Change We Don’t Have To Believe In

Post-Quantumifying Privacy-Preserving Electronic Payments with GNU Taler

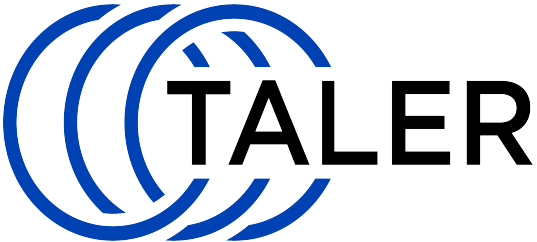
Kathrin Hövelmanns, Özgür Kesim, Tanja Lange,
Jonathan Levin, **Elisa Pioldi**

RWPQC 2026

GNU Taler @ RWPQC

GNU Taler is a private-by-design payment system

- deployed in Switzerland (real world!)
- cryptographically enforced financial/security guarantees
- not yet PQ



A bit more about GNU Taler

- Customers anonymous
- Merchants identified (easily taxable)
- Immune by design to fraud/money laundering
- Easy to use
- Based on open standards
- Uses (mostly) **boring** cryptography
- No data to collect, nothing to leak
- Better energy-efficiency/transaction speed than credit cards

Necessary properties

Not just:

- privacy-by-design *for customers*
- double-spending prevention
- scalability

Also complex, real-world financial constraints:

- Anti-Money-Laundering (AML)
- Know-Your-Customer (KYC)/Know-Your-Business (KYB)
- Counter-terrorism financing (CTF)

