ChainCert: Public key infrastructure for cryptoassets

Lawrence Deacon, Nicholas Gregory and Tom Trevethan

lawrence@commerceblock.com

CommerceBlock

May 9, 2019

Abstract

For all cryptocurrency networks, and especially newly established ones, vulnerabilities exist at the distribution stage in the form of Sybil and man-in-the-middle attacks. Furthermore, unlike pure digital asset blockchains such as Bitcoin, tokenised physical assets or securities require a link between the token and the real-world asset, usually established via a legal contract; therefore the identity and jurisdiction of the token issuer must be established. Trust at the distribution layer is typically established using digital certificates and/or digital signatures from formal or informal *certificate authorities* (CAs). Real-world identities are typically established using an *extended validation* CA certificate, in which the identity of the certificate owner is validated by the CA. However, these CAs are centralized, and vulnerable to attack and/or human error, particularly due to incorrectly or maliciously issued certificates. To counter these problems we propose *ChainCert*, which comprises CA certificate extensions including blockchain identifiers, and additional public and transparent certificate monitoring and revocation tools.

Introduction

Secure web address identities are commonly established, linking real-world organizations to domain names and IP addresses, via a public key infrastructure (PKI) certificate au-

thority and the X.509/public key certificate protocol [1]. This system enables website users to interact securely with known organizations and businesses (e.g. online banking). In a world where trusted organizations have issued tokens to represent ownership of either assets or equity on public or permissioned blockchain networks, the X.509 certificate can be extended to incorporate cryptographic data that unambiguously identifies the tokens issued by the organization. This data may uniquely identify a token issuance smart contract (such as an ERC-20 contract on the Ethereum network), or it may identify the genesis block of a federated blockchain that is operated by the organization. However, a CA certificate in this approach is a single point of failure, and many have been affected by security breaches [2]. Vendors maintain certificate revocation lists; however, these can be slow to access and update. Tools for monitoring rogue certificates exist [2], but the revocation process can be inefficient. There are many intermediate certificate authorities to choose from; however, there are few root CAs, so many intermediate certificates are derived from the same root certificate. This could be improved by introducing an additional signing layer, with transparent, immutable and fast certificate validation systems.

In this paper we describe how crypto-assets can be identified, how they can be unambiguously linked to real-world assets such as precious metals, property and equity.

We investigate some existing cryptocurrencies and tokenised assets that are currently available, and evaluate their trust and identity models.

Finally we describe *ChainCert*, which adapts and enhances the PKI infrastructure to enable more secure identification of token contracts and federated blockchains.

Defining cryptoassets

Pure cryptoassets

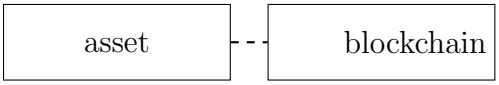

Figure 1: Identifying a pure cryptoasset. The link between asset and blockchain is self-evident since the asset is defined as the unit of exchange on a particular blockchain.

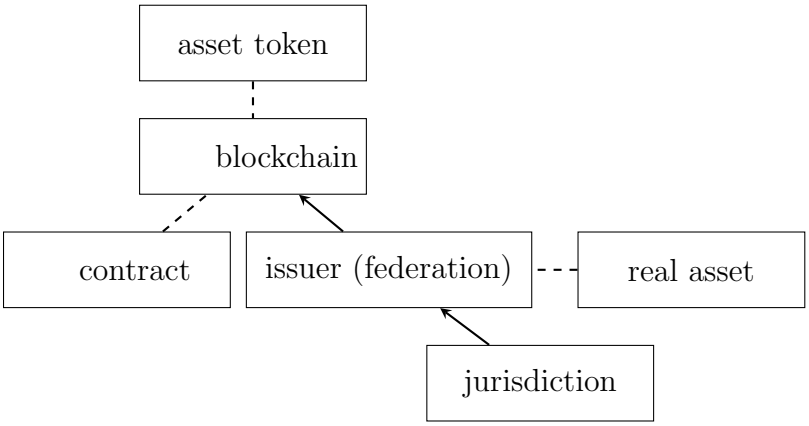

Figure 2: Identifying a tokenised real asset. The links indicated by solid arrows require proof external to the blockchain. The dashed links are either proved internally or by a combination of the other external links.

Before we can identify a cryptoasset (or token) we must first define it. Pure cryptoassets such as cryptocurrencies are defined as assets that exist only on a blockchain and do not represent ownership of rights to real-world physical assets or securities. Assuming fungibility, in the case of single asset blockchains, the definition of the asset is simple: it is defined as the unit of exchange on a particular blockchain. Therefore, identifying an asset is achieved by identifying its blockchain (i.e. the genesis block and client/node implementation) and then independently validating the state (i.e. determining the ownership of coins as unspent transactions) according to consensus rules of the client. In multiple asset blockchains, the definition of an asset also requires the asset code.

