taler.net
taler@twitter
taler-systems.com

Christian Grothoff
grothoff@taler.net
Agenda

Motivation & Background

GNU Taler: Introduction

Protocol Basics

Programmable money: Age restrictions

Blockchain integration: Project Depolymerization

Future Work & Conclusion
A Social Problem

This was a question posed to RAND researchers in 1971:

“Suppose you were an advisor to the head of the KGB, the Soviet Secret Police. Suppose you are given the assignment of designing a system for the surveillance of all citizens and visitors within the boundaries of the USSR. The system is not to be too obtrusive or obvious. What would be your decision?”
A Social Problem

This was a question posed to RAND researchers in 1971:

“Suppose you were an advisor to the head of the KGB, the Soviet Secret Police. Suppose you are given the assignment of designing a system for the surveillance of all citizens and visitors within the boundaries of the USSR. The system is not to be too obtrusive or obvious. What would be your decision?”

“I think one of the big things that we need to do, is we need to get away from true-name payments on the Internet. The credit card payment system is one of the worst things that happened for the user, in terms of being able to divorce their access from their identity.”

The Bank of International Settlements
GNU Taler: Introduction
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 or speculative asset
▶ *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
Design principles
https://taler.net/en/principles.html

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

Exchange

verify

Customer
withdraw coins

Auditor
deposit coins

spend coins

Merchant
Architecture of Taler

1. Pay exchange
2. Wire transfer
3. Create coins
4. Withdraw coins
5. Spend coins
6. Deposit coins (in bulk)
7. Retire coins (in bulk)
8. Wire transfer
9. View balance

Browser/Mobile Wallet
Customer

Central Bank
or Exchange’s Bank Escrow

Fees

Merchant’s Bank

Taler Exchange

Webshop
Merchant

Auditor

Verify
Usability of Taler

https://demo.taler.net/

1. Install browser extension.
2. Visit the bank.demo.taler.net to withdraw coins.
3. Visit the shop.demo.taler.net to spend coins.
Protocol Basics
How does it work?

We use a few ancient constructions:

- Cryptographic hash function (1989)
- Blind signature (1983)
- Schnorr signature (1989)
- Diffie-Hellman key exchange (1976)
- Cut-and-choose zero-knowledge proof (1985)

But of course we use modern instantiations.
Definition: 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.
Definition: 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
Exchange setup: Create a denomination key (RSA)

1. Pick random primes \( p, q \).
2. Compute \( n := pq \),
   \[ \phi(n) = (p - 1)(q - 1) \]
3. Pick small \( e < \phi(n) \) such that
   \( d := e^{-1} \mod \phi(n) \) exists.
4. Publish public key \((e, n)\).
Merchant: Create a signing key (EdDSA)

- pick random $m \mod o$ as private key
- $M = mG$ public key

Capability: $m \Rightarrow M$
Customer: Create a planchet (EdDSA)

- Pick random $c \mod o$ private key
- $C = cG$ public key

Capability: $c \Rightarrow$
Customer: Blind planchet (RSA)

1. Obtain public key \((e, n)\)
2. Compute \(f := FDH(C), f < n\).
3. Pick blinding factor \(b \in \mathbb{Z}_n\)
4. Transmit \(f' := fb^e \mod n\)
Exchange: Blind sign (RSA)

1. Receive \( f' \).
2. Compute \( s' := f'^d \mod n \).
3. Send signature \( s' \).
Customer: Unblind coin (RSA)

1. Receive $s'$.
2. Compute $s := s'b^{-1} \mod n$
Customer: Build shopping cart
Merchant: Propose contract (EdDSA)

1. Complete proposal $D$.
2. Send $D, \text{EdDSA}_m(D)$
Customer: Spend coin (EdDSA)

1. Receive proposal $D$, $EdDSA_m(D)$.
2. Send $s$, $C$, $EdDSA_c(D)$

Merchant
Merchant and Exchange: Verify coin (RSA)

\[ s^e \equiv FDH(C) \mod n \]

The exchange does not only verify the signature, but also checks that the coin was not double-spent.
Merchant and Exchange: Verify coin (RSA)

\[ s^e \equiv FDH(C) \mod n \]

The exchange does not only verify the signature, but also checks that the coin was not double-spent.

