A Decentralized Approach to Blockcerts Credential Revocation

A White Paper from Rebooting the Web of Trust V

By João Santos (Instituto Superior Técnico) and Kim Hamilton Duffy (Learning Machine)

ABSTRACT

Blockcerts are blockchain-anchored credentials with a verification process designed to be decentralized and trustless. This proposal describes an alternate Ethereum-based method of issuing Blockcerts that addresses known limitations in the current reference implementation. This approach allows revocation by issuer, recipient, or pre-defined party (guardian); enables revocation auditing; and is a compatible extension to the Blockcerts standard and verification process.

INTRODUCTION & MOTIVATION

The Blockcerts standard specifies a record for accomplishments compliant with the Open Badges v2.0 specification -- and soon, Verifiable Credentials. A distinguishing part of the Blockcerts standard is the verification process, which checks the integrity and authenticity of the credential via its presence in a (timestamped) blockchain transaction.

The initial release of the Blockcerts standard and reference implementation described only one revocation mechanism, the issuer-hosted revocation list approach also used by Open Badges. This has known limitations, including: centralization, single point of failure, and inability for a recipient to revoke. Other approaches to revocation were considered, but none were technically or economically feasible at the time given the project goals, including Bitcoin blockchain anchoring, low overhead, and minimal cost.

For example, one approach included spending a transaction output. This had the advantage that revocations were on-chain, and that either issuer or recipient could revoke. But the approach caused transaction cost to scale with the number of recipients for a batch of certificates, which became too expensive.

Revocation is one of the most difficult and incomplete aspects of any verification process, and therefore -- as outlined in Goals and Non-Goals -- a general solution is outside the scope of this paper. In this paper we revisit the revocation aspect of Blockcerts and consider other, decentralized approaches to revocation using smart contracts.

Terminology

Actors in this scenario: Bob - Issuer Alice - Recipient Carol - Verifier

An *issuer* issues a record of *recipient*'s accomplishment (*credential*) on a blockchain and shares the resulting blockchain-anchored credential (which we also call a *Blockcert*) with the *recipient.* The *recipient* can share their Blockcert and an indepedendent *verifier* can establish the authenticity and integrity of the record.

Bob issues a Blockcert to Alice. Alice then gives the Blockcert to Carol, who is able to verify that Bob actually issued that Blockcert, that it hasn't been tampered with, and that it hasn't been revoked.

Why Revocation is Important

There are several reasons for a credential to be revoked. Let us look at reasons why Alice and Bob might want to revoke Alice's credential. 1. Let us assume that some time after issuing the credential Bob notices an inaccuracy in Alice's achievement. At this point he may want to revoke the credential he issued Alice. 2. Similar to the example above, let us assume that some time after the issuance of the credential, Alice learns new information about Bob that makes her no longer want to be associated with him. She may wish to revoke the credential she received.

Goals and Non-goals

The goal of this proposal is to outline an approach to revocation that has better properties than the current method, including:

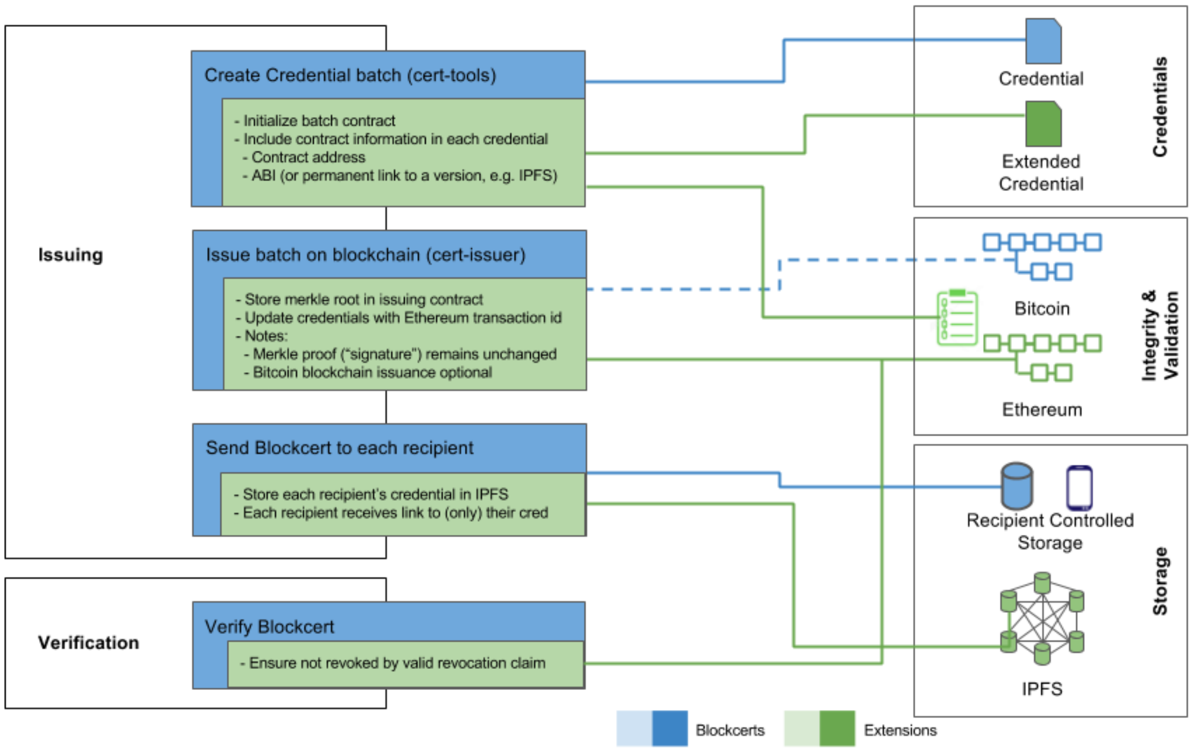
- Granting recipient ability to revoke
- Reducing the centralization point caused by issuer revocation lists
- Improving the auditability of revocations (e.g. on-chain approaches have the advantage that the issuer cannot lie/rewrite history/etc.)
- Preserving privacy as least as well as the current method used in Blockcerts (more details in Privacy)
- Scaling cost with number of revocations, not number of recipients

The approach described here is not intended to address all revocation scenarios. The intent is to allow issuer and recipient revocation in order to increase recipient control and improve auditability of revocation events. Longer term solutions could have improved privacy characteristics, described in Context and Future Directions.

ISSUING, REVOKING, AND VERIFYING

This section describes a way of issuing, revoking, and verifying Blockcerts by leveraging Ethereum smart-contracts. This extends the Blockcerts reference implementations described in

- Blockcerts issuing
- Blockcerts verification



Issuing
Create Credential batch (cert-tools)
- Initialize batch contract
- Include contract information in each credential
- Contract address
- ABI (or permanent link to a version, e.g. IPFS)
Issue batch on blockchain (cert-issuer)
- Store merkle root in issuing contract
- Update credentials with Ethereum transaction id
- Notes:
- Merkle proof ("signature") remains unchanged
- Bitcoin blockchain issuance optional
Send Blockcert to each recipient
- Store each recipient's credential in IPFS
- Each recipient receives link to (only) their cred
Verification
Verify Blockcert
- Ensure not revoked by valid revocation claim
Credential
Extended Credential
Credentials
Bitcoin
Ethereum
Integrity & Validation
Recipient Controlled Storage
IPFS
Storage
Blockcerts
Extensions

Issuing

We assume that the Issuer knows each receiver's Ethereum address to be included in the credential.

Creating a Credential Batch

First the issuer instantiates an Blockcerts issuance smart contract, which we'll refer to as the "issuance contract". This will eventially include the batch's Merkle root and list of revoked credentials. For the moment, we only need the issuance contract address and its Application Binary Interface (ABI), which will be embedded in each recipient's credential.

To embed the contract address and ABI in each credential, we extend the Blockcerts cert-tools utility, which generates Blockcerts-formatted credentials ready for blockchain issuance. The following abbreviated excerpt shows the important changes in the credentials after `cert-tools` is finished:

- New verification type, for now called `BlockcertsVerification2018`
- Addition of the `contractAddress`
- Addition of the contract `abi`

```
{
  "type": "Assertion",
  ...
  "badge": {
    ...
    "issuer": {
      "id": "https://www.blockcerts.org/samples/2.0/issuer-testnet.json",
      "type": "Profile",
      ...
      "revocationList": null
    },
  },
  "verification": {
    "type": "BlockcertsVerification2018",
    "contractAddress": "0x8efce4923b3238a747e3ee0f725da50bc245142d",
    "abi": [
       ...
    ]
  }
}
```

It's important that the issuance contract's address and ABI are included in the credential to be part of the hashed input of the batch's Merkle Tree, because otherwise the proper revocation contract could be spoofed.