Tokenised real assets

Additionally, for tokenised assets the token is linked to the real asset via a legal contract enforceable in a particular jurisdiction. Therefore a link must be established between the blockchain, the blockchain issuer, and the legal contract that defines the real-world asset corresponding to the token. The implementation details of the how the contract is linked to the blockchain can vary. As an example, the contract can be embedded as a contract hash in the genesis block of the blockchain, and in every transaction address and signature via homomorphic pay-to-contract protocols [3]. The solid arrows in figure 2 represent links that are not intrinsic to the validation of blockchain consensus protocols and must be established externally. We discuss how this can be achieved in the following

sections.

Identifying blockchains: identity by authority or indirect social consensus?

Common to the identity of pure cryptoassets and tokenised assets alike is *blockchain identity*. Methods of establishing blockchain identity fall into two main categories: *identity by authority* and *identity by social consensus*. In the social consensus model, users consult with their peers to establish blockchain identities. In the consensus by authority model, users consult a trusted authority such as a public key infrastructure certificate authority (PKI/CA) to link blockchains to known organizations. Diligent users have a number of certificate authorities to choose from, and their choices will depend on information obtained from sources such as their social networks and trusted media sources. Therefore so-called *identity by authority* arguably falls under the category of social consensus as well; if a particular CA performs badly,and is exposed to security breaches, and the facts become widely known, then diligent users will no longer trust their certificates. In practice, every web browser contains a list of trusted root CA certificate issuing authorities curated by the authors of the web browser. Many users accept the default trusted CAs list in their web browser. This arguably implies trust in an authority. However, the same underlying social consensus rules apply: web browsers trusting fraudulent contracts would lose trust, and there are a handful of competing web browsers to choose from, but an *indirect social consensus* or *reputation* has developed around the most popular and trusted web browsers.

In contrast, direct identity by consensus has been established for the most popular blockchains. For example, the date and coinbase parameter of the Bitcoin genesis block, and the genesis block hash are widely known. Conversely, new blockchains will initially have to rely on indirect social consensus to establish their identity. The use of CA certificates is in practice a feature of established public blockchains; the Bitcoin protocol is trustless, but the software tools required to access the system are not - these are typically downloaded from a web site such as www.bitcoin.org, which is certified and encrypted using X.509/public key certificates [1]. However, in principle users do not need to trust the website from which they download the client implementation from, as the source code can be independently verified. In practice, the hashes of downloaded client software can be verified with other trusted individuals.

ChainCert requirements

Because pure digital cryptocurrencies such as bitcoin exist only as entries in a digital ledger, identity of the blockchain implies identity of the asset. For tokenised real-world assets, a link must also be established between a particular blockchain (as defined by the genesis block and client implementation), the federation running it, and the *contract* or *terms and conditions* that specify the process of redeeming the token for the real asset. The federation must also be locatable within the jurisdiction enforcing the *contract* or *terms and conditions*. For newly launched federated blockchains, social consensus may not be well established beyond a group of early adopters. Therefore the requirements of ChainCert are as follows:

- Verify the identities and locations of the federation members.
- Verify the federation web address and end-to-end encryption..
- Provably link the federation (defined by public signing keys) and contract to the genesis block.
- Verify any software implementations or servers used for interacting with the blockchain such as wallet software and servers (via cryptographic hashes).
- The above should be achieved in a manner that is as transparent, secure, intuitive and reactive as possible.

The implementation meeting the above requirements is outlined as follows:

- **The federation** will be verified and associated with their web site via digital certificate.
- **The digital certificate** is to be appended with the blockchain name, genesis block hash and other metadata in order to uniquely identify the blockchain and link it with the federation. It will can also contain hashes identifying trusted custom wallet software and wallet servers. The digital certificate is to be composed of N (probably 3) extended verification (EV) certificates issued by multiple trusted CAs. The N certificates are to be grouped into a single **ChainCert Multisignature Certificate** (*CMC*) (similar to the **Multisignature Master Key-Signed Certificate** defined in [6]) and signed by an *additional trusted entity* (ATE), and with the M of

N signature of the blockchain secured by the certificate. The ATE key will be held in a secure hardware signing module. The federation keys are also likely to be held in secure HSMs depending on the requirements of the blockchain owners. In this manner, multiple points of failure are introduced to protect the network in case one or more CAs issues a rogue certificate.

