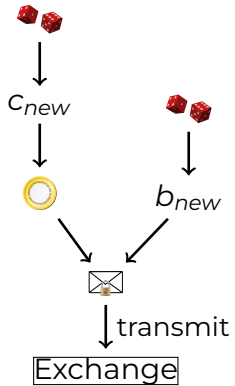
- $(sk, pk) := VRF_{keygen}()$
- $(v, p) := VRF_{sign}(M, sk)$
- v is deterministic, unpredictable and high-entropy for any M and sk, and (v, p) can only be computed with sk
- $VRF_{verify}(M, pk, v, p)$ returns true only if v was computed correctly
- sk cannot be derived from M, pk, v and p

Straw-man solution

Given partially spent private coin key c_{old}:

1. Pick random $c_{new} \mod o$ private key
2. Compute $C_{new} := c_{new}G$ public key
3. Pick random b_{new}
4. Compute $f_{new} := FDH(C_{new})$, $m < n$.
5. Transmit $f'_{new} := f_{new} b^e_{new} \mod n$

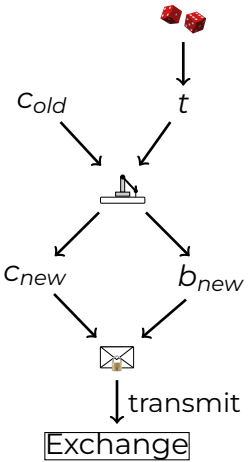
... and sign request for change with c_{old}.



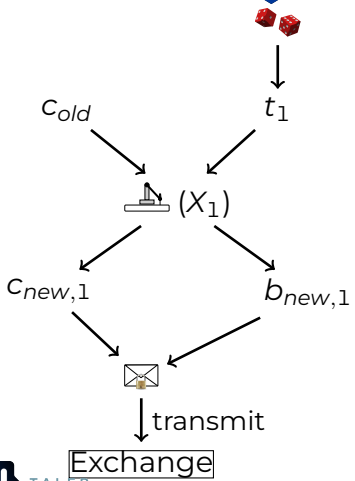
Customer: Transfer setup (UNISIG)

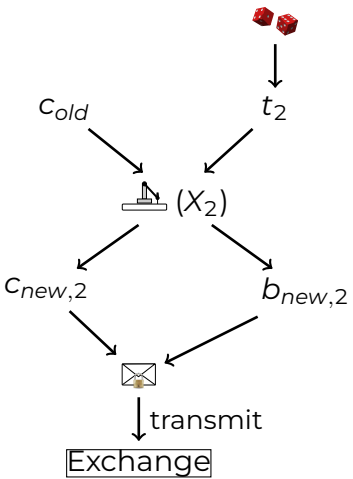
Given partially spent private coin key c_{old}:

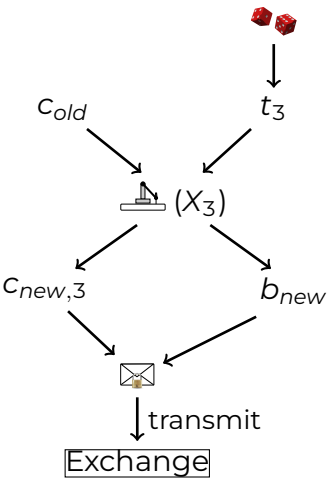
1. Let $C_{old} := c_{old}G$ (as before)
2. Create random nonce t
3. Compute unique signature $X := UNISIG_{c_{old}}(t)$
4. Derive c_{new} and b_{new} from X using HKDF
5. Compute $C_{new} := c_{new}G$
6. Compute $f_{new} := FDH(C_{new})$
7. Transmit $f'_{new} := f_{new}b^e_{new}$



Cut-and-Choose







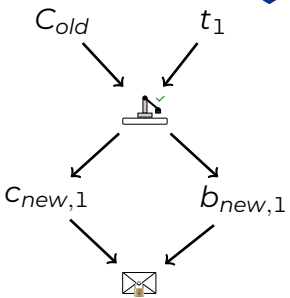
Exchange: Choose!

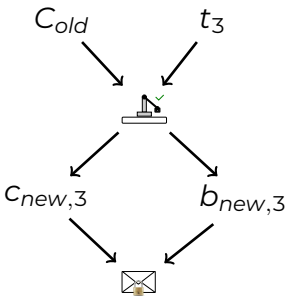
Exchange sends back random $\gamma \in \{1, 2, 3\}$ to the customer.

