Anonymous usage tokens from Curve Trees

February 2, 2024

The thinking behind the title will be explained only in Section 2. Initially, we cover specificially how a ZKP of knowledge of commitment opening, with a key image, works.

1 Proof of Knowledge of Representation with Key Image

(Caveat: there is nothing original here, this is just an application of standard $\Sigma-$ protocol techniques; *however*, it needs to be checked that there is no error in this protocol, see also Section 1.4).

The term "representation" here is in agreement with what is seen in certain other places e.g. [2], but it can also just be described as "proof of knowledge of opening of a Pedersen commitment".

Setting: a DLOG-hard, prime order, additively homomorphic group $(\mathbb{G}, \cdot)$ of order n. Additive notation is used.
Goal: a Prover wants to prove that, for commitments $(\in \mathbb{G})$ C, C_2, that they present, the following hold true:

- the Prover knows a secret witness (x, δ) such that the commitment C is represented by those values with respect to generators G, H: $C = xG+\delta H$.
- $C_2 = xJ$ for a third generator J.

(For the generators G, H, J referred to, it's understood that they are elements of $\mathbb{G}$, and that their relative DLOGs are unknown).

1.1 Setup

1.1.1 Setup for prover

- Generate scalars $\in \mathbb{Z}_n$: x, δ - but note, the prover may already possess the x such that it is the pre-image for a public key $P = xG$
- Generate the commitment $C = xG + \delta H$
- Generate the "key image" commitment $C_2 = xJ$
- Agree on a context-specific message m.

1.1.2 Setup for Verifier

- Agree on a context-specific message m.
- Receive from prover the claimed values C, C_2.

1.2 Prover steps

- Choose values $s, t \in \mathbb{Z}_n$
- Calculate $R_1 = sG + tH$
- Calculate $R_2 = sJ$
- Calculate the Fiat-Shamir hash challenge: $e = \mathbb{H}(R_1, R_2, C, C_2, m)$
- Calculate response 1: $\sigma_1 = s + ex$
- Calculate response 2: $\sigma_2 = t + e\delta$
- Send to Verifier: $(R_1, R_2, \sigma_1, \sigma_2)$

1.3 Verifier steps

Verifier receives the above message $(R, R_2, \sigma_1, \sigma_2)$.

- Calculate the Fiat-Shamir hash challenge: $e = \mathbb{H}(R_1, R_2, C, C_2, m)$
- Check that $\sigma_1 G + \sigma_2 H \stackrel{?}{=} R_1 + eC$, reject if not
- Check that $\sigma_1 J \stackrel{?}{=} R_2 + eC_2$, reject if not.
- Accept.

1.4 Security

The above techniques are standard application of $\Sigma-$protocols. More specifically, this is a variant on the standard Schnorr-based DLEQ proof, where we prove knowledge of representation (a Pedersen commitment), instead of only the secret x, for the base generator G.
But some proof that the properties of **correctness, soundness and zero-knowledgeness** hold, should be presented here.

2 In connection with ZK set membership

The above protocol can be "attached" to the zero knowledge set membership protocol "Curve Trees"[1]. In the notation of that paper, our C here is $\hat{C}$, i.e the **rerandomized** value of an underlying curve element, which, in the EC group setting, can mean a public key P such that its private key is the x referred to in the above protocol.
By running the **Select-And-Rerandomize** algorithm on a curve tree over a set of such public keys P_i, again using the notation of that paper, and generating a proof Π_{SR} for a prover-provided blinded value $\hat{C}$, connecting it to a root rt of

the tree, and then running the above algorithm to generate a "key-image" C_2 as per above, we can allow a verifier to know not only that the rerandomized public key was indeed in the tree, but that no further use of the same key is possible, without re-use of the same key image.

Relation to Vcash: It should be noted in this context, that the goal described here (the non-"double-spendability" property for a commitment) is already achieved, as it must be, for the theoretical construct of "Vcash" in [1], so that coins (which are really commitments) cannot be spent twice. But Vcash requires users to actively "mint" new coins, and then create proofs of valid spending. If we want to create trees non-interactively (i.e. without requesting actions from the owners of the "coins", in this case just public keys P_i, and then record their usage locally, this more basic "key image" idea might make more sense (in certain contexts).

To give the concrete example intended: imagine creating the Curve Tree from a large set of existing Bitcoin utxos (taproot specifically, so that the public keys are exposed without owner interaction; see the concept of "spontaneous" linkable ring signatures), it's necessary that tree construction is "local" to the prover and verifier separately, and that the update of key images can be local to the verifier (if they are managing access to their own resources only; for a more complex scenario where resource usage is being restricted across a distributed group, then some kind of global ledger or gossip network may be needed).

3 References

1. "Curve Trees: Practical and Transparent Zero-Knowledge Accumulators", Campanelli, Hall-Andersen, Kamp, 2022 `https://eprint.iacr.org/2022/756`
2. "A Graduate Course in Applied Cryptograhy" v0.6, Boneh, Shoup, 2023 `https://toc.cryptobook.us/book.pdf`