Certificate Transparency Using Blockchain

D S V Madala[1], Mahabir Prasad Jhanwar[1], and Anupam Chattopadhyay[2]

[1]Department of Computer Science. Ashoka University, India
[2]School of Computer Science and Engineering. NTU, Singapore

Abstract

The security of web communication via the SSL/TLS protocols relies on safe distributions of public keys associated with web domains in the form of X.509 certificates. Certificate authorities (CAs) are trusted third parties that issue these certificates. However, the CA ecosystem is fragile and prone to compromises. Starting with Google's Certificate Transparency project, a number of research works have recently looked at adding transparency for better CA accountability, effectively through public logs of all certificates issued by certification authorities, to augment the current X.509 certificate validation process into SSL/TLS.

In this paper, leveraging recent progress in blockchain technology, we propose a novel system, called CTB, that makes it impossible for a CA to issue a certificate for a domain without obtaining consent from the domain owner. We further make progress to equip CTB with certificate revocation mechanism. We implement CTB using IBM's Hyperledger Fabric blockchain platform. CTB's smart contract, written in Go, is provided for complete reference.

1 Introduction

The overwhelming adoption of SSL/TLS (Secure Socket Layer/Transport Layer Security Protocols) [4, 33] for most HTTP traffic has transformed the Internet into a communication platform with strong measures of confidentiality and integrity. It is one of the most widely deployed cryptographic protocols in practice, protecting majority of network-based financial or commercial transactions every day. The primary goal of SSL/TLS is to enhance a TCP connection between two communicating peers (usually a client and a server that hosts a domain) with the following properties: Authentication - the server side of the connection is always authenticated; the client side is optionally authenticated; Confidentiality - data sent over the connection after establishment is only visible to the endpoints; and Integrity: data sent over the channel after establishment cannot be modified by attackers. These properties should be true even in the face of an attacker who has complete control of the network.

SSL/TLS consists of two primary components: A *handshake protocol* that allows a client and a server to authenticate each other and to establish a *key*, and the subsequent *record layer protocol* that provides confidentiality and integrity for communication of application data. For server authentication, the SSL/TLS handshake phase requires a server to present its public key to the client browser. The communication is made secure by this public key. If the client's browser accepts an incorrect public key for the server, then the communication can be intercepted and manipulated by an attacker. This warrants a mechanism to help client-browsers accept correct public keys.

Currently, SSL/TLS heavily relies on trusted third parties to assert the authenticity of a server's claim to its public key. Certificate authorities (CAs) are trusted third parties that endorse the public keys of servers by performing checks and issuing $\mathsf{X.509}$ certificates. An $\mathsf{X.509}$ certificate is a digital document, signed using a CA's secret key, attesting to the binding of a public key to a server's identity. The client browser is pre-configured with the public keys of a number of known CAs. An $\mathsf{X.509}$ digital certificate (signed using a CA's secret key) received over a TLS handshake session is then verified by the client browser using the CA's public key, if it is available with the client browser. A typical installation of Firefox has public keys of more than 1000 CAs in its database [26].

The above CA model suffers from the following major problem. Suppose the key-pair for a server hosting a domain, say d, is $(\mathsf{pk}_d, \mathsf{sk}_d)$. Let T be a CA who is having the key pair $(\mathsf{pk}_T, \mathsf{sk}_T)$ and who, on the request of d's owner, has issued, using sk_T, a certificate $\mathsf{cert}_{T \rightarrow \mathsf{d}/\mathsf{pk}_d}$ attesting to the binding between d and pk_d. For the $\mathsf{cert}_{T \rightarrow \mathsf{d}/\mathsf{pk}_d}$ to be accepted by any browser, T must be trusted (in practice a chain of CAs must be trusted), i.e., the public key pk_T must be available with the browser. The communication to d can be intercepted by an adversary $\mathcal{A}$ (having key pair $(\mathsf{pk}_\mathcal{A}, \mathsf{sk}_\mathcal{A})$) in one of the following ways. As already mentioned, a browser typically has thousands of CAs registered in it, and the user cannot be expected to have evaluated the trustworthiness of all of them. $\mathcal{A}$ might first get an untrusted CA, say U with key pair $(\mathsf{pk}_U, \mathsf{sk}_U)$, to issue a certificate $\mathsf{cert}_{U \rightarrow \mathsf{d}/\mathsf{pk}_\mathcal{A}}$ attesting to an incorrect binding between its public key $\mathsf{pk}_\mathcal{A}$ and d. Finally if $\mathcal{A}$ manages to insert the public key pk_U of U into the client's browser, the malicious certificate $\mathsf{cert}_{U \rightarrow \mathsf{d}/\mathsf{pk}_\mathcal{A}}$ gets accepted in turn. And therefore by getting a browser to accept fake keys for standard services (such as bank web sites and webmail sites), the attacker can intercept and manipulate the user's traffic with those sites [11, 22, 27].