The protocols in Taler must ensure all these

Asymmetric privacy

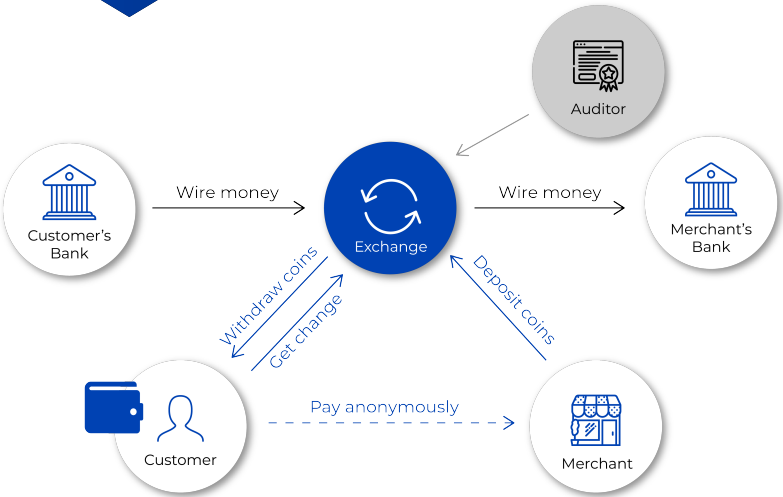
Customer

Merchant

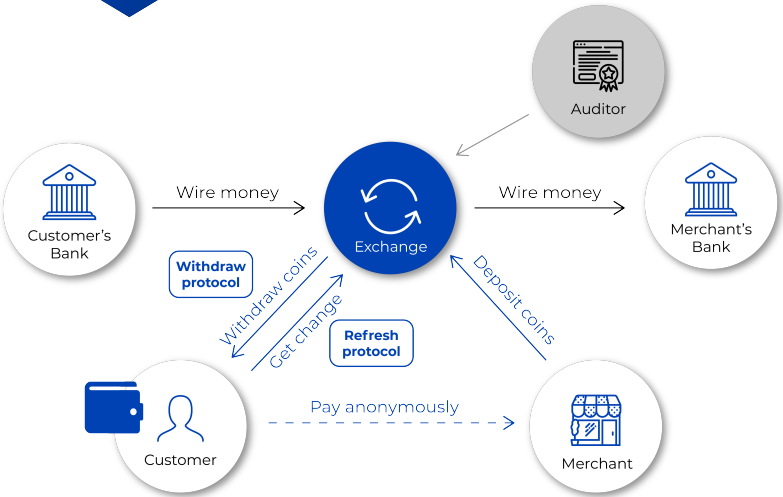
The plan

1. Intro to Taler protocols
2. *Unlinkable* Change
3. Fun making things PQ

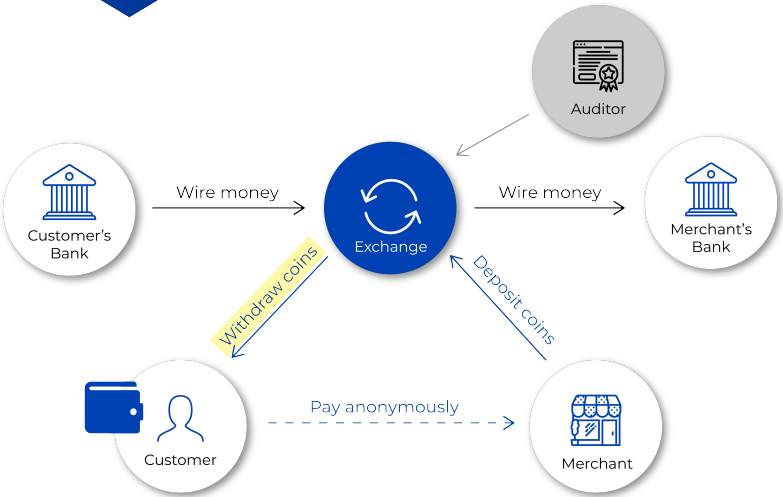
System overview



System overview



System overview: withdrawing coins



System overview: withdrawing coins

Very similar to Chaum's e-cash:

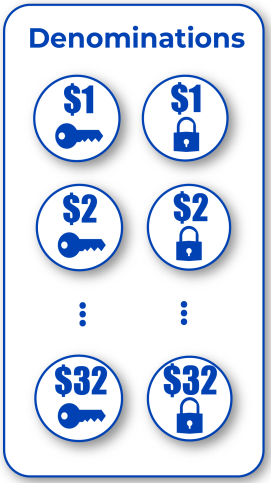
- anonymity from blind signatures

except...

- coins themselves are signing keypairs instead of a serial number, along with a blind signature conferring its value
- coins spent by signing contracts, not by transferring ownership