Taler is an online payment system.
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.
Diffie-Hellman (ECDH)

1. Create private keys $c, t \mod o$
2. Define $C = cG$
3. Define $T = tG$
4. Compute DH
   \[ cT = c(tG) = t(cG) = tC \]
Given partially spent private coin key $c_{old}$:

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

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

---

**Exchange**

---

**Problem:** Owner of $c_{new}$ may differ from owner of $c_{old}$!
Strawman solution

Given partially spent private coin key $c_{old}$:

1. Pick random $c_{new}$ mod 0 private key
2. $C_{new} = c_{new} G$ public key
3. Pick random $b_{new}$
4. Compute $f_{new} := FDH(C_{new}), m < n.$
5. Transmit $f'_{new} := f_{new} b_{new}^e \mod n$

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

Problem: Owner of $c_{new}$ may differ from owner of $c_{old}$!
Customer: Transfer key setup (ECDH)

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 $X$
6. Compute $C_{new} := c_{new} G$
7. Compute $f_{new} := FDH(C_{new})$
8. Transmit $f'_{new} := f_{new} b_{new}^e$
Cut-and-Choose

\[ \text{old} \rightarrow t_1 \rightarrow c_{\text{new},1} \rightarrow b_{\text{new},1} \rightarrow \text{Exchange} \]

\[ \text{old} \rightarrow t_2 \rightarrow c_{\text{new},2} \rightarrow b_{\text{new},2} \rightarrow \text{Exchange} \]

\[ \text{old} \rightarrow t_3 \rightarrow c_{\text{new},3} \rightarrow b_{\text{new},3} \rightarrow \text{Exchange} \]
Exchange: Choose!

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

1. If $\gamma = 1$, send $t_2, t_3$ to exchange
2. If $\gamma = 2$, send $t_1, t_3$ to exchange
3. If $\gamma = 3$, send $t_1, t_2$ to exchange
Exchange: Verify ($\gamma = 2$)
Exchange: Blind sign change (RSA)

1. Take $f'_{\text{new},\gamma}$.
2. Compute $s' := f'^d_{\text{new},\gamma} \mod n$.
3. Send signature $s'$.
Customer: Unblind change (RSA)

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

$XNAGYE6P65735P4H1NGN8DT528W$

$S3PXZT8T0YDYPS8770GCDZ5$

$b_{new, \gamma}$
Exchange: Allow linking change

Given $C_{old}$

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

Diagram:

- $C_{old}$
- $T_{\gamma}$
- Customer

Links:
- $C_{old}$ to $T_{\gamma}$
- $T_{\gamma}$ to Customer

Equation:

\[ C_{old} \]

\[ T_{\gamma} \]

\[ \text{return } T_{\gamma}, s := s' b_{new, \gamma}^{-1} \mod n. \]
Customer: Link (threat!)

1. Have $c_{old}$.
2. Obtain $T_\gamma, s$ from exchange
3. Compute $X_\gamma = c_{old} T_\gamma$
4. Derive $c_{new,\gamma}$ and $b_{new,\gamma}$ from $X_\gamma$
5. Unblind $s := s' b_{new,\gamma}^{-1}$ mod $n$
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.
Programmable money: Age restrictions
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:

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

<table>
<thead>
<tr>
<th>Privacy</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ID Verification</td>
<td>bad</td>
</tr>
<tr>
<td>Restricted Accounts</td>
<td>bad</td>
</tr>
<tr>
<td>Attribute-based</td>
<td>good</td>
</tr>
</tbody>
</table>
# Age restriction in E-commerce

## Problem:

Verification of minimum age requirements in e-commerce.

## Common solutions:

<table>
<thead>
<tr>
<th>Solution</th>
<th>Privacy</th>
<th>Ext. authority</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ID Verification</td>
<td>bad</td>
<td>required</td>
</tr>
<tr>
<td>2. Restricted Accounts</td>
<td>bad</td>
<td>required</td>
</tr>
<tr>
<td>3. Attribute-based</td>
<td>good</td>
<td>required</td>
</tr>
</tbody>
</table>
Age restriction in E-commerce

Problem:

Verification of minimum age requirements in e-commerce.