- **The *additional trusted entity*** is defined as an entity such as a blockchain infrastructure and/or software provider with an established track record, technical expertise, *etc* that can provide an additional source of trust outside of the existing PKI infrastructure.

- **A cooling off period** will be included in an extension to the CA issued certificates. The cooling off period will allow time for scrutiny of newly issued certificates. Each certificate must be published in the ChainCert immutable ledger for the specified time period before it becomes fully valid. **Proof of publication** before a specified time is easily verified by querying the ChainCert attestation tree for presence of the certificate hash T blocks in the past, where T is the required cooling-off period.

- **Custom sidechain wallets** will include ChainCert verification modules in order to validate SPV (simplified payment verification) proofs against chain certificates for real time transaction confirmation.

- **The contract** is embedded in the genesis block and every transaction (as in, for example, CommerceBlock Ocean sidechain infrastructure).

- **Certificate staychain initialisation:** With ChainCert, a number of chained, immutable state attestations (known as *staychains*) will be created in order to track valid digital certificates, and their transaction IDs added to the genesis block. This could be implemented using *e.g.* the *MainStay* [4] protocol.

- **Certificate staychain attestation:** every time a new digital certificate is issued, its hash will be committed to one of the certificate staychains. Every N_b blocks, the digital certificate hash will be recommitted to the staychain, reconfirming the validity status of the digital certificate. In this manner, an immutable and up-to-date history of the certificate is continually reaffirmed. In order to revoke a certificate, the federation can simply stop signing commitments for that certificate, or attest a NULL hash in that staychain. In this manner, digital certificates can be rapidly issued and revoked transparently and immutably. The certificates can be stored in distributed file systems so that anyone can independently verify the contents of each certificate.

- **Certificate confirmation and monitoring:** the certificate attestation is confirmed in the same manner as other MainStay attestations. Any certificate not confirmed in ChainCert for a number of blocks is considered revoked. ChainCert certificate monitoring tools will be made available as stand-alone open-source tools, and can be embedded into other software components such as custom sidechain wallets.

ChainCert architecture

In order to fulfill the above requirements, existing tools for attesting and verifying staychain commitments can be extended, covering sidechain deployment, certificate issuance, revocation and monitoring.

ChainCert multisignature certificate (*CMC*)

We define a chaincert multisignature certificate as a collection of X509 v3 extended validation certificates, at least one of which is a *ChainCert policy certificate* (*CPC*), appended with the signatures of all the certificate private keys, the CEC signature, and the federation M of N multisig block signing signature. The *CPC* contains identifying information about the blockchain in question, and is an extension of an EV certificate. The C_t certificate is owned by the additional trusted entity (ATE). The CMC is defined and constructed as follows:

$$\mathbb{C}_u = (C^1|C^2|...|C^n|C_t|C_{ts}),$$

where $\mathbb{C}_u$ denotes the unsigned CMC certificate, and | is the append operator. C_t is derived from a certificate signed by a certificate authority. C_{ts} is a certificate with the same contents as C_t, but self-signed by the ATE using a certificate derived from a private key stored in a dedicated location inside a hardware security module. This provides an extra point of security outside of the existing public key infrastructure. The completed (signed) CMC certificate is constructed by appending with the signatures of all the component certificates, such that:

$$\mathbb{C} = (\mathbb{C}_u|S^1(\mathbb{C}_u)|S^2(\mathbb{C}_u)|...|S^n(\mathbb{C}_u)|S_t(\mathbb{C}_u)|S_{ts}(\mathbb{C}_u)|S_F(\mathbb{C}_u)),$$

where S^n denotes the signature by the private key of certificate n, *etc*, and S_F is the federation block signing signature (requirement for this signature is policy-dependent).

This scheme certifies agreement by the owners of all the above certificates to the contents of all the other certificates.

ChainCert Policy Certificate *CPC*

The CPC protocol is defined a X509 standard CA certificate with additional parameters appended to is as X509 *extensions*, in order to maintain backwards compatibility with the existing PKI infrastructure. The extensions are to be defined as *non-critical* in order to maintain compatibility with existing browser software; ChainCert-enabled software such as wallets and browser extensions will be required to process the CEC parameters. We derive these parameters from a subset of the parameters found in the *Master-Key Certificate Policy* proposed in [6], with additional parameters in order to identify a particular blockchain, and extra security parameters. The CPC parameters are as follows:

1. General parameters
   (a) PROTOCOL_VERSION: ChainCert protocol version number.
   (b) POLICY_VERSION: current CPC certificate policy version number.
   (c) CA_LIST: list of trusted CAs.
   (d) MIN_CA: the minimum number of CA certificates required in the CPC.
   (e) COP_CMC: the cooling off period in days. This is the time that the CMC certificate must be published to the ChainCert staychain for before it will become trusted.
   (f) COP_CHANGE: the cooling off period in days if this certificate replaces an older certificate, and any changes have been made to any of the parameters.
   (g) SOFTWARE_HASH_<N>: The hash of a trusted wallet/node software source code or executable.
   (h) SERVER_<N>: The domain name and port of a trusted wallet server.
2. Blockchain parameters
   (a) TOKEN_FULL_NAME: *e.g.* Platinum Token.