Customer: Reveal

1. If $\gamma = 1$, send $\langle t_2, X_2 \rangle$, $\langle t_3, X_3 \rangle$ to exchange
2. If $\gamma = 2$, send $\langle t_1, X_1 \rangle$, $\langle t_3, X_3 \rangle$ to exchange
3. If $\gamma = 3$, send $\langle t_1, X_1 \rangle$, $\langle t_2, X_2 \rangle$ to exchange

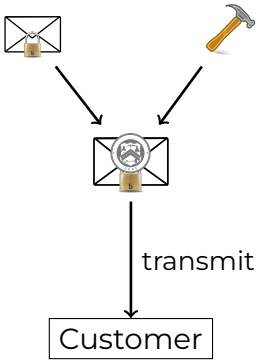
Exchange: Verify ($\gamma = 2$)





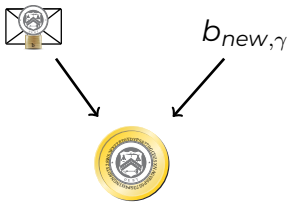
Exchange: Blind sign change (RSA)

1. Take $f'_{new,\gamma}$.
2. Compute $s' := f'^{d}_{new,\gamma} \mod n$.
3. Return signature s'.



Customer: Unblind change (RSA)

1. Receive s'.
2. Compute $s := s' b_{new,\gamma}^{-1} \mod n$.

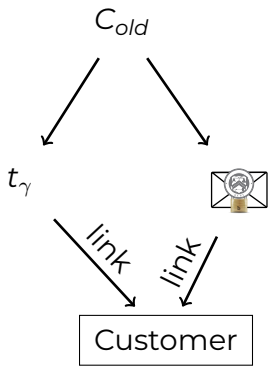


Exchange: Allow linking change

Given C_{old}

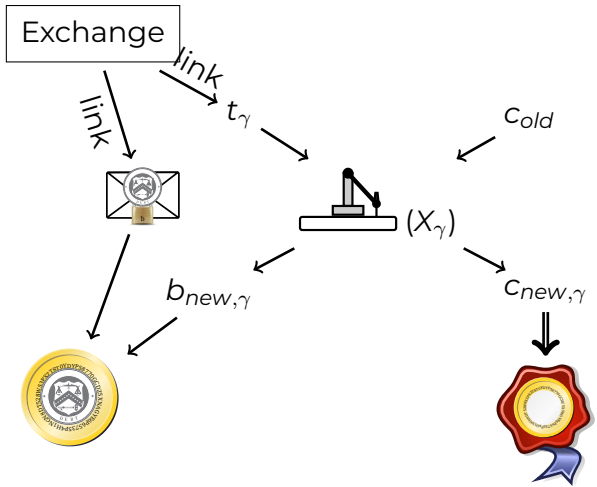
return t_γ and

$s := s' b_{new,\gamma}^{-1} \mod n.$



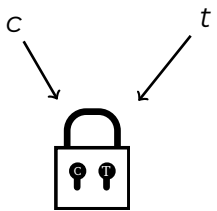
Customer: Link (threat!)

1. Have c_{old}.
2. Obtain T_γ, s from exchange
3. Compute $X_\gamma = UNISIG_{c_{old}}(t_\gamma)$
4. Derive $c_{new,\gamma}$ and $b_{new,\gamma}$ from X_γ
5. Unblind $s := s' b_{new,\gamma}^{-1} \mod n$



VRF via Diffie-Hellman (ECDH)

1. Create private keys $c, t \mod o$
2. Define $C = cG$
3. Define $T = tG$
4. Compute DH $cT = c(tG) = t(cG) = tC$
5. Sign T with EdDSA $\Rightarrow$ DH + EdDSA $\equiv$ VRF: DH is unique, with EdDSA we have a signature, t allows verifier to check!

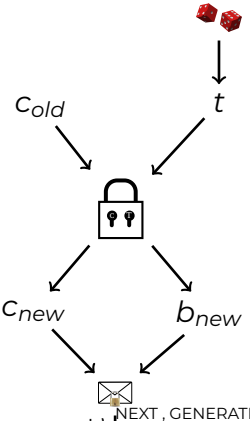


Transfer setup with ECDH-based VRF

This is the VRF replacement for slide 25.