Common solutions:

<table>
<thead>
<tr>
<th></th>
<th>Privacy</th>
<th>Ext. authority</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ID Verification</td>
<td>bad</td>
<td>required</td>
</tr>
<tr>
<td>2. Restricted Accounts</td>
<td>bad</td>
<td>required</td>
</tr>
<tr>
<td>3. Attribute-based</td>
<td>good</td>
<td>required</td>
</tr>
</tbody>
</table>

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 subsidiarty
5. is practical and efficient
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 a maximum age
- **Minors** attest their adequate age
- **Merchants** verify the attestations

Note: Scheme is independent of payment service protocol.
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

Note: Scheme is independent of payment service protocol.
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

Note: Scheme is independent of payment service protocol.
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

Note: Scheme is independent of payment service protocol.
Formal Function Signatures

Searching for functions

Commit
Attest
Verify
Derive
Compare

with \( \Omega \), \( P \), \( O \), \( T \), \( B \) sufficiently large sets.

Basic and security requirements are defined later.

Mnemonics:

- \( O = \text{commitments} \)
- \( Q = \text{commitment} \)
- \( P = \text{proofs} \)
- \( T = \text{testations} \)
- \( B = \text{bindings} \)
- \( \beta = \text{binding} \)
Formal Function Signatures

Searching for functions with the following signatures

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

Attest

Verify

Derive

Compare

Mnemonics:

$\mathbb{O} = \text{commitments}, \quad Q = Q\text{-mitment} \ (\text{commitment}), \quad \mathbb{P} = \text{Proofs},$
Formal Function Signatures

Searching for functions with the following signatures

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

Attest : \((m, Q, P) \mapsto T\)  \(\mathbb{N}_M \times \mathbb{O} \times \mathbb{P} \rightarrow \mathbb{T} \cup \{\perp\},\)

Verify

Derive

Compare

Mnemonics:
\(\mathbb{O} = \text{commitments}, \ Q = \text{Q-mitment} \ (\text{commitment}), \ \mathbb{P} = \text{Proofs}, \ \mathbb{P} = \text{Proof}, \)
\(\mathbb{T} = \text{testations}, \ T = \text{aTtestation},\)
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: \((m, Q, P) \mapsto T\) \(\mathbb{N}_M \times \mathbb{O} \times \mathbb{P} \to \mathbb{T} \cup \{\bot\}\),

Verify: \((m, Q, T) \mapsto b\) \(\mathbb{N}_M \times \mathbb{O} \times \mathbb{T} \to \mathbb{Z}_2\),

Derive

Compare

Mnemonics:
\(\mathbb{O} = \text{commitments}\), \(Q = Q\text{-mitment}\) (commitment), \(\mathbb{P} = \text{proofs}\), \(P = \text{proof}\),
\(\mathbb{T} = \text{attestations}\), \(T = \text{attestation}\)
Formal Function Signatures

Searching for functions with the following signatures

<table>
<thead>
<tr>
<th>Function</th>
<th>Signature</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commit</td>
<td>((a, \omega) \mapsto (Q, P))</td>
<td>(\mathbb{N}_M \times \Omega \rightarrow \mathbb{O} \times \mathbb{P}),</td>
</tr>
<tr>
<td>Attest</td>
<td>((m, Q, P) \mapsto T)</td>
<td>(\mathbb{N}_M \times \mathbb{O} \times \mathbb{P} \rightarrow \mathbb{T} \cup {\bot}),</td>
</tr>
<tr>
<td>Verify</td>
<td>((m, Q, T) \mapsto b)</td>
<td>(\mathbb{N}_M \times \mathbb{O} \times \mathbb{T} \rightarrow \mathbb{Z}_2),</td>
</tr>
<tr>
<td>Derive</td>
<td>((Q, P, \omega) \mapsto (Q', P', \beta))</td>
<td>(\mathbb{O} \times \mathbb{P} \times \Omega \rightarrow \mathbb{O} \times \mathbb{P} \times \mathbb{B}),</td>
</tr>
<tr>
<td>Compare</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Mnemonics:
\(\mathbb{O} = cOmmittments\), \(Q = Q\text{-mitment}\) (commitment), \(P = P\text{roofs}\), \(P = P\text{roof}\),
\(T = aT\text{estations}\), \(T = aT\text{estation}\), \(\mathbb{B} = B\text{blindings}\), \(\beta = \beta\text{linding}\).
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 : \((m, Q, P) \mapsto T\) \(\mathbb{N}_M \times \mathbb{O} \times \mathbb{P} \to T \cup \{\bot\},\)

