LRC-20: Scalable and Fast Tokenization on Lightning

Akita Mia

April 16, 2024
v0.1

Abstract

The LRC-20 standard embeds additional information into Bitcoin transactions by tweaking public keys in Bitcoin outputs, enabling a novel, compact, and efficient approach for tokenized assets to be issued and transferred on Bitcoin[1] and Lightning[2]. LRC-20 nodes communicate on a peer-to-peer layer in order to recognize and validate LRC-20 transactions. Due to the simplicity and standardization of the key-tweaking mechanism, LRC-20 tokens can be seamlessly ported onto the Lightning Network, delivering faster and cheaper LRC-20 transactions. LRC-20 transaction amounts can be blinded for additional privacy, and issuers can implement freezing capabilities if they choose to.

1 Introduction

The LRC-20 standard supports the issuance and use of fungible and semi-fungible tokenized assets on Bitcoin and the Lightning Network. Through a simple and elegant design, LRC-20 delivers an efficient and scalable approach to tokenization. LRC-20 should not only improve blockspace usage on Bitcoin L1 relative to BRC-20[3], but also enable a number of use cases and applications that do not exist today.

The fundamental premise of the LRC-20 protocol is that extra information can be embedded into the Bitcoin blockchain by tweaking public keys in Bitcoin transaction outputs. From an outsider's perspective, LRC-20 transactions are indistinguishable from regular Bitcoin transactions. LRC-20 nodes construct LRC-20 transactions and submit them to the Bitcoin blockchain. The nodes communicate on a peer-to-peer gossip layer to exchange data in order to validate LRC-20 transactions.

Because the key tweaking mechanism is compatible with any output script, all LRC-20 tokens can be used in Lightning Network channels. Thus, LRC-20 tokens are designed to inherit all the existing features of the Lighting Network to achieve faster and cheaper LRC-20 payments.

Existing tokenization on Bitcoin standards, including Taproot Assets[4], BRC-20, and RGB[5], provide similar functionality to LRC-20. But LRC-20 has a more simple technical design relative to Taproot Assets and RGB, and, critically, a substantially lower on-chain footprint than BRC-20. The LRC-20 protocol also introduces additional, optional features that may be important for select use cases:

1. **Privacy**: LRC-20 transaction amounts can be blinded via range proofs, like Bulletproofs;
2. **Option for freezing and unfreezing**: an LRC-20 token issuer can opt into the ability to freeze and unfreeze LRC-20 tokens.

The rest of the paper is organized as follows: Section 2 describes the general key-tweaking standard for LRC-20 tokens; Section 3 describes how to identify and validate LRC-20 transactions based on the transaction type (including transfer, issuance, and freezing/unfreezing of tokens); Section 4 discusses how the LRC-20 standard works on the Lightning Network; the paper concludes with a discussion on novel use cases like on-chain non-interactive swaps.

2 Key Tweaking

The LRC-20 protocol changes how Bitcoin outputs are constructed and commits LRC-20 transaction data into Bitcoin transactions. An LRC-20 information tuple is embedded in every transaction output. This information tuple, called a pixel, denoted as PX, is composed of the token type, denoted as UV, and token amount, denoted as Y, i.e. $PX = (Y, UV)$. Amounts, Y, are 64-bit integers. Token types, UV, are secp256k1 public keys. The UV public key is the token issuer's public key (more on token issuance is discussed in Section 3).

For every output, a pixel key, denoted as PXK, is constructed based on the pixel and the recipient's public key pk_b (b as in Bob, the recipient) with the same key-tweaking technique as in BIP 341[6]. Explicitly,

$$PXK = hash(hash(hash(Y), UV), pk_b) * G + pk_b, \tag{1}$$

where the hash function is a tagged SHA256 hash, and G is the generator point of the Elliptic curve.

The key tweaking standard is defined generally to support any output type. If the output script contains multiple pubkeys, the first pubkey is always chosen to be tweaked according to the pixel key formula (in place of the pk_b in the formula). All other pubkeys are ignored from the perspective of the LRC-20 protocol.

As an additional privacy measure, the transaction amount Y can be concealed via range proofs. The protocol can substitute the actual amount with a range proof hash that commits to the amount. This method allows for the confidentiality of transaction amounts while still ensuring adherence to transaction validity rules.

With only the pixel key of a UTXO, the recipient, Bob, cannot yet identify the LRC-20 information contained therein. Alice needs to provide Bob the pixel information and the recipient pubkey, i.e. a (Y, UV, pk_b)-tuple called a pixel proof, so that Bob can recognize how many tokens of a given LRC-20 type he has received. Bob can then verify the pixel in the output by computing the pixel key according to Equation 1 (along with any additional necessary computation) and check whether it matches the public key found in the transaction output that Alice points to. When spending the output, Bob signs with a tweaked private key:

$$k = hash(hash(hash(Y), UV), pk_b) + sk_b,$$

where sk_b is Bob's private key.

3 Transaction Validation

LRC-20 transactions must satisfy both Bitcoin and LRC-20 validity requirements. An invalid Bitcoin transaction with valid LRC-20 data will never be confirmed on the blockchain.

Besides being a valid Bitcoin transaction, a valid LRC-20 transaction also needs to meet the additional requirements based on the transaction type: anyone can submit a transfer transaction as long as it satisfies the token conservation rule; only token issuers can submit issuance transactions and (if opted in) freezing/unfreezing transactions. If the transaction adheres to Bitcoin rules but violates LRC-20 rules, the transaction will be confirmed on Bitcoin, but the LRC-20 transaction will be deemed invalid and the LRC-20 tokens in the transaction destroyed.

3.1 Transfer Transaction

A valid LRC-20 transfer transaction must satisfy the **token conservation rule**: for every LRC-20 token type within a transaction, the total amount across all outputs must be equal to the total amount across all inputs. In a valid LRC-20 transfer transaction, every output must contain a pixel proof in order to be validated. Outputs that do not need to contain an LRC-20 token must still commit to $Y = 0$. Without an LRC-20 commitment and its proof for every output, there could be unrevealed pixels hidden in an output, which could violate the token conservation rule. Therefore, LRC-20 nodes will not consider the transaction invalid until a proof is provided for every output.

3.2 Token Issuance

To create new LRC-20 tokens, the issuer submits issuance transactions where the input amount of a given token type is less than the output amount. Only token issuance transactions can violate the token conservation rule. The public key of the issuer (the signing key of the first issuance transaction) becomes the LRC-20 token identifier. This secp256k1 public key represents the token type,

UV. The issuer can publish the issuing key online, making it publicly known what the LRC-20 token represents. The issuer can use the first token issuance transaction to announce and establish whether there is a supply cap.

If LRC-20 tokens are sent back to the issuing key itself (without the pixel key construction), then those tokens are provably destroyed. No extra proof data is needed because it is impossible (without a hash cycle) for the issuing key to equal a pixel key of that same type.

3.3 Freezing and Unfreezing Outputs

As an optional feature, at the time of issuance, the issuer can opt into the ability to freeze and unfreeze LRC-20.

To freeze a UTXO with LRC-20 pixels attached, the issuer can send a transaction containing an `OP_RETURN` output for the intended UTXO. Nodes can monitor for these freeze transactions and validate that they are signed by the token issuer. Once an LRC-20 UTXO is marked as frozen, every transaction that spends this UTXO within the same block or in a block thereafter is deemed invalid. That is, block boundaries matter. All freeze transactions are treated as if the `OP_RETURN` was in the coinbase transaction, regardless of the transaction ordering in the block. This means that any transaction spending a frozen UTXO will fail, even if the freeze transaction shows up "later" within the same block. Freezing an already spent UTXO will also lead to a failed freeze transaction.

To unfreeze previously-frozen LRC-20 tokens, the issuer signs a second `OP_RETURN` transaction to the previously-frozen output to signal to LRC-20 nodes that this UTXO is no longer frozen. Block boundaries work the same way for freezing and unfreezing transactions. Further, if a user spends their Bitcoin UTXO after it has been frozen in the LRC-20 issuer, then the unfreezing operation of the same UTXO will fail because the UTXO has been spent.

3.4 The Peer-to-Peer Network

The LRC-20 Protocol is supported by a network of LRC-20 nodes that use Bitcoin nodes to submit transactions and monitor the Bitcoin blockchain, and broadcast LRC-20 transaction data via the LRC-20 peer-to-peer network. LRC-20 transaction data is specified by the Bitcoin transaction `TXID`s, and contains the pixel key and pixel proof for every input and every output to prove the LRC-20 data commitment. Therefore, LRC-20 nodes will have the capability to identify LRC-20 transactions, create and confirm the validity of transaction proofs, and exchange transaction information with other nodes.

LRC-20 nodes cannot block the confirmation of LRC-20 transactions on Bitcoin, but they can identify invalid LRC-20 transactions and flag them to each other.

4 LRC-20 on Lightning

While LRC-20 is already useful on Bitcoin, it was fundamentally designed to be used on the Lightning Network for speed, scalability and low transaction fees. The security of the Lightning Network relies on a combination of on-chain and off-chain (but pre-signed) transactions. The same key-tweaking mechanism for LRC-20 transactions can be applied to the Lightning Network, allowing LRC-20 tokens to exist in Lightning Channels alongside Bitcoin. LRC-20 Lightning nodes can signal LRC-20 compatibility and recognize each other in the connection handshake, and then start Lightning channels with each other that contain LRC-20 tokens.

The funding output on which the Lightning channel is based uses a 2-of-2 multisig script. The first key in that script commits to the pixel data, which allows the channel to contain LRC-20 tokens. Similarly, commitment transaction outputs, which have more complex scripts, still commit to the pixel data in the first pubkey in their scripts. It is possible to sign new commitment transactions with updated LRC-20 token balances without changing the bitcoin balance within a channel. When signing commitment transactions that contain HTLCs, nodes also need to add LRC-20 data into the HTLC outputs and the HTLC transactions.

The current Lightning Protocol requires all HTLCs to be funded with at least the dust bitcoin amount, which means it is not possible to have LRC-20-only HTLCs. But it is possible to modify HTLC construction and resolution to enable a better LRC-20 user experience. For example, if Alice wants to send Bob LRC-20 tokens in an HTLC without giving him any bitcoin, Alice can add 2 HTLCs with the same timeout and preimage, where the first HTLC is funded by Alice and pays Bob the LRC-20 token along with the dust amount of bitcoin, and the second HTLC is funded by Bob and pays Alice the same dust amount of bitcoin. Once the HTLC is resolved on-chain or off-chain, both parties' bitcoin balance will remain the same.

All Lightning Network functionalities, including force-closures, justice transactions, and routing, are inherited by LRC-20 tokens on the Lightning Network. Routing LRC-20 payments is no different from routing bitcoin payments on the Lightning Network. LRC-20 Lightning payments require all nodes on the route to have sufficient liquidity in the same type of LRC-20 token. LRC-20 nodes can communicate and charge routing fees in LRC-20 tokens instead of bitcoin, if they choose to.

5 Conclusion

The LRC-20 Protocol defines a simple yet general key-tweaking standard that is applicable to any Bitcoin output script, so that LRC-20 token information can be embedded in the Bitcoin blockchain. LRC-20 tokens can be seamlessly ported on the Lightning Network, delivering faster and cheaper LRC-20 transactions. Trades between LRC-20 tokens and Bitcoin are possible on the Lightning

Network.

The LRC-20 protocol enables a range of new use cases, including on-chain non-interactive swaps between LRC-20 tokens and Bitcoin pairs. For example, users can share partial Bitcoin transactions signed with `SIGHASH_SINGLE` that have more bitcoin in outputs than inputs (and lower LRC-20 tokens in the outputs than inputs), which is, on its own, an invalid Bitcoin transaction. A user willing to trade can complete the transaction by adding their own input and output pair, which supplies enough bitcoin to make the transaction valid and claims the LRC-20 tokens.

References

[1] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. 2008.

[2] Joseph Poon and Thaddeus Dryja. *The bitcoin lightning network: Scalable off-chain instant payments*. 2016.

[3] Domo. *BRC-20*. 2023. URL: `https://domo-2.gitbook.io/brc-20-experiment`.

[4] Lightning Lab. *Taproot Assets*. 2024. URL: `https://docs.lightning.engineering/the-lightning-network/taproot-assets`.

[5] Maxim Orlovsky et al. *RGB Blackpaper*. 2023. URL: `https://blackpaper.rgb.tech/`.

[6] Pieter Wuille, Nick Jonas, and Anthony Towns. *BIP 341 – Taproot: SegWit version 1 spending rules*. 2020. URL: `https://github.com/bitcoin/bips/blob/master/bip-0341.mediawiki`.