Given partially spent private coin key c_{old}:

1. Let $C_{old} := c_{old}G$ (as before)
2. Create random private transfer key $t \mod o$
3. Compute $T := tG$
4. Compute $X := c_{old}(tG) = t(c_{old}G) = tC_{old}$
5. Derive c_{new} and b_{new} from X



Refresh protocol summary

- Customer asks exchange to convert old coin to new coin
- Protocol ensures new coins can be recovered from old coin

$\Rightarrow$ New coins are owned by the same entity!

Thus, the refresh protocol allows:

- To give unlinkable change.
- To give refunds to an anonymous customer.
- To expire old keys and migrate coins to new ones.
- To handle protocol aborts.

Transactions via refresh are equivalent to *sharing* a wallet.

Part III: How to prove protocols secure with cryptographic games?

Reminder: Cryptographic games

An *oracle* is a party in a game that the adversary can call upon to indirectly access information that is otherwise hidden from it.
For example, **IND-CPA** can be formalized like this:

Setup Generate random key k, select $b \in \{0, 1\}$ for $i \in \{1, \ldots, q\}$.

Oracle Given M_0 and M_1 (of same length), return $C := \mathtt{enc}(k, M_b)$.

The adversary wins, if it can guess b with probability greater than $\frac{1}{2} + \epsilon(\kappa)$ where $\epsilon(\kappa)$ is a negligible function in the security parameter κ.

Programmable money: Age restrictions [3]

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 violated

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

Age restriction design for GNU Taler

Design and implementation of an age restriction scheme with the following goals:

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

Age restriction

Assumptions and scenario

- Assumption: Checking accounts are under control of eligible adults/guardians.

Age restriction

Assumptions and scenario

- Assumption: Checking accounts are under control of eligible adults/guardians.
- *Guardians* **commit** to an maximum age

Age restriction

Assumptions and scenario

- Assumption: Checking accounts are under control of eligible adults/guardians.
- *Guardians* **commit** to an maximum age
- *Minors* **attest** their adequate age

Age restriction

Assumptions and scenario

- Assumption: Checking accounts are under control of eligible adults/guardians.
- *Guardians* **commit** to an maximum age
- *Minors* **attest** their adequate age
- *Merchants* **verify** the attestations

Age restriction

Assumptions and scenario

- Assumption: Checking accounts are under control of eligible adults/guardians.
- *Guardians* **commit** to an maximum age
- *Minors* **attest** their adequate age
- *Merchants* **verify** the attestations
- Minors **derive** age commitments from existing ones

Age restriction

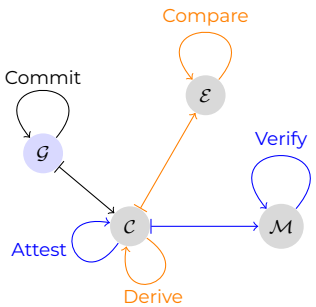
Assumptions and scenario

- Assumption: Checking accounts are under control of eligible adults/guardians.
- *Guardians* **commit** to an maximum age
- *Minors* **attest** their adequate age
- *Merchants* **verify** the attestations
- Minors **derive** age commitments from existing ones
- *Exchanges* **compare** the derived age commitments

Age restriction

Assumptions and scenario

- Assumption: Checking accounts are under control of eligible adults/guardians.
- *Guardians* **commit** to an maximum age
- *Minors* **attest** their adequate age
- *Merchants* **verify** the attestations
- Minors **derive** age commitments from existing ones
- *Exchanges* **compare** the derived age commitments



Formal Function Signatures

Searching for functions

- Commit
- Attest
- Verify
- Derive
- Compare

Formal Function Signatures

Searching for functions with the following signatures

Commit : $(a, \omega) \mapsto (Q, P)$ $\mathbb{N}_M \times \Omega \to \mathbb{O} \times \mathbb{P}$,

Attest

Verify

Derive

Compare

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

Formal Function Signatures

Searching for functions with the following signatures

Commit :	$(a, \omega) \mapsto (Q, P)$	$\mathbb{N}_M \times \Omega \to \mathbb{O} \times \mathbb{P},$
Attest :	$(m, Q, P) \mapsto T$	$\mathbb{N}_M \times \mathbb{O} \times \mathbb{P} \to \mathbb{T} \cup \{\perp\},$
Verify		
Derive		
Compare		