(b) TOKEN_SHORT_NAME: *e.g.* PLT for platinum token.

(c) GENESIS_BLOCK_HASH: uniquely identifying the token's blockchain.

(d) CONTRACT_HASH: hash of the blockchain terms and conditions.

(e) CHAINCERT_SLOT_ID: the slot ID the CPC certificate will be attested to in the ChainCert staychain.

(f) FEDERATION_SCRIPTSIG: the federation block signing signature script, if this signature is required for the CMC.

(g) ASSET_ID: optional - required to identify assets of multi-asset blockchains, such as Ethereum ERC20 tokens.

A tool for appending signed genesis block metadata to a digital certificate issued by a CA and linked to a web site and the members of the federation, including which jurisdictions they are located in. The certificate will typically be signed using hardware security modules. All, or a subset of the federation could be required to be signatories, depending on requirements; these criteria should be included in the blockchain terms and conditions and in a parameter in the genesis block. The metadata are to be signed by both the certificate owner and the blockchain signatories, thereby proving that the federation and certificate owner are identical. The signed metadata are to include the following:

This "blockchain certificate", and will be attested using the *ChainCert* attestation service (figure 3). Because the certificate contains the blockchain hash, the certificate owner certifies that their specified token is traded on that blockchain.

Conversely, the blockchain certificate hash has to be validated in the *ChainCert* staychain, which is signed by the signing nodes. Therefore, fraudulent certificates not signed by the signing nodes will be invalid.

However, an attacker could launch a fake sidechain, web site and wallet software to go with their fake certificate, and impersonate a token issuer. In order to do this, they would have to attest the fake certificate to one of the staychains defined in the fake blockchain header. To guard against this, an additional trusted entity (ATE) will run an additional staychain to validate genesis block hashes and *ChainCert* staychains. *ChainCert* attestations into staychains not attested by this third party (*staychain certifier*) will be considered invalid. The staychain certifier will use a PKI/CA certificate derived from an entirely separate certificate chain.