Verify : \((m, Q, T) \mapsto b\) \(\mathbb{N}_M \times \mathbb{O} \times T \to \mathbb{Z}_2,\)

Derive : \((Q, P, \omega) \mapsto (Q', P', \beta)\) \(\mathbb{O} \times \mathbb{P} \times \Omega \to \mathbb{O} \times \mathbb{P} \times B,\)

Compare : \((Q, Q', \beta) \mapsto b\) \(\mathbb{O} \times \mathbb{O} \times B \to \mathbb{Z}_2,\)

Mnemonics:
\(\mathbb{O} = c\text{ommitments}, \ Q = Q\text{-mitment} \ (\text{commitment}), \ \mathbb{P} = \text{Proofs}, \ \mathbb{P} = \text{Proof},\)
\(\mathbb{T} = a\text{TTestations}, \ \mathbb{T} = a\text{Ttestation}, \ \mathbb{B} = \text{Blindings}, \ \beta = \text{blinding}.\)
Formal Function Signatures

Searching for functions with the following signatures

\[
\text{Commit : } (a, \omega) \mapsto (Q, P) \quad N_M \times \Omega \rightarrow O \times P,
\]
\[
\text{Attest : } (m, Q, P) \mapsto T \quad N_M \times O \times P \rightarrow T \cup \{\bot\},
\]
\[
\text{Verify : } (m, Q, T) \mapsto b \quad N_M \times O \times T \rightarrow \mathbb{Z}_2,
\]
\[
\text{Derive : } (Q, P, \omega) \mapsto (Q', P', \beta) \quad O \times P \times \Omega \rightarrow O \times P \times B,
\]
\[
\text{Compare : } (Q, Q', \beta) \mapsto b \quad O \times O \times B \rightarrow \mathbb{Z}_2,
\]

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

Basic and security requirements are defined later.

Mnemonics:
\[
O = \text{commitments}, \quad Q = Q\text{-mitment (commitment)}, \quad P = \text{proofs}, \quad P = \text{Proof},
\]
\[
T = \text{attestations}, \quad T = \text{aTtestation}, \quad B = \text{blindings}, \quad \beta = \text{blinding}.
\]
Age restriction
Naïve scheme
Simple use of Derive() and Compare() is problematic.
Achieving Unlinkability

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

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

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

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

$\implies$ Exchange identifies sequence  
$\implies$ Unlinkability broken
Achieving Unlinkability

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

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

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

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

Sketch:

1. $C$ derives commitments $(Q_1, \ldots, Q_\kappa)$ from $Q_0$ by calling $\text{Derive}()$ with blindings $(\beta_1, \ldots, \beta_\kappa)$

2. $C$ calculates $h_0 := H(H(Q_1, \beta_1)|| \ldots ||H(Q_\kappa, \beta_\kappa))$

3. $C$ sends $Q_0$ and $h_0$ to $E$

4. $E$ chooses $\gamma \in \{1, \ldots, \kappa\}$ randomly

5. $C$ reveals $h_\gamma := H(Q_\gamma, \beta_\gamma)$ and all $(Q_i, \beta_i)$, except $(Q_\gamma, \beta_\gamma)$

6. $E$ compares $h_0$ and $H(H(Q_1, \beta_1)|| \ldots ||h_\gamma|| \ldots ||H(Q_\kappa, \beta_\kappa))$ and evaluates $\text{Compare}(Q_0, Q_i, \beta_i)$.