System overview: withdrawing coins

Blind signatures
key pairs

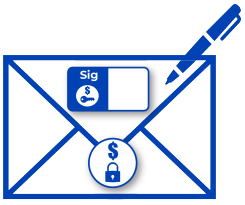


System overview: withdrawing coins

System overview: withdrawing coins

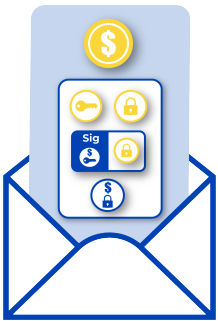
System overview: withdrawing coins

System overview: withdrawing coins

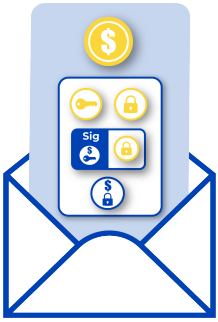


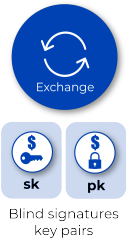
System overview: withdrawing coins

System overview: withdrawing coins



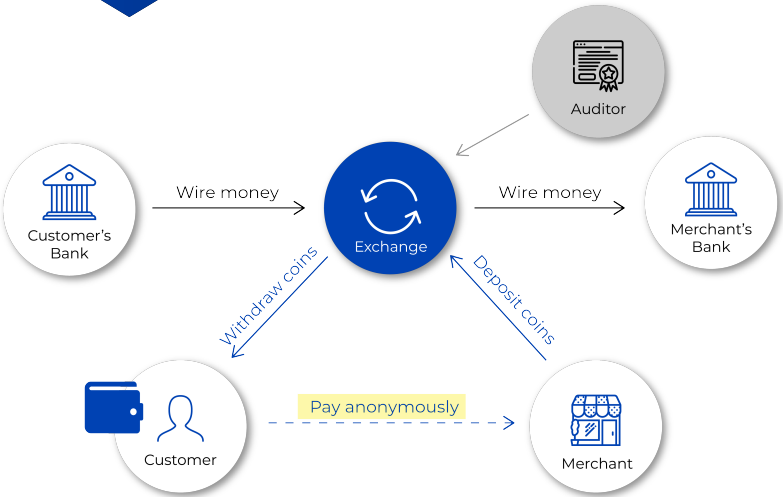
System overview: withdrawing coins



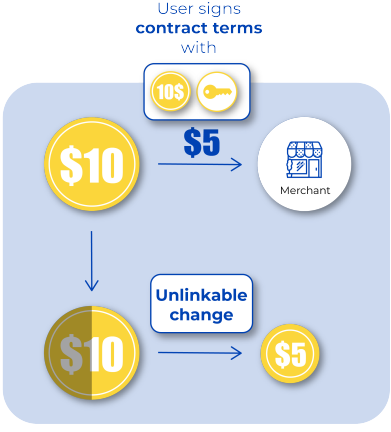


Blind signatures
key pairs

System overview: payment and change



System overview: payment and change



Unlinkable Change

- For anonymity, change must be **unlinkable** from “parent” coins

Unlinkable Change

- For anonymity, change must be **unlinkable** from “parent” coins

BUT!

Unlinkable Change

- For anonymity, change must be **unlinkable** from “parent” coins

BUT!

- If fully unlinkable, money-laundering can occur

Unlinkable Change

- For anonymity, change must be **unlinkable** from “parent” coins

BUT!

- If fully unlinkable, money-laundering can occur

Change must be unlinkable for everyone **except** the coin owner
(i.e. ownership of a coin must never be fully transferred)

Unlinkable Change

- For anonymity, change must be **unlinkable** from “parent” coins

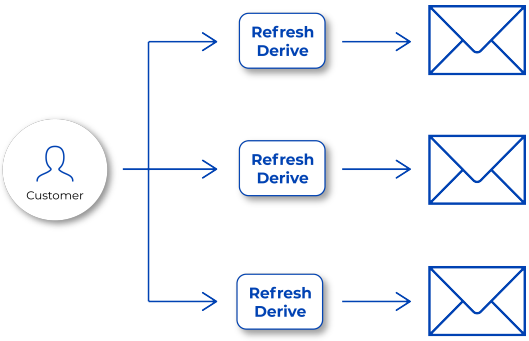
BUT!

- If fully unlinkable, money-laundering can occur

Change must be unlinkable for everyone **except** the coin owner
(i.e. ownership of a coin must never be fully transferred)

