1 Related Work

1.1 Laser Beam

Laser Beam is an adaptation of the Lightning Network for the Mimblewimble protocol, to be implemented for Beam ([1, 2, 3]). At the time of writing of this report the specifications were far advanced, but still work in progress. Beam has a working demonstration in their mainnet repository, which at this stage demonstrates off-chain transactions in a single channel between two parties [4]. According to the Request for Comment (RFC) documents, they do not plan to support multiparty (more than two) payment channels, but rather implement routing across different payment channels in the Lightning Network style.

Their version of a multisig is actually a 2-of-2 multi-party UTXO, where each party keep their share of the blinding factor of the Pedersen commitment, $C(v, k_1{+}k_2) = (vH{+}(k_1{+}k_2)G)$, secret. (Refer to **Appendix A: Notation Used**). The multi-party commitment is accompanied by a single multi-party Bulletproof range proof[1], where the individual shares of the blinding factor are used to create the combined range proof [5].

Funding transaction: The parties collaborate to create the multi-party UTXO (i.e. commitment and associated multi-party range proof), combined funding transaction (on-chain) and an initial refund transaction for each party (off-chain). All refund transactions have a relative time lock in its kernel, referencing the kernel of the original combined funding transaction, which have to be confirmed on the blockchain.

The initial funding transaction between Alice and Bob is depicted in (1) to (2). The capitalized use of R and P in (2) denote public nonce and public blinding factor respectively, f is the fee and $\mathcal{X}$ is the excess. The lock height h_0 corresponds to the current blockchain height.

$$-\text{Inputs.}0_{AB} + \text{MultiSig.}0 + \text{fee} = \mathcal{X}_{0_{T1}} \tag{1}$$

$$-\left(v_a H + k_{a_{T1}} G\right) - \left(v_b H + k_{b_{T1}} G\right) + \left(v_0 H + \left(k_{0_a} + k_{0_b}\right) G\right) + fH = \mathcal{X}_{0_{T1}}$$

$$\text{Challenge: } e_{0_{T1}} = \mathcal{H}\left(R_{N_{A0}} + R_{N_{B0}} \parallel P_{N_{A0}} + P_{N_{B0}} \parallel f \parallel h_0\right) \tag{2}$$

Alice and Bob also need to set up their own respective refund transaction so they can be compensated should the channel never be used; this is done via a refund procedure. A refund procedure (off-chain) consists of 4 parts, whereby each user creates 2 transactions, one kept partially secret and the other shared. Each partially secret transaction creates a different intermediate multiparty UTXO, which are then used as input in two shared transactions, to pay the same set of outputs to each participant.

All consecutive refund procedures works in exactly the same manner. The N_{th} refund procedure is shown below.

Refund procedure part 1 - Alice: Alice and Bob set up Alice's intermediate multisig funding transaction, spending the original funding multisig UTXO. The lock height h_N corresponds to the current blockchain height. Alice does not share the kernel and thereby keeps her part of the final aggregated signature hidden.

[1] This is not an aggregated Bulletproof range proof.

$$-\text{MultiSig}.0 + \text{MultiSig}.N_A + \text{fee} = \mathcal{X}_{N_{A1}} \tag{3}$$

$$-\left(v_0 H + \left(k_{0_a} + k_{0_b}\right) G\right) + \left(v_0 H + \left(\hat{k}_{N_a} + k_{N_b}\right) G\right) + fH = \mathcal{X}_{N_{A1}}$$

$$\text{Challenge: } e_{N_{A1}} = \mathcal{H}\left(R_{N_{AA1}} + R_{N_{AB1}} \parallel P_{N_{AA1}} + P_{N_{AB1}} \parallel f \parallel h_N\right) \tag{4}$$

$$\text{Secret: } s_{N_{AA1}} = r_{N_{AA1}} + e_{N_{A1}} \cdot \left(k_{0_a} + \hat{k}_{N_a}\right) \tag{5}$$

$$\text{Shared: } s_{N_{AB1}} = r_{N_{AB1}} + e_{N_{A1}} \cdot \left(k_{0_b} + k_{N_b}\right) \tag{6}$$

$$\text{Signature tuple: } \left(s_{N_{AA1}} + s_{N_{AB1}}, R_{N_{AA1}} + R_{N_{AB1}}\right) \tag{7}$$

$$\text{Kernel of this transaction, kept secret: } \mathcal{K}_{N_{A1}} \tag{8}$$