Alternatively, if a trusted CA (say V) gets dishonest or compromised by $\mathcal{A}$, it may issue malicious certificate $\mathsf{cert}_{V \rightarrow \mathsf{d}/\mathsf{pk}_\mathcal{A}}$ asserting fake binding. In recent years there have been high-profile cases of fraudulent certificates being issued by well trusted CAs and were used to spoof legitimate websites. For example, in 2011 an intruder managed to issue itself a valid certificate for the domain google.com and its subdomains from the prominent Dutch Certificate Authority DigiNotar [31]. In another instance, the Comodo Group suffered from an attack which resulted in the issuance of nine fraudulent certificates for domains owned by Google, Yahoo!, Skype, and others [7].

Since hundreds of certificate authorities (CAs) can issue browser-trusted certificates, it can be difficult for domain owners to detect, in real time, certificates that have been fraudulently issued attesting to the binding of fake public keys to their domain. The detection is often being reported long after the fake-certificate-assisted attacks are being carried out. For example, the malicious certificate from DigiNotar was issued in July 2011 and may have been used maliciously for weeks before its eventual detection on August 28, 2011.

In order to combat this threat, there has been some recent efforts to develop a mechanism for *efficient detection of fraudulent certificates - in almost real time.*

1.1 Related Work

The preceding discussion emphasizes the glaring vulnerability in placing trust on certifications authorities under the existing SSL/TLS system. In the current trust model of X.509-based PKI, a single compromised CA can issue a certificate for any domain. Moreover, such malicious certificates can go unnoticed over long periods of time. A variety of proposals have been made to reduce trust in the CAs.

Trusted certificate observatories such as Perspectives [34], Convergence [1], and SSL Observatory [2] confirms that a TLS certificate seen by a client is the same as the one seen by the notary. Other approaches attempt to reduce the scope of CAs' authority [13, 16, 18], thereby reducing the amount of trust and power held by CAs today.

Another promising approach is to create public-logs of all certificate operations. This approach leverages high-availability servers called public logs that maintain append-only databases of certificates issued by CAs. These databases provide efficient proofs of a certificate's presence in a log and of the log's temporal consistency. Google's Certificate Transparency (CT) project [21] was among the first to propose publicly audit-able logs as a way of providing better CA accountability. CT creates a system of public logs, which maintains a database of observed certificates issued by CAs. The log then provides a proof of a certificate's presence in the log's database, and this proof can be checked by clients during the TLS handshake. Additionally, the log is publicly-auditable so that any party can fetch proofs of presence or consistency from the hash tree of the log to monitor its operations. Special entities called auditors and monitors may perform these functions as a service for clients, publishing any evidence of CA misbehavior. The end goal of Certificate Transparency is that web clients should only accept certificates that are publicly logged and that it should be impossible for a CA to issue a certificate for a domain without it being publicly visible.

Google's Certificate Transparency works on top of the existing X.509 PKI assisted SSL/TLS. Therefore it is not a disruptive technology and only creates an additional layer of security. Unlike CT, some of the other proposals in this field seek to recommend fundamental changes to the existing system and therefore a considerable analysis must be carried out for them to be applied in practice. The Accountable Key Infrastructure (AKI) [19], like CT, adds new entities such as public log servers (called Integrity Log Servers (ILS)) and Validators. Validators are entities that monitor ILS operations, by downloading