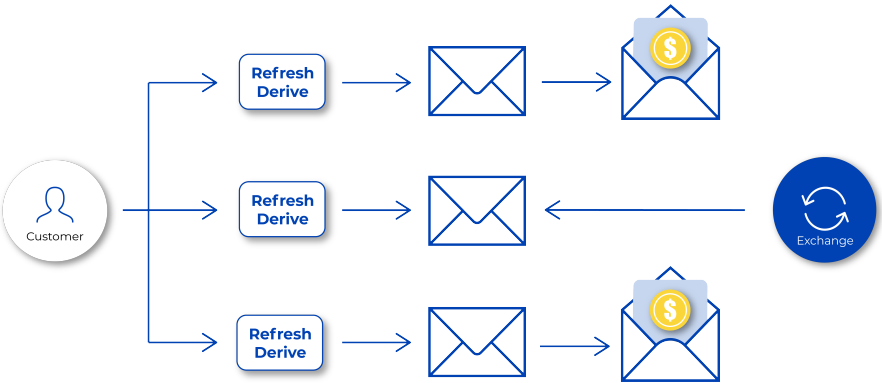
Enter the **Refresh Protocol**

Refresh protocol: cut-and-choose



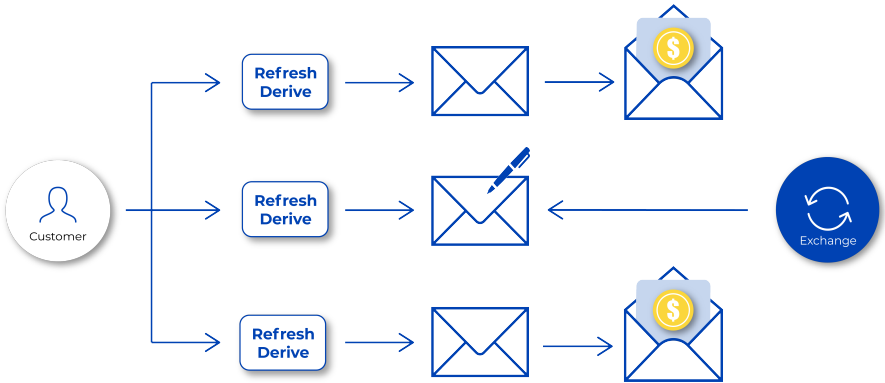
Exchange deducts value before choosing, so 3 shares suffice

Refresh protocol: cut-and-choose



Exchange deducts value before choosing, so 3 shares suffice

Refresh protocol: cut-and-choose

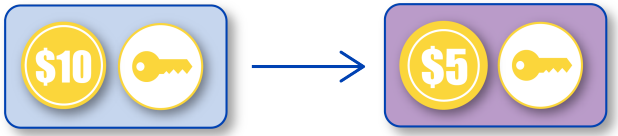


Exchange deducts value before choosing, so 3 shares suffice

RefreshDerive: all-but-owner unlinkability

Given **secret** seed r, denomination pubkey pkD, and coin $C = (C_p, c_s)$:

1. Generate a fresh "linking" ECDH key $L = (L_p, \ell_s)$ from r
2. Compute DH secret x from L and C (**Ed25519 keys can be used for Curve25519 ECDH**)
3. Deterministically obtain new coin $C' = (C'_p, c'_s)$ and blind C'_p from x



RefreshDerive

RefreshDerive(r, pkD, C_p)

1 : $\ell := \text{HKDF}(256, r, \texttt{"link"})$
2 : $L := \text{Curve25519.GetPub}(\ell)$
3 : $x := \text{ECDH}(\ell, C_p)$
4 : $c'_s := \text{HKDF}(256, x, \texttt{"coin"})$
5 : $C'_p := \text{Ed25519.GetPub}(c'_s)$
6 : $\overline{m} := \text{BlindSig.Blind}(C'_p, \text{pkD}, x)$
7 : **return** $\langle L, x, c'_s, C'_p, \overline{m} \rangle$

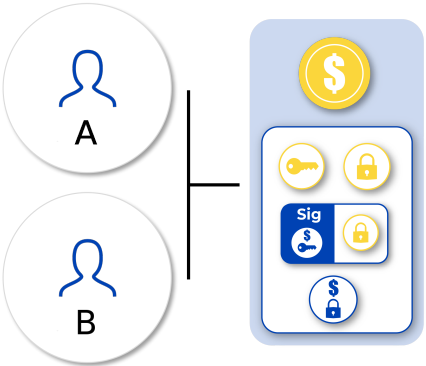
RefreshDerive

RefreshDerive(r, pkD, C_p)