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

With $\text{DeriveCompare}_\kappa$

- $\mathcal{E}$ learns nothing about $Q_\gamma$,
- 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

Commit(a) → G

DeriveCompare(κ)

C → E

(M, Q) → M

Attest(m, Q, P_a)

Verify(m, Q, T_m)
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

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

Efficacy of attestations

$$\forall n \leq a : \text{Verify}(n, Q, \text{Attest}(n, Q, P)) = 1.$$
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 $G^{FA}_{A}(\lambda)$—Forging an attest:

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

Requirement: Unforgeability of minimum age

$$
\forall A \in \mathcal{A}(\mathbb{N}_M \times \emptyset \times P \rightarrow \mathbb{N}_M \times T) : \text{Pr}\left[G^{FA}_{A}(\lambda) = 1\right] \leq \varepsilon(\lambda)
$$
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):

\[
⟨(q_1, p_1), \ldots, (q_M, p_M)⟩
\]

\( \vec{Q} := (q_1, \ldots, q_M) \) 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 \( i > a \):

\[
\langle (q_1, p_1), \ldots, (q_a, p_a), (q_{a+1}, \perp), \ldots, (q_M, \perp) \rangle
\]

\( \bar{Q} := (q_1, \ldots, q_M) \) is the *Commitment*,

\( \bar{P}_a := (p_1, \ldots, p_a, \perp, \ldots, \perp) \) 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 \( i > a \):

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

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

3. Guardian gives child \( \langle \bar{Q}, \bar{P}_a \rangle \)
Instantiation with ECDSA
Definitions of Attest and Verify

Child has

- ordered public-keys \( \vec{Q} = (q_1, \ldots, q_M) \),
- (some) private-keys \( \vec{P} = (p_1, \ldots, p_a, \bot, \ldots, \bot) \).
Instantiation with ECDSA
Definitions of Attest and Verify

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

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 $\vec{P} = (p_1, \ldots, p_a, \bot, \ldots, \bot)$.

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 $\sigma$
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 $\sigma$.

To Verify a minimum age $m$:

Verify the ECDSA-Signature $\sigma$ with public key $q_m$. 
Child has $\vec{Q} = (q_1, \ldots, q_M)$ and $\vec{P} = (p_1, \ldots, p_a, \bot, \ldots, \bot)$. 

To Derive new $\vec{Q}'$ and $\vec{P}'$: Choose random $\beta \in \mathbb{Z}_g$ and calculate $\vec{Q}' = \beta \ast q_1, \ldots, \beta \ast q_M$, $\vec{P}' = \beta p_1, \ldots, \beta p_a, \bot, \ldots, \bot$. 

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

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

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

$\beta \ast q_i$ 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

\[
\begin{align*}
\vec{Q}' &:= (\beta \ast q_1, \ldots, \beta \ast q_M), \\
\vec{P}' &:= (\beta p_1, \ldots, \beta p_a, \perp, \ldots, \perp)
\end{align*}
\]

Note: \((\beta p_i) \ast G = \beta \ast (p_i \ast G) = \beta \ast q_i\)

\(\beta \ast q_i\) is scalar multiplication on the elliptic curve.

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

To Compare, calculate: \((\beta \ast q_1, \ldots, \beta \ast q_M) \overset{?}{=} (q'_1, \ldots, q'_M)\)
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.
Reminder: GNU Taler Fundamentals

- Coins are public-/private key-pairs \((C_p, c_s)\).
- Exchange blindly signs \(\text{FDH}(C_p)\) with denomination key \(d_p\).
- Verification:
  \[
  1 \overset{?}{=} \text{SigCheck}(\text{FDH}(C_p), D_p, \sigma_p)
  \]
  \((D_p = \text{public key of denomination and } \sigma_p = \text{signature})\)
Integration with GNU Taler

Binding age restriction to coins

To bind an age commitment $Q$ to a coin $C_p$, instead of signing $\text{FDH}(C_p)$, $E$ now blindly signs

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

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

$$1 \overset{?}{=} \text{SigCheck}(\text{FDH}(C_p, H(Q)), D_p, \sigma_p)$$
Integration with GNU Taler

Integrated schemes

\[ \text{Commit}(a) \]

\[ \text{withdraw}, \text{using} \quad \text{FDH}(C_p, H(Q)) \]

\[ \text{deposit} + H(Q) \]

\[ \text{refresh} + \text{DeriveCompare}_\kappa \]

\[ \text{purchase} + (T_m, Q) \]

\[ \text{Attest}(m, Q, P_a) \]

\[ \text{Verify}(m, Q, T_m) \]
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.
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
Blockchain integration: Project Depolymerization
Blockchain based cryptocurrencies