In our prototypes, we included the full ABI for convenience. Eventually the contracts can be standardized; for example, other variants could support only-issuer revocation, per-issuer revocation rules, etc. Doing so would mean only the contract address and reference to the contract ABI would need to be included in the credential.

Issuing a Credential Batch

The issuer now issues the credential batch on the blockchain using cert-issuer. However, after forming the Merkle Tree of the credential hashes, the issuer updates the issuance contract as follows.

- Set `merkleRoot` to the batch's Merkle root
- Set `issuerId` to the Issuer's Ethereum address

```
{
merkleRoot = "0x0043...",
issuerId = "0x12345..."
batchRevocationStatus = false,
individualRevokedList = []
}
```

(**`batchRevocationStatus`** and **`individualRevokedList`** fields are explained in the Revoking section.)

This update records the issuance of the credential batch (via its Merkle root) on the Ethereum blockchain. *Note that the issuer does not need to issue to the Bitcoin blockchain unless Bitcoin is specifically desired. We omit that here*

After blockchain issuance, cert-issuer embeds the signature ("receipt") into each Blockcert, so that each recipient can prove their credential is part of the batch. As usual, the receipt contains the current Blockcert's expected hash and Merkle proof (the path from the credential hash to the value on the blockchain). However, instead of the

Bitcoin transaction id (as used in the current reference implementation) cert-issuer records the transaction id of the above issuance contract update.

Aditionally a link to an Ethereum blockchain explorer can be added, which would allow for the Blockcert's issuance/revocation status to be checked in real time by querying the contract, without the need to run an Ethereum node.

Revoking a Credential

`batchRevocationStatus` keeps track of the batch's revocation status and can only by changed by the Issuer. The **`individualRevokedList`** is what allows for individual credentials to be revoked. Anyone can append an item to this list, which can be seen as a claim.

Extending the example above, let's assume user *Alice*, whose Ethereum address is `0xew3428376...` makes a revocation statement about the Blockcert with `certificateId = "0x353456354..."`. It would be up to the verifying party (who is verifying Blockcert `0x353456354...`) to check whether Alice's claim is valid; that is, to check whether Alice is authorized to revoke the Blockcert in question. This can be done by checking the Blockcert's `authorizedRevokingParties` for Alice's Ethereum Wallet address.

```
{
    merkleRoot = "0x0043...",
    issuerId = "0x12345..."
    batchRevocationStatus = false,
    individualRevokedList = [
        {
            userId = "0xew3428376...", // this comes from msg.sender, so can not
be spoofed
            certificateId = "0x353456354..." // this should be a part of the
merkle tree
        }]
}
```

Verification of Credentials

Blockcerts verification performs the usual steps: - Ensure the local Blockcert hash (H_local, computed) matches the expected hash (H_target, from the receipt) - Ensure the Merkle proof from the receipt is valid - Lookup the Blockcert's batch contract address (embedded in the hashed input, therefore tamper resistant) - Ensure the Merkle root (M_receipt) matches the value in the contract (M_target) - Ensure the batch is not revoked - Ensure the Blockcert is not revoked - A Blockcert is revoked if the **`certificateId`** appears in the **`individualRevokedList`** AND **`userId`** is a member of **`authorizedRevokingParties`** for this Blockcert.

DESIGN AND IMPLEMENTATION CHOICES

Who Can Revoke

This approach describes a means of allowing either recipient or issuer to revoke a credential. However, in general an issuing contract can support a variety of revocation rules. For example, some credentials such as driver's licenses may not need recipient revocation. Perhaps the issuer may want different revocation rules per-recipient in batch or the ability for parties other than the issuer and recipient to revoke.