Refund procedure part 2 - Alice: Alice and Bob set up a refund transaction, which Alice controls, with a relative time lock h_{rel} to the intermediate funding transaction's kernel. She shares the final kernel with Bob.

$$-\text{MultiSig}.N_A + \text{Outputs}.N + \text{fee} = \mathcal{X}_{N_{A2}} \tag{9}$$

$$-\left(v_0 H + \left(\hat{k}_{N_a} + k_{N_b}\right) G\right) + \left(v_{a_N} H + k_{a_N} G\right) + \left(v_{b_N} H + k_{b_N} G\right) + fH = \mathcal{X}_{N_{A2}}$$

$$\text{Challenge: } e_{N_{A2}} = \mathcal{H}\left(R_{N_{AA2}} + R_{N_{AB2}} \parallel P_{N_{AA2}} + P_{N_{AB2}} \parallel f \parallel \mathcal{H}\left(\mathcal{K}_{N_{A1}}\right) \parallel h_{rel}\right) \tag{10}$$

Refund procedure part 1 - Bob: Alice and Bob set up Bob's intermediate multisig funding transaction, also spending the original funding multisig UTXO. The lock height h_N again corresponds to the current blockchain height. Bob does not share the kernel and thereby keeps his part of the final aggregated signature hidden.

$$-\text{MultiSig}.0 + \text{MultiSig}.N_B + \text{fee} = \mathcal{X}_{N_{B1}} \tag{11}$$

$$-\left(v_0 H + \left(k_{0_a} + k_{0_b}\right) G\right) + \left(v_0 H + \left(k_{N_a} + \hat{k}_{N_b}\right) G\right) + fH = \mathcal{X}_{N_{B1}}$$

$$\text{Challenge: } e_{N_{B1}} = \mathcal{H}\left(R_{N_{BA1}} + R_{N_{BB1}} \parallel P_{N_{BA1}} + P_{N_{BB1}} \parallel f \parallel h_N\right) \tag{12}$$

$$\text{Shared: } s_{N_{BA1}} = r_{N_{BA1}} + e_{N_{B1}} \cdot \left(k_{0_a} + k_{N_a}\right) \tag{13}$$

$$\text{Secret: } s_{N_{BB1}} = r_{N_{BB1}} + e_{N_{A1}} \cdot \left(k_{0_b} + \hat{k}_{N_b}\right) \tag{14}$$

$$\text{Signature tuple: } \left(s_{N_{BA1}} + s_{N_{BB1}}, R_{N_{BA1}} + R_{N_{BB1}}\right) \tag{15}$$

$$\text{Kernel of this transaction, kept secret: } \mathcal{K}_{N_{B1}} \tag{16}$$

Refund procedure part 2 - Bob: Alice and Bob set up a refund transaction, which Alice controls, with a relative time lock h_{rel} to the intermediate funding transaction. She shares the final kernel with Bob.

$$-\text{MultiSig}.N_B + \text{Outputs}.N + \text{fee} = \mathcal{X}_{N_{B2}} \tag{17}$$

$$-\left(v_0 H + \left(k_{N_a} + \hat{k}_{N_b}\right) G\right) + \left(v_{a_N} H + k_{a_N} G\right) + \left(v_{b_N} H + k_{b_N} G\right) + fH = \mathcal{X}_{N_{B2}}$$

$$\text{Challenge: } e_{N_{B2}} = \mathcal{H}\left(R_{N_{BA2}} + R_{N_{BB2}} \parallel P_{N_{BA2}} + P_{N_{BB2}} \parallel f \parallel \mathcal{H}\left(\mathcal{K}_{N_{B1}}\right) \parallel h_{rel}\right) \tag{18}$$