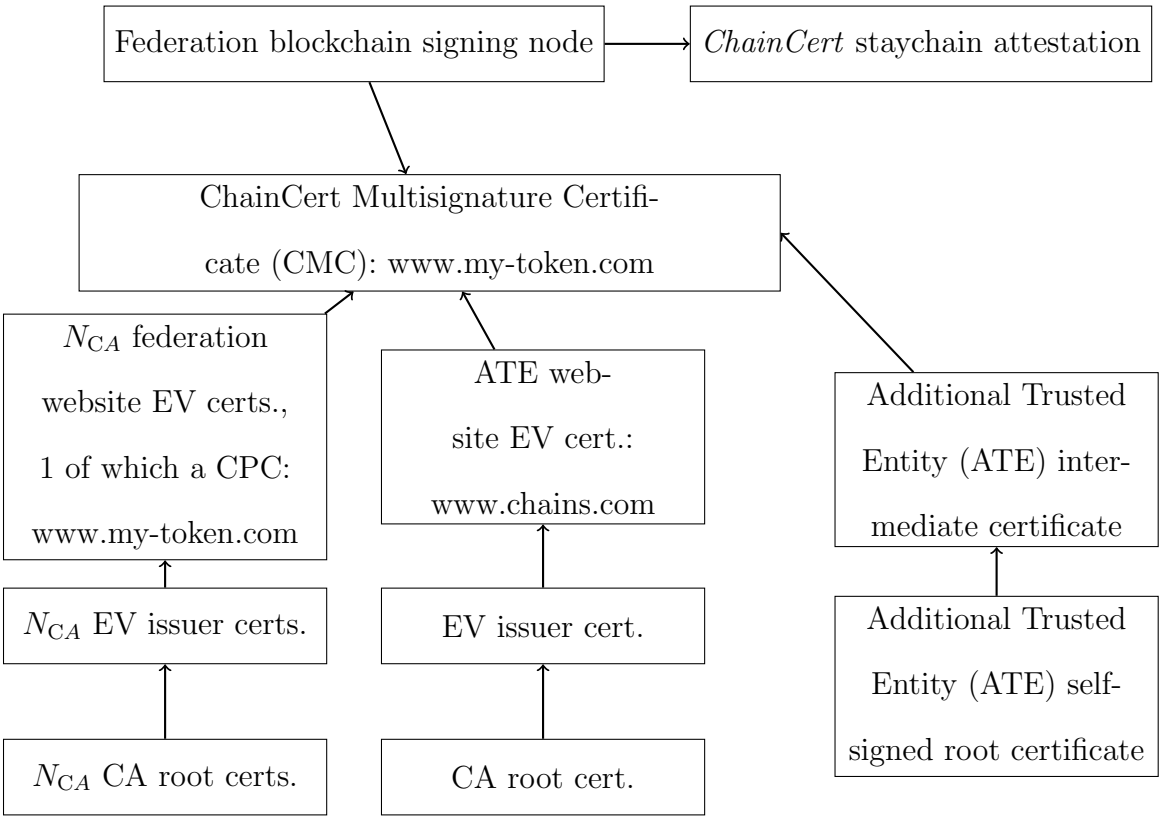

Figure 3: The certificate chain. Arrows indicate direction of signature.

Certificate attestation daemon and command line interface

The existing *Mainstay* attestation tools [5] will be extend to include *ChainCert* attestations. This will enable federations to attest the *ChainCert* certificates along with the blockchain attestations. Existing valid certificates will be reattested automatically at the same interval as block attestations. Alterations or revocations to the certificate hashes can be performed as required and will be executed via RPC. Command executions will be logged to create an audit trail.

Blockchain validator

Simple checks are to be performed to verify the validity of the CMC (and, by inclusion, the component certificates):

1. **Absence of M of N component certificates from certificate revocation lists.**
2. **Presence of the CMC certificate in the ChainCert staychain for longer then the cooling-off period, to be specified.**
3. **Identical entries in M of N of the component certificates.** For example, the domain names, company names, blockchain genesis block hashes, etc. must match, so that a successful attack including *e.g.* a spoof domain name would require at least M identical rogue certificates to be issued, each from a different CA.
4. **Each of the component certificates used comes from an accepted CA**. The list of accepted CAs is to be a subset of commonly accepted CAs.
5. **Each of the component certificates is issued by a different ROOT CA.**

The *MainStay* validation tools [5] will be extended to include *ChainCert* validation. The *ChainCert* validation tool will be included in custom wallets to provide live verification of the blockchain and the wallet server connection.

Federated sidechain deployment

One or more ChainCert attestation hashes are to be added to the blockchain configuration options (*e.g.* Ocean configuration [3]). The corresponding staychains will be used for attesting ChainCert CA certificate hashes.

Conclusions

As far as we are aware this will the first system of its kind; there is no other system available providing staychain-attested blockchain certificates for federated sidechains. This system secures real-world asset federated blockchains by leveraging CA enhanced validation to provably link the asset issuer to the asset token. The system is more decentralised than basic CA certificates because multiple entities participate in the validation; two or more distinct *certificate authorities* and an *additional trusted entity* (ATE) provide extended validation certificates, certifying the real-world identities of federation members and their web servers. The certificates are combined and each certificate owner signs the complete certificate, thereby agreeing to the contents of all the other certificates.

Additional security measures are provided through the extra parameters included in the CPC, such as:

- A cooling off period to ensure any changes are pre-committed for a specified period before certificates become active.
- The staychain attestation slot ID is specified in the blockchain header and/or CPC.
- Trusted wallet/node software hashes.
- Trusted wallet servers.

ChainCert could also be used in Ethereum ERC20 tokens and smart contracts, for example by including the contract hash in the certificate as the ASSET_ID. Furthermore, newly generated ethereum smart contracts such as ERC20 contracts could be generated using ChainCert certificate keys. The smart contract can then be validated against the certificate, and details such as contract issuer and full token name can be verified.

References

[1] https://cryptography.io/en/latest/x509/

[2] https://sslmate.com/certspotter/failures

[3] https://github.com/commerceblock/ocean-api-reference/blob/master/how-to-ocean-config.md

[4] https://www.commerceblock.com/wp-content/uploads/2018/03/commerceblock-mainstay-whitepaper.pdf

[5] https://github.com/commerceblock/mainstay

[6] P. Erdős, *Accountable and Transparent TLS Certificate Management: An Alternate Public-Key Infrastructure with Verifiable Trusted Parties*, Security and Communication Networks (Wiley/Hindawi, 2018), Wiley/Hindawi, 2018. https://doi.org/10.1155/2018/8527010