How Revocation Is Enforced

We originally considered a permissioned approach to revocation. For example, since we know the issuer and recipient Ethereum addresses, we could enforce in the contract that the caller is in a valid list. However, this would allow anyone inspecting the contract to see all recipient Ethereum addresses -- not just the revoked ones. this violates the goal of being *at least* as privacy preserving as the current Blockcerts revocation method (more in Privacy).

Instead, we opted to allow any contract caller to submit a revocation claim, which may or may not be ignored by a verifier. The verifier must ignore any revocation claim from a message sender that is not in the credential's `authorizedRevokingParties` list. Spam no-op revocation claims are discouraged because parties must spend money to revoke.

Credential States

We kept a simple model of a binary revoked status. In general, a contract would support multiple states, including a "suspended" state, which could be used, for example, if the credential were under review.

Omitted Bitcoin Blockchain Anchoring

If the use case does not require issuance to the Bitcoin blockchain, this approach results in a verifiable Ethereum-anchored Blockcert. For simplicity, we skipped the Bitcoin issuance step. This can be done if the issuer desires extra assurance.

Storing individual Blockcerts in IPFS

After the Blockcert is issued (i.e. Issue Credential Batch is finished), individual Blockcerts may be stored in IPFS by either the issuer or recipient. This allows the recipient to retain their credential in a distributed store, accessible and shareable with a permanent and immutable link (which is also tamper-evident by construction).

CONTEXT AND FUTURE DIRECTIONS

Privacy

The Verifiable Credentials data model's privacy considerations section provides a more complete framework of privacy concerns.

We'll focus on a few aspects: - Tracking: ability for the issuer or anyone other than the recipient to track verification of a credential - Discoverability of credential content - Discoverability of other credential recipient addresses

Because the contract does not list any recipient addresses -- it only lists revoked credential UIDs, which are intended to be unique and non-correlatable -- the only time an address is revealed is if the recipient revokes their credential; it won't be revealed if the issuer revokes it. Some third party scanning the contract could obtain credential IDs from the contract, but there is in general no way to look up credential contents from an ID, unless the issuer and recipient mutually agree.

There is the concern that correlation could be performed on an address. Because of this, recipients are encouraged to provide new addresses for each credential. Note that the recipient may want to "advertise" a certain credential or curate groups of credentials, but there are better ways to achieve this than reuse of addresses for credentials.

Data Minimization and Selective Disclosure

This paper doesn't touch on other efforts we are pursuing for high-stakes credentials, including the ability to selectively disclose contents of a credential, requiring the recipient's (or a guardian's) involvement in a verifying transaction, or zero-knowledge proofs for verification/revocation steps that reveal addresses.

Identity

This paper continues use of public keys for identification of both issuer and recipient. The Blockcerts roadmap requires replacing these with Decentralized Identifiers (DIDs), which are better suited to long-lived recipient-owned credentials. In the Blockcerts schema, this means that **`publicKey`** fields will be deprecated in favor of **`@id`** fields with DID values, per the Verifiable Credentials data model. This also could integrate with DID authentication methods for interacting with the Blockcerts issuing contract.

ADDITIONAL CREDITS

Authors: João Santos and Kim Hamilton Duffy

About Rebooting the Web of Trust

This paper was produced as part of the Rebooting the Web of Trust V design workshop. On October 3rd through October 5th, 2017, over 50 tech visionaries came together in Cambridge, Massachusetts to talk about the future of decentralized trust on the internet with the goal of writing 3-5 white papers and specs. This is one of them.

Preliminary Workshop Sponsors List: BigChainDB, Blockchain Lab, Digital Contract Design, IDEO, IPFS, Protocol Labs, Toni Lane Casserly

Workshop Producer: Christopher Allen

Workshop Facilitators: Christopher Allen, with additional paper editorial & layout by Shannon Appelcline.

What's Next?

The design workshop and this paper are just starting points for Rebooting the Web of Trust. If you have any comments, thoughts, or expansions on this paper, please post them to our GitHub issues page:

https://github.com/WebOfTrustInfo/rebooting-the-web-of-trust-fall2017/issues

The next Rebooting the Web of Trust design workshop is scheduled for March 2018 in Santa Barbara, California.

If you'd like to be involved or would like to help sponsor these events, email:

> ChristopherA@LifeWithAlacrity.com