Biggest cryptocurrencies

- **BTC** Bitcoin
- **ETH** Ethereum

Common blockchain limitations

- **Delay** block and confirmation delay
- **Cost** transaction fees
- **Scalability** limited amount of transaction per second
- **Ecological impact** computation redundancy
- **Privacy**
- **Regulatory risk**
Taler
Architecture

Settlement layer
- For Depolymerization: Blockchain!
- Realtime transactions, 1 RTT
- Scalable microtransactions
- Blind signatures (privacy)

Taler payment system
- Deposit money
- Withdraw money
- Deposit coins
- Withdraw coins
- Spend coins

Auditor

Settlement Layer

Exchange

Customer

Merchant

Taler payment system
Taler
Blockchain settlement layer

Taler
Exchange

Depolymerization

Node
Blockchain

Off-chain transactions

Credit
Debit
Challenges

Taler Metadata

▶ Metadata are required to link a wallet to credits and allow merchant to link deposits to debits
▶ Putting metadata in blockchain transactions can be tricky

Blockchain based cryptocurrencies

▶ Blockchain transactions lack finality (fork)
▶ Transactions can be stuck for a long time (mempool)
A fork is when concurrent blockchain states coexist. Nodes will follow the longest chain, replacing recent blocks if necessary during a blockchain reorganization. If a deposit transaction disappears from the blockchain, an irrevocable withdraw transactions would no longer be backed by credit.
Blockchain challenges

Stuck transactions

We want confirmed debits within a limited time frame.

When we trigger a debit with a fee too small, it may not be confirmed in a timely fashion.
Blockchain challenges
Stuck transactions

We want confirmed debits within a limited time frame.

Figure: Bitcoin average transaction fee over 6 months (ychart)

However, transaction fees are unpredictable.
Depolymerization
Architecture

Wire Gateway API

Taler Exchange

HTTP

Wire Gateway

SQL

PostgreSQL

SQL

DLT Full Node

RPC

DLT specific

▶ Common database to store transactions state and communicate with notifications
▶ Wire Gateway for Taler API compatibility
▶ DLT specific adapter
Storing metadata

Bitcoin

Bitcoin - Credit

- Transactions from code
- Only 32B + URI
- \texttt{OP\_RETURN}

Bitcoin - Debit

- Transactions from common wallet software
- Only 32B
- Fake Segwit Addresses
Storing metadata
Ethereum

Smart contract?

- Logs in smart contract is the recommend way (ethereum.org)
- Expensive (additional storage and execution fees)
- Avoidable attack surface (error prone)

Custom input format
Use input data in transactions, usually used to call smart contract, to store our metadata.
As small reorganizations are common, Satoshi already recommended to apply a confirmation delay to handle most disturbances and attacks.
Handling blockchain reorganization

If a reorganization longer than the confirmation delay happens, but it did not remove credits, Depolymerizer is safe and automatically resumes.
Handling blockchain reorganization

If a fork removed a confirmed debit, an attacker may create a conflicting transaction. Depolymerizer suspends operation until lost credits reappear.
Related work

**Centralization - Coinbase off-chain sending**
- Fast and cheap: off chain transaction
  - Trust in Coinbase: privacy, security & transparency

**Layering - Lightning Network**
- Fast and cheap: off-chain transactions
  - Requires setting up bidirectional payment channels
  - Fraud attempts are mitigated via a complex penalty system
Conclusion

Blockchains can be used as a settlement layer for GNU Taler with Depolymerizer.

- Trust exchange operator or auditors
- Fast and cheap
- Realtime, ms latency
- Linear scalability
- Ecological
- Privacy when it can, transparency when it must (avoid tax evasion and money laundering)

Future work:

- Universal auditability, using sharded transactions history
- Smarter analysis, update confirmation delay based on currency network behavior
- Multisig by multiple operator for transactions validation
Future Work & Conclusion
How to support?

Join: https://lists.gnu.org/mailman/listinfo/taler
Develop: https://bugs.taler.net/, https://git.taler.net/
Apply: https://nlnet.nl/propose, https://nlnet.nl/taler
Translate: https://weblate.taler.net/, translation-volunteer@taler.net
Integrate: https://docs.taler.net/
Donate: https://gnunet.org/ev
Partner: https://taler-systems.com/
References:


