BTCR From First Principles

By Kate Sills <katelynsills@gmail.com>

The did:btcr method enables DIDs on the Bitcoin blockchain. Prior work by many individuals during RWOT workshops has resulted in the current spec, as well as a number of other documents listed below under "Relevant Work".

This paper is an attempt to reconstruct BTCR from first principles, and consider its design in terms of particular attacks. The methodology is to start with a simplistic or naive idea about how BTCR should be created (as a relative newcomer to did:btcr, the author of this paper finds naive ideas easy to generate!) and then compare it to the current spec and prior work, noting which features or safeguards are missing in the naive version ("NaiveBTCR").

Defining NaiveBTCR

In NaiveBTCR, the identifier is simply a Bitcoin address (public key hash), which might look like this:

`did:naive-btcr:12cANGh1fyEm3x8agjcjvFtXQJeTaxg4WC`

Bitcoin uses public-private key pairs on the Secp256k1 elliptic curve. A new key pair can be created entirely permissionlessly at no cost, and with no requirement to register the address on-chain.

Resolving DIDs in NaiveBTCR

In NaiveBTCR, DID Documents are implicit, and can be generated by the resolver based on knowledge of the DID alone. A valid digital signature for the Bitcoin address is sufficient for both authentication and signing verifiable credentials.

However, we have not yet explained how updating or deleting the implicit DID Document might work, and how a resolver might check for updates and deletions. For that, we need to discuss key rotation.

Key Rotation in NaiveBTCR

Using the recently popular Taxonomy of Access Control by Ittay Eyal, we can define the following potential states for a private key:

State	User Has Access?	Adversary has Access?
Safe	Yes	No
Loss	No	No
Leak	Yes	Yes
Theft	No	Yes

At all times, the user desires to be in the "Safe" state: the only state where they have exclusive access. However, leaks may be undetected until theft occurs, so the user has an interest in regular key rotation: revoking the current keypair and creating a new keypair.

("Pre-rotation" is a solution in the case of loss or theft. In pre-rotation, the user creates a new keypair ahead of time and only rotates after the loss/theft, thus being able to authorize the new key even if the old keypair is lost.)

Key rotation requires a credible way of informing all interested parties that 1) because the old keypair is revoked, anything signed with the old key after revocation (but not necessarily before) is repudiated, and 2) any concept of identity or reputation that was attached to the old pair should be transferred to the new pair.

In NaiveBTCR, let us assume that nothing is posted on the Bitcoin blockchain. To rotate keys, users send a message peer-to-peer to interested parties announcing the new keypair. They sign this message with the old keypair. Thus, resolving a NaiveBTCR DID requires checking whether the controller of the DID has sent such a signed message announcing the rotation.

Differences Between NaiveBTCR and did:btcr v0.1

At this point, it may be helpful to compare NaiveBTCR to the current did:btcr v0.1 spec. In did:btcr, the identifier points to a transaction on the bitcoin blockchain, not an address. Thus, DIDs are by design always registered on-chain. The initial transaction may optionally include a URL in the OP_RETURN data field which points to an explicit DID Document. Because the explicit DID Document is optional on creation, any prior Bitcoin transaction with an empty OP_RETURN can also function as a DID.

Updating a DID is done without changing the identifier. To update, the DID controller signs a new transaction with the old keypair, that "spends" the current transaction outpoint to a new address. This update transaction must have a DID Document url in the OP_RETURN data field, or the DID is considered deleted. Thus, resolving a did:btcr DID may require following a chain of transactions until the "tip" is found. It's important to note that the DID Document is required to be mutable in did:btcr, because the identifier is not known until the transaction is included in the Bitcoin blockchain.

did:btcr has a number of important differences compared to NaiveBTCR: 1. A DID can resolve to an explicit DID Document, with much more information than the "implicit" version. For example, an explicit DID Document can specify different keys for signing VCs (using a variety of digital signature algorithms) besides the Bitcoin key for updating the DID itself. 2. Because the identifier points to a particular transaction and follows the chain of spends, the signing key can rotated without having to change the identifier itself. 3. Because official updates and deletions are on-chain transactions, they are timestamped and

recorded immutably by virtue of being put into a tamperproof ledger in between other blocks. However, the explicit DID Document is not timestamped and is mutable at any moment without notice. 4. There is a particular, public place to look for official updates and deletions, allowing for common knowledge about whether an update or deletion has occurred. In other words, it is possible to prove a negative, that the DID has NOT been officially updated or deleted, by checking that an update transaction for that DID does not exist on-chain in the only "place" it could be. (Note that because Bitcoin does not have finality, this common knowledge could be changed due to a reorg. Furthermore, no common knowledge is possible for the DID document itself, since because it is mutable, it could be different for any viewer at any time.)

More exploration of these differences and their pros and cons is left for future work (and collaboration at RWOT), but we will identify a few attacks to consider.

Faking a Leak

In the "faking a leak" attack, the malicious user signs a Verifiable Credential (say, a promise to pay for an item bought online), receives some benefit because of it, then repudiates the signature, saying that they didn't sign it. The malicious user says that the private key had been leaked to an adversary, and the user had revoked the leaked keypair, which the online vendor should have known. Because the supposed adversary supposedly received the benefit due to the vendor not keeping up their records, the malicious user asks for a cancellation of the transaction, thus getting a refund, and secretly keeps the benefits for themselves. This attack depends on the user and the vendor disagreeing about whether the vendor should have known the signature was invalid because the key was revoked.

Double-Spending Reputation

A malicious user can "double-spend" reputation by telling some people that their DID has updated to X, and others that their DID has updated to Y. In other words, their DID is forked. For example, they might be able to get two very large loans by telling the first lender that X is their identity, and telling the second lender that Y is their identity. The second lender would not know about the first loan, if they don't know that X and Y are actually forks of the same identity. Thus, the malicious user can get two loans when they would normally only be able to get one based on their financial reputation.

A forked DID is different than a new fresh DID, because the forked DIDs do have a prior history, including any identity and reputation that has built up. In the case of the two loans, the DID X and the DID Y are identical before any loans occur. The only difference between X and Y is that Y lacks the first loan, making it possible for the malicious user to get a second.

Scapegoat DIDs

Similar to the double-spending attack, a malicious user can fork their identity to create a DID specifically for any "negative" VCs. For example, a restaurant that wants to dodge bad reviews can tell anyone who seems likely to complain to make public reviews about identity X, while more positive customers are told to review identity Y. Then, their website only shows reviews for identity Y and unsurprisingly, the reviews are overwhelmingly positive.

Opportunities For Collaboration at RWOT and Beyond

- Identifying more potential attacks against DID methods
- Evaluating and comparing the vulnerabilities of NaiveDID and did:btcr v0.1 in terms of the above attacks
- Identifying features that prevent the attacks, such as immutable DID documents, credible common knowledge, canonical orderings of updates that prevent forks, and credible timestamps for updates and revocations
- Identifying and designing desirable features for BTCR 2.0

Relevant Work

- DID:BTCR Draft Spec - 2019
- Querying Bitcoin blockchain for BTCR Support - 2019
- BTCR Hackathon Final Report- 2019
- BTCR DID Resolver Specification - 2018
- BTCR 0.1 Design Decisions - 2018
- did:btcr Command Line Utility
- libtxref