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

Formal Function Signatures

Searching for functions with the following signatures

Commit :	$(a, \omega) \mapsto (Q, P)$	$\mathbb{N}_M \times \Omega \to \mathbb{O} \times \mathbb{P}$,
Attest :	$(m, Q, P) \mapsto T$	$\mathbb{N}_M \times \mathbb{O} \times \mathbb{P} \to \mathbb{T} \cup \{\perp\}$,
Verify :	$(m, Q, T) \mapsto b$	$\mathbb{N}_M \times \mathbb{O} \times \mathbb{T} \to \mathbb{Z}_2$,
Derive		
Compare		

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*t*estation*,

Formal Function Signatures

Searching for functions with the following signatures

Commit :	$(a, \omega) \mapsto (Q, P)$	$\mathbb{N}_M \times \Omega \to \mathbb{O} \times \mathbb{P},$
Attest :	$(m, Q, P) \mapsto T$	$\mathbb{N}_M \times \mathbb{O} \times \mathbb{P} \to \mathbb{T} \cup \{\perp\},$
Verify :	$(m, Q, T) \mapsto b$	$\mathbb{N}_M \times \mathbb{O} \times \mathbb{T} \to \mathbb{Z}_2,$
Derive :	$(Q, P, \omega) \mapsto (Q', P', \beta)$	$\mathbb{O} \times \mathbb{P} \times \Omega \to \mathbb{O} \times \mathbb{P} \times \mathbb{B},$
Compare		

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

Formal Function Signatures

Searching for functions with the following signatures

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

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

Formal Function Signatures

Searching for functions with the following signatures

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

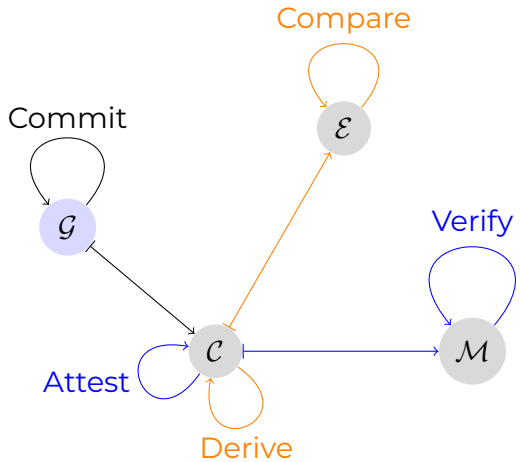
with $\Omega, \mathbb{P}, \mathbb{O}, \mathbb{T}, \mathbb{B}$ sufficiently large sets.

Basic and security requirements are defined later.

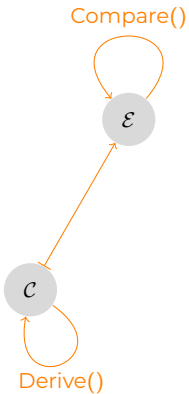
Mnemonics:
$\mathbb{O}$ = *cOmmitments*, Q = *Q-mitment* (commitment), $\mathbb{P}$ = *Proofs*, P = *Proof*,
$\mathbb{T}$ = *aTtestations*, T = *aTtestation*, $\mathbb{B}$ = *Blindings*, β = *βlinding*.

Age restriction

Naïve scheme

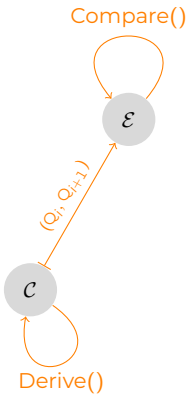


Achieving Unlinkability



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

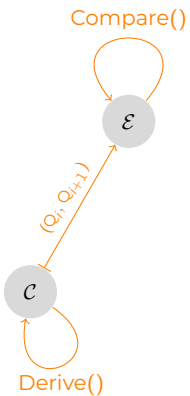
Achieving Unlinkability



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

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

Achieving Unlinkability



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

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

$\Longrightarrow$ **Exchange identifies sequence**

$\Longrightarrow$ **Unlinkability broken**

Achieving Unlinkability

Define cut&choose protocol DeriveCompare$_\kappa$, using Derive() and Compare().

Achieving Unlinkability

Define cut&choose protocol DeriveCompare$_\kappa$, using Derive() and Compare().