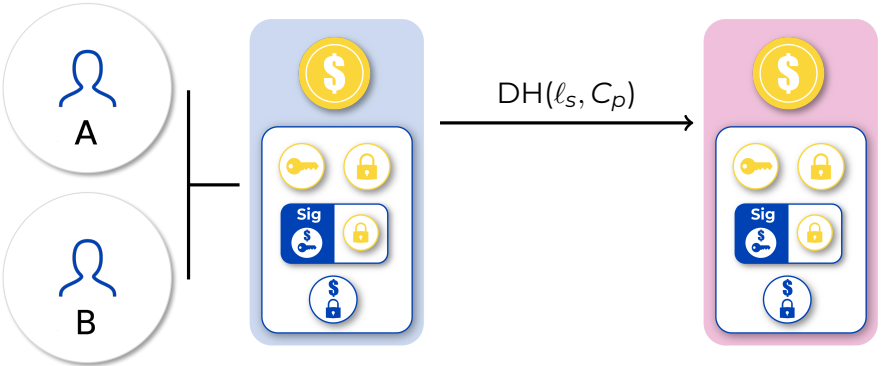
1 : $\ell := \text{HKDF}(256, r, \texttt{"link"})$
2 : $L := \text{Curve25519.GetPub}(\ell)$
3 : $x := \text{ECDH}(\ell, C_p)$
4 : $c'_s := \text{HKDF}(256, x, \texttt{"coin"})$
5 : $C'_p := \text{Ed25519.GetPub}(c'_s)$
6 : $\overline{m} := \text{BlindSig.Blind}(C'_p, \text{pkD}, x)$
7 : **return** $\langle L, x, c'_s, C'_p, \overline{m} \rangle$

Knowledge of x sufficient to compute C'

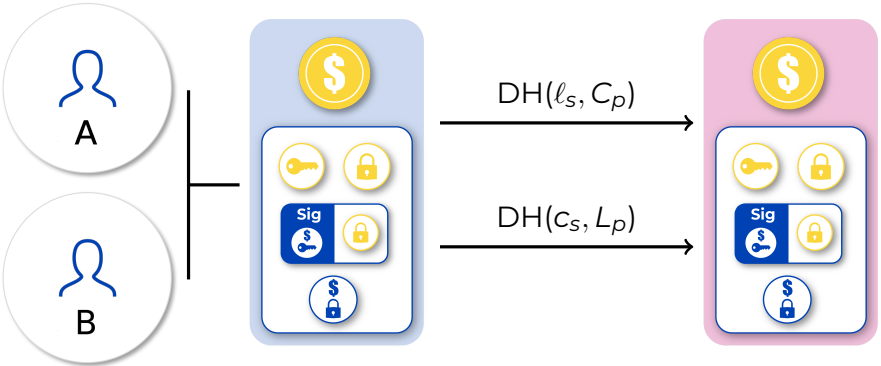
Re-obtaining a refreshed coin



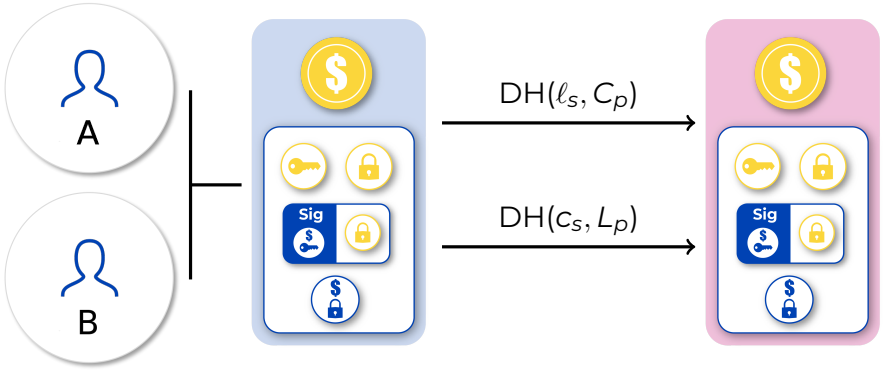
Re-obtaining a refreshed coin



Re-obtaining a refreshed coin



Re-obtaining a refreshed coin



(L_p publicly available from exchange $\Rightarrow$ anyone with c_s can compute x)

Post-Quantum Refresh?

Refresh protocol exploits ability of Ed25519 keys to do Curve25519 ECDH.

How do we make this **Post-Quantum**?

Post-Quantum Refresh?

Refresh protocol exploits ability of Ed25519 keys to do Curve25519 ECDH.

How do we make this **Post-Quantum**?

*We need to keep signatures (e.g., for spending)

PQ Refresh: Idea #1 — hybrid coins?

Can we just add a DH/KEM key to coins (with e.g. CSIDH, ML-KEM, etc.)?

Suppose we generate

1. $C^{sig} \leftarrow \text{Sig.Keygen}(), C^{DH} \leftarrow \text{DH.Keygen}()$
2. Define $C = (C^{sig}, C^{DH})$, exchange signs $(C_p^{sig} || C_p^{DH})$.

PQ Refresh: Idea #1 — hybrid coins?

Can we just add a DH/KEM key to coins (with e.g. CSIDH, ML-KEM, etc.)?

Suppose we generate

1. $C^{sig} \leftarrow \text{Sig.Keygen}(), C^{DH} \leftarrow \text{DH.Keygen}()$
2. Define $C = (C^{sig}, C^{DH})$, exchange signs $(C_p^{sig} || C_p^{DH})$.

Eh...

PQ Refresh: Idea #1 — hybrid coins?

Can we just add a DH/KEM key to coins (with e.g. CSIDH, ML-KEM, etc.)?

Suppose we generate

1. $C^{sig} \leftarrow \text{Sig.Keygen}(), C^{DH} \leftarrow \text{DH.Keygen}()$
2. Define $C = (C^{sig}, C^{DH})$, exchange signs $(C_p^{sig} || C_p^{DH})$.

Eh...

Not ideal. # round-trips and message size both ↑, because Exchange must ensure coins are correct before blind-signing.

PQ Refresh: Idea #2 — Signature-only?

Can we use a signature on a public message as the seed for C'?

PQ Refresh: Idea #2 — Signature-only?

Can we use a signature on a public message as the seed for C'?

Benefits:

PQ Refresh: Idea #2 — Signature-only?

Can we use a signature on a public message as the seed for C'?

Benefits:

- Only the owner of c_S can generate valid signatures

PQ Refresh: Idea #2 — Signature-only?

Can we use a signature on a public message as the seed for C'?

Benefits:

- Only the owner of c_S can generate valid signatures
- Exchange can verify correctness during cut-and-choose

PQ Refresh: Idea #2 — Signature-only?

Can we use a signature on a public message as the seed for C'?

Benefits:

- Only the owner of c_S can generate valid signatures
- Exchange can verify correctness during cut-and-choose
- Deterministic signatures would allow users to re-obtain coins

PQ Refresh: Idea #2 — Signature-only?

Can we use a signature on a public message as the seed for C'?

Benefits:

- Only the owner of c_s can generate valid signatures
- Exchange can verify correctness during cut-and-choose
- Deterministic signatures would allow users to re-obtain coins

So what's the problem?

PQ Refresh: Idea #2 — Signature-only?

~~Deterministic signatures would allow users to re-obtain coins~~

Wrong!

PQ Refresh: Idea #2 — Signature-only?

~~Deterministic signatures would allow users to re-obtain coins~~

Wrong!

What we *actually* need are **(almost)-unique signatures**!

t-unique signatures

Definition

A digital signature scheme is almost-unique or *t*-unique if the set of possible valid signatures for a fixed (msg, pk) pair is bound by some constant t.

t-unique signatures

Definition

A digital signature scheme is almost-unique or t-unique if the set of possible valid signatures for a fixed (msg, pk) pair is bound by some constant t.

Example: VRFs

Verifiable Random Functions (VRFs): $t = 1$.

Example: Rabin-Williams

s is a signature on m satisfying $s^2 \equiv m \mod N$, where $N = pq$: $t = 4$.

Example: (single-tree) XMSS

t equals the number of leaves

t-unique signatures

*All signatures on *m* are iterable*

PQ Refresh: almost-unique signatures

What can happen if we don't use a t-unique signature in RefreshDerive to generate $C' = (C'_p, c'_s)$?

PQ Refresh: almost-unique signatures

What can happen if we don't use a t-unique signature in RefreshDerive to generate $C' = (C'_p, c'_s)$?

1. A shares $C = (C_p, c_s)$ with B

PQ Refresh: almost-unique signatures

What can happen if we don't use a t-unique signature in RefreshDerive to generate $C' = (C'_p, c'_s)$?

1. A shares $C = (C_p, c_s)$ with B
2. B generates C' from a signature s_0 with c_s on new message m.

PQ Refresh: almost-unique signatures

What can happen if we don't use a t-unique signature in RefreshDerive to generate $C' = (C'_p, c'_s)$?

1. A shares $C = (C_p, c_s)$ with B
2. B generates C' from a signature s_0 with c_s on new message m.
3. If A cannot efficiently compute the **same** signature s_0 on m, A cannot re-obtain C'.

PQ Refresh: almost-unique signatures

What can happen if we don't use a t-unique signature in RefreshDerive to generate $C' = (C'_p, c'_s)$?

1. A shares $C = (C_p, c_s)$ with B
2. B generates C' from a signature s_0 with c_s on new message m.
3. If A cannot efficiently compute the **same** signature s_0 on m, A cannot re-obtain C'.

Result: A has effectively transferred ownership of C' to B.

NewRefreshDerive

Given **public** seed r, denomination key pkD, and coin $C = (C_p, c_s)$:

1. Compute t-unique signature s on m =(“Refresh”,C_p, r, pkD) with c_s
2. Compute new coin keypair $C' = (C'_p, c'_s)$ and blind C'_p using s

NewRefreshDerive

Given **public** seed r, denomination key pkD, and coin $C = (C_p, c_s)$:

1. Compute t-unique signature s on m =(“Refresh”,C_p, r, pkD) with c_s
2. Compute new coin keypair $C' = (C'_p, c'_s)$ and blind C'_p using s

NewRefreshDerive$(r, c_s, C_p, \text{pkD})$

1 : $m := (\text{“Refresh”} \| C_p \| r \| \text{pkD})$
2 : $s := \text{USig.Sign}(c_s, m)$
3 : $x := HKDF(256, s)$
4 : $(c'_s, C'_p) := \text{USig.KeyGen}(x)$
5 : $\overline{m} := \text{BlindSig.Blind}(C'_p, \text{pkD}, x)$
6 : **return** $\langle s, c'_s, C'_p, \overline{m} \rangle$

RefreshDerive(r, pkD, C_p)

1 : $\ell := \text{HKDF}(256, r, \texttt{"link"})$
2 : $L := \text{Curve25519.GetPub}(\ell)$
3 : $x := \text{ECDH}(\ell, C_p)$
4 : $c'_s := \text{HKDF}(256, x, \texttt{"coin"})$
5 : $C'_p := \text{Ed25519.GetPub}(c'_s)$
6 : $\overline{m} := \text{BlindSig.Blind}(C'_p, \text{pkD}, x)$
7 : **return** $\langle L, x, c'_s, C'_p, \overline{m} \rangle$

Takeaways

- We redesigned a pre-quantum cryptographic protocol to achieve the same security/functionality guarantees with existing PQ primitives
- Learned that updating an apparently simple pre-quantum protocol to post-quantum cryptography is not always trivial
- Not all the desired properties of pre-quantum cryptography may be available in PQ
- Formalized the notion of t-unique signatures and highlighted their application in Taler