Revoke previous refund: Whenever the individual balances in the channel changes, a new refund procedure are negotiated, revoking previous agreements. Revoking refund transactions

involve revealing blinding factor shares for the intermediate multi-party UTXOs, thereby nullifying its further use. After the four parts of the refund procedure have been concluded successfully, the previous round's blinding factor shares $\hat{k}_{(N-1)}$ are revealed to each other (in an atomic fashion), in order to revoke the previous agreement.

$$\text{MultiSig.}\,(N-1)_A: \quad \left(v_0 H + \left(\hat{k}_{(N-1)_a} + k_{(N-1)_b}\right) G\right)\{\text{Alice's commitment}\} \tag{19}$$
$$\hat{k}_{(N-1)_a}:\{\text{Alice shares with Bob}\}$$
$$\text{MultiSig.}\,(N-1)_B: \quad \left(v_0 H + \left(k_{(N-1)_a} + \hat{k}_{(N-1)_b}\right) G\right)\{\text{Bob's commitment}\} \tag{20}$$
$$\hat{k}_{(N-1)_b}:\{\text{Bob shares with Alice}\}$$

Although each party will now have their counterparty's blinding factor share in the counterparty's intermediate multiparty UTXO, they will still not be able to spend it, because the transaction kernel is still kept secret by the counterparty.

Punishment transaction: If a counterparty decides to broadcast a revoked set of refund transactions, and the honest party are actively monitoring the blockchain and able to detect the attempted foul play, a punishment transaction can immediately be constructed before the relative time lock h_{rel} expires. Whenever the counterparty's MultiSig. $(N-1)$ becomes available in the blockchain, the honest party can spend all the funds to their own output, because they know the total blinding factor.

Channel closure: Whenever the parties agree to a channel closure, the original on-chain multi-party UTXO is spend to their respective outputs in a collaborative transaction. In case a single party decides to close the channel unilaterally due to whatever reason, their latest refund transaction is broadcasted, effectively closing the channel.

Opening a channel requires one collaborative funding transaction on the blockchain. Closing a channel involves for each party to broadcast their respective portion of the refund transaction to the blockchain, or collaborate to broadcast a single settlement transaction. A round trip open channel, multiple off-chain spending and close channel thus involves at most three on-chain transactions.

2 Appendices

Appendix A: Notation Used

- Let p be a large prime number.
- Let $\mathbb{Z}_p$ denote the ring of integers *modulo* p .
- Let $\mathbb{F}_p$ be the group of elliptic curve points.
- Let $G \in \mathbb{F}_p$ be a random generator point (base point) and let $H \in \mathbb{F}_p$ be specially chosen so that the value x_H to satisfy $H = x_H G$ cannot be found except if the Elliptic Curve Discrete Logarithm Problem (ECDLP) is solved.
- Let commitment to value $v \in \mathbb{Z}_p$ be determined by calculating $C(v,k) = (vH + kG)$, which is called the Elliptic Curve Pedersen Commitment (Pedersen Commitment), with $k \in \mathbb{Z}_p$ (the blinding factor) a random value.
- Let scalar multiplication be depicted by "$\cdot$", e.g. $e \cdot (vH + kG) = e \cdot vH + e \cdot kG$.

References

[1] The Beam Team, GitHub: "Lightning Network · BeamMW/beam Wiki"" [online]. Available: https://github.com/BeamMW/beam/wiki/Lightning-Network. Date accessed: July 5, 2019

[2] F. Jahr, "Beam - Lightning network position paper. (v 1.0)" [online]. Available: https://docs.beam.mw/Beam_lightning_network_position_paper.pdf. Date accessed: July 4, 2019.

[3] F. Jahr, GitHub: "fjahr/lightning-mw, Lightning Network Specifications" [online]. Available: https://github.com/fjahr/lightning-mw. Date accessed: July 4, 2019.

[4] The Beam Team, GitHub: "beam/node/laser_beam_demo at master · BeamMW/beam" [online]. Available: https://github.com/BeamMW/beam/tree/master/node/laser_beam_demo. Date accessed: July 5, 2019.

[5] The Beam Team, GitHub: "beam/ecc_bulletproof.cpp at mainnet · BeamMW/beam" [online]. Available: https://github.com/BeamMW/beam/blob/mainnet/core/ecc_bulletproof.cpp. Date accessed: July 5, 2019.