Sketch:

1. $\mathcal{C}$ derives commitments $(Q_1, \ldots, Q_\kappa)$ from Q_0 by calling Derive() with blindings $(\beta_1, \ldots, \beta_\kappa)$
2. $\mathcal{C}$ calculates $h_0 := H\left(H(Q_1, \beta_1)|| \ldots ||H(Q_\kappa, \beta_\kappa)\right)$
3. $\mathcal{C}$ sends Q_0 and h_0 to $\mathcal{E}$
4. $\mathcal{E}$ chooses $\gamma \in \{1, \ldots, \kappa\}$ randomly
5. $\mathcal{C}$ reveals $h_\gamma := H(Q_\gamma, \beta_\gamma)$ and all (Q_i, β_i), except (Q_γ, β_γ)
6. $\mathcal{E}$ compares h_0 and $H\left(H(Q_1, \beta_1)||...||h_\gamma||...||H(Q_\kappa, \beta_\kappa)\right)$ and evaluates Compare(Q_0, Q_i, β_i).

Note: Scheme is similar to the *refresh* protocol in GNU Taler.

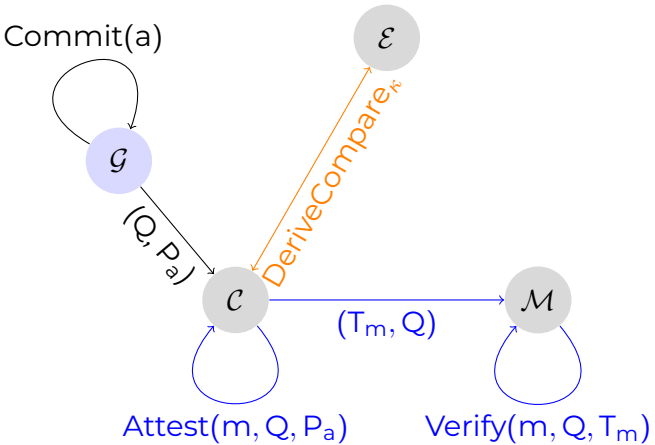
Achieving Unlinkability

With DeriveCompare$_\kappa$

- $\mathcal{E}$ learns nothing about Q_γ,
- trusts outcome with $\frac{\kappa-1}{\kappa}$ certainty,
- i.e. $\mathcal{C}$ has $\frac{1}{\kappa}$ chance to cheat.

Note: Still need Derive and Compare to be defined.

Refined scheme



Achieving Unlinkability

$\mathsf{DeriveCompare}_\kappa : \mathbb{O} \times \mathbb{P} \times \Omega \to \{0,1\}$
$\mathsf{DeriveCompare}_\kappa(\mathsf{Q},\mathsf{P},\omega) =$

$\mathcal{C}$: 1. *for all* $i \in \{1,\ldots,\kappa\}$: $(\mathsf{Q}_i,\mathsf{P}_i,\beta_i) \leftarrow \mathsf{Derive}(\mathsf{Q},\mathsf{P},\omega+i)$
2. $h \leftarrow \mathsf{H}\big(\mathsf{H}(\mathsf{Q}_1,\beta_1) \parallel \cdots \parallel \mathsf{H}(\mathsf{Q}_\kappa,\beta_\kappa)\big)$
3. *send* (Q,h) *to* $\mathcal{E}$

$\mathcal{E}$: 4. *save* (Q,h)
5. $\gamma \xleftarrow{\$} \{1,\ldots,\kappa\}$
6. *send* γ *to* $\mathcal{C}$

$\mathcal{C}$: 7. $h'_\gamma \leftarrow \mathsf{H}(\mathsf{Q}_\gamma,\beta_\gamma)$
8. $\mathbf{E}_\gamma \leftarrow \big[(\mathsf{Q}_1,\beta_1),\ldots,(\mathsf{Q}_{\gamma-1},\beta_{\gamma-1}),\perp,(\mathsf{Q}_{\gamma+1},\beta_{\gamma+1}),\ldots,(\mathsf{Q}_\kappa,\beta_\kappa)\big]$
9. *send* $(\mathbf{E}_\gamma,h'_\gamma)$ *to* $\mathcal{E}$

$\mathcal{E}$: 10. *for all* $i \in \{1,\ldots,\kappa\} \setminus \{\gamma\}$: $h_i \leftarrow \mathsf{H}(\mathbf{E}_\gamma[i])$
11. *if* $h \overset{?}{\neq} \mathsf{H}(h_1\|\ldots\|h_{\gamma-1}\|h'_\gamma\|h_{\gamma+1}\|\ldots\|h_{\kappa-1})$ *return* 0
12. *for all* $i \in \{1,\ldots,\kappa\} \setminus \{\gamma\}$: *if* $0 \overset{?}{=} \mathsf{Compare}(\mathsf{Q},\mathsf{Q}_i,\beta_i)$ *return* 0
13. *return* 1

Basic Requirements

Candidate functions

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

must first meet *basic* requirements:

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

Basic Requirements

Formal Details

Existence of attestations

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

Efficacy of attestations

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

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

etc.

Requirements
Details

Derivability of commitments and proofs:

Let

$$\begin{aligned} \mathsf{a} \in \mathbb{N}_{\mathsf{M}},\ \omega_0, \omega_1 \in \Omega \\ (\mathsf{Q}_0, \mathsf{P}_0) \leftarrow \mathsf{Commit}(\mathsf{a}, \omega_0), \\ (\mathsf{Q}_1, \mathsf{P}_1, \beta) \leftarrow \mathsf{Derive}(\mathsf{Q}_0, \mathsf{P}_0, \omega_1). \end{aligned}$$

We require

$$\mathsf{Compare}(\mathsf{Q}_0, \mathsf{Q}_1, \beta) = 1$$

and for all $n \leq \mathsf{a}$:

$$\mathsf{Verify}(n, \mathsf{Q}_1, \mathsf{Attest}(n, \mathsf{Q}_1, \mathsf{P}_1)) = \mathsf{Verify}(n, \mathsf{Q}_0, \mathsf{Attest}(n, \mathsf{Q}_0, \mathsf{P}_0))$$

Security Requirements

Candidate functions must also meet *security* requirements. Those are defined via security games:

- ▶ Game: Age disclosure by commitment or attestation
- ↔ Requirement: Non-disclosure of age
- ▶ Game: Forging attestation
- ↔ Requirement: Unforgeability of minimum age
- ▶ Game: Distinguishing derived commitments and attestations
- ↔ Requirement: Unlinkability of 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 $G_{\mathcal{A}}^{\mathsf{FA}}(\lambda)$—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: Unforgeability of minimum age

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

Solution: Instantiation with ECDSA

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

Solution: Instantiation with ECDSA

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

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

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

Solution: Instantiation with ECDSA

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

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

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

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

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

- $\vec{Q} := (q_1, \ldots, q_M)$ is the *Commitment*,
- $\vec{P}_a := (p_1, \ldots, p_a, \bot, \ldots, \bot)$ is the *Proof*

Solution: Instantiation with ECDSA

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

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

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

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

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

   - $\vec{Q} := (q_1, \ldots, q_M)$ is the *Commitment*,
   - $\vec{P}_a := (p_1, \ldots, p_a, \bot, \ldots, \bot)$ is the *Proof*

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

Instantiation with ECDSA

Definitions of Attest and Verify

Child has

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

Instantiation with ECDSA

Definitions of Attest and Verify

Child has

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

To Attest a minimum age $m \leq a$:

Sign a message with ECDSA using private key p_m

Instantiation with ECDSA

Definitions of Attest and Verify

Child has

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

To Attest a minimum age $m \leq a$:

Sign a message with ECDSA using private key p_m

Merchant gets

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

Instantiation with ECDSA

Definitions of Attest and Verify

Child has

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

To Attest a minimum age $m \leq a$:

Sign a message with ECDSA using private key p_m

Merchant gets

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

To Verify a minimum age m:

Verify the ECDSA-Signature σ with public key q_m.

Instantiation with ECDSA

Definitions of Derive and Compare

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

Instantiation with ECDSA

Definitions of Derive and Compare

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

$$\vec{Q}' := (\beta * q_1, \ldots, \beta * q_M),$$
$$\vec{P}' := (\beta p_1, \ldots, \beta p_a, \perp, \ldots, \perp)$$

Note: $(\beta p_i) * G = \beta * (p_i * G) = \beta * q_i$

$\beta * q_i$ is scalar multiplication on the elliptic curve.

Instantiation with ECDSA

Definitions of Derive and Compare

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

$$\vec{Q}' := (\beta * q_1, \ldots, \beta * q_M),$$
$$\vec{P}' := (\beta p_1, \ldots, \beta p_a, \perp, \ldots, \perp)$$

Note: $(\beta p_i) * G = \beta * (p_i * G) = \beta * q_i$

$\beta * q_i$ is scalar multiplication on the elliptic curve.

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.
  Proofs by security reduction, details are in the paper.

Instantiation with ECDSA

Full definitions

$$\text{Commit}_{E,[\cdot]_g}(a,\omega) := \Big\langle \overbrace{(q_1,\ldots,q_M)}^{=\vec{Q}}, \overbrace{(p_1,\ldots,p_a,\perp,\ldots,\perp)}^{=\vec{P},\ \text{length M}} \Big\rangle$$

$$\text{Attest}_{E,H}(b,\vec{Q},\vec{P}) := \begin{cases} T_b := \text{Sig}_{E,H}(b,\vec{P}[b]) & \text{if } \vec{P}[b] \stackrel{?}{\neq} \perp \\ \perp & \text{otherwise} \end{cases}$$

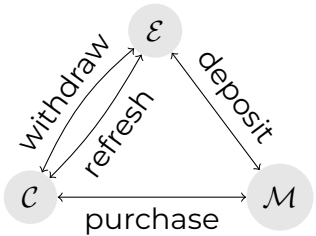
$$\text{Verify}_{E,H}(b,\vec{Q},T) := \text{Ver}_{E,H}(b,\vec{Q}[b],T)$$

$$\text{Derive}_{E,[\cdot]_g}(\vec{Q},\vec{P},\omega) := \Big\langle (\beta * q_1,\ldots,\beta * q_M),(\beta p_1,\ldots,\beta p_a,\perp,\ldots,\perp),\beta \Big\rangle$$

$$\text{with } \beta := [\omega]_g \text{ and multiplication } \beta p_i \text{ modulo } g$$

$$\text{Compare}_E(\vec{Q},\vec{Q}',\beta) := \begin{cases} 1 & \text{if } (\beta * q_1,\ldots,\beta * q_M) \stackrel{?}{=} (q'_1,\ldots,q'_M) \\ 0 & \text{otherwise} \end{cases}$$

Reminder: GNU Taler Fundamentals



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

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

(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 signing FDH(C_p), $\mathcal{E}$ now blindly signs

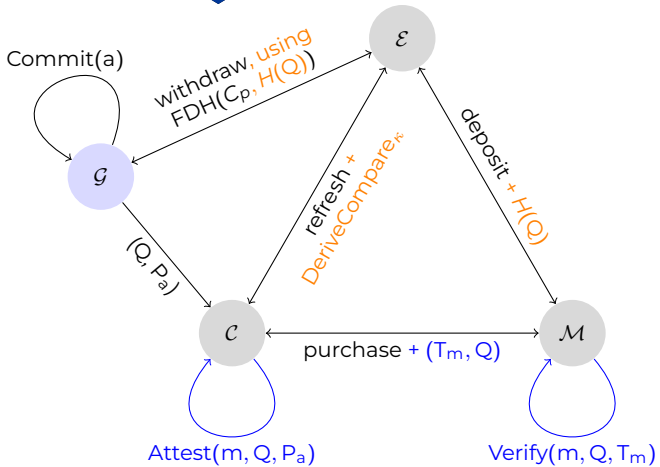
$$\text{FDH}(C_p, H(\text{Q}))$$

Verfication of a coin now requires H(Q), too:

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

Integration with GNU Taler

Integrated schemes



Instantiation with Edx25519

Paper also formally defines another signature scheme: Edx25519.

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

Current implementation of age restriction in GNU Taler uses Edx25519.

Age Restrictions based on KYC

Subsidiarity requires bank accounts being owned by adults.

- Scheme can be adapted to case where minors have bank accounts
  - Assumption: banks provide minimum age information during bank transactions.
  - Child and Exchange execute a variant of the cut&choose protocol.

Discussion

- Our solution can in principle be used with any token-based payment scheme
- GNU Taler best aligned with our design goals (security, privacy and efficiency)
- Subsidiarity requires bank accounts being owned by adults
  - Scheme can be adapted to case where minors have bank accounts
    - Assumption: banks provide minimum age information during bank transactions.
    - Child and Exchange execute a variant of the cut&choose protocol.
- Our scheme offers an alternative to identity management systems (IMS)

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 third authority

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

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 a solution that is

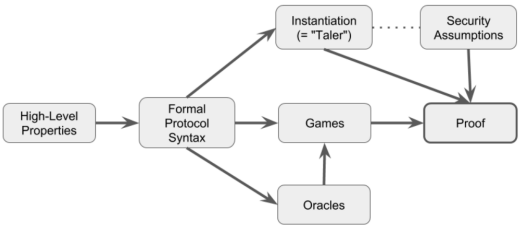
- based on subsidiarity
- privacy preserving
- efficient
- an alternative to IMS

Part VI: What are the future plans for GNU Taler?

Summary

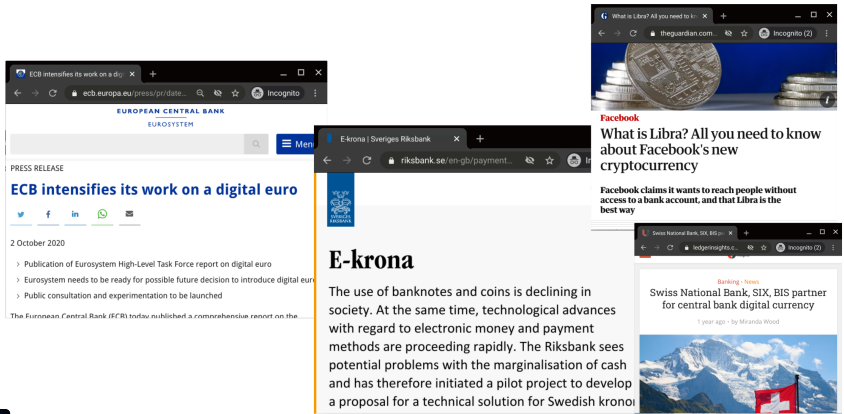
GNU Taler:

- Gives change, can provide refunds
- Integrates nicely with HTTP, handles network failures
- Has high performance
- Is Free Software
- Includes formal security proofs



CBDC initiatives and GNU Taler

Many initiatives are currently at the level of requirements discussion:



Unique regulatory features for CBs

1. Central bank issues digital coins equivalent to issuing cash
2. Architecture with consumer accounts at commercial banks
3. Withdrawal limits and denomination expiration
4. Income transparency and possibility to set fees
5. Revocation protocols and loss limitations
6. Privacy by cryptographic design not organizational compliance

Political support needed, talk to your representatives!

Ongoing work

- Post-quantum blind signatures
- Unlinkable subscriptions and discounts
- Privacy-preserving donations
- SAP integration
- Design and usability for illiterate and innumerate users
- Internationalization $\Rightarrow$ `https://weblate.taler.net/`

`https://bugs.taler.net/` tracks open issues.

Open issues / Future Work

- Integration into more physical machines
- Support more core banking / blockchain protocols
- Wallet backup and recovery with Anastasis
- Defeat AI & spam with micropayments
- Implement *usable* card game on Polkadot
- Break more HSMs (side-channels, fault injection)
- Currency conversion
- Integration with e-commerce frameworks (Prestashop, OpenCart, ECWID, ...)
- Federated exchange (wads)
- ...

Help needed, talk to us (e.g. at `https://ich.taler.net/`)

Visions

- Be paid to read advertising, starting with spam
- Give welfare without intermediaries taking huge cuts
- Forster regional trade via regional currencies
- Eliminate corruption by making all income visible
- Stop the mining by making crypto-currencies useless for anything but crime

References I

Jeffrey Burdges, Florian Dold, Christian Grothoff, and Marcello Stanisci.
Enabling secure web payments with GNU Taler.
In Claude Carlet, M. Anwar Hasan, and Vishal Saraswat, editors, *6th International Conference on Security, Privacy and Applied Cryptographic Engineering*, number 10076 in LNCS, pages 251–270. Springer, Dec 2016.

References II

David Chaum, Christian Grothoff, and Thomas Moser.
How to issue a central bank digital currency.
In *SNB Working Papers*, number 2021-3. Swiss National Bank, February 2021.

Özgür Kesim, Christian Grothoff, Florian Dold, and Martin Schanzenbach.
Zero-Knowledge Age Restriction for GNU Taler.
In Vijayalakshmi Atluri, Roberto Di Pietro, Christian D. Jensen, and Weizhi Meng, editors, *Computer Security – ESORICS 2022*, pages 110–129, Cham, 2022. Springer International Publishing.

Acknowledgements

Co-funded by the European Union (Project 101135475).

Co-funded by SERI (HEU-Projekt 101135475-TALER).

Project funded by

Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Swiss Confederation

Federal Department of Economic Affairs,
Education and Research EAER
State Secretariat for Education,
Research and Innovation SERI

Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union.

Neither the European Union nor the granting authority can be held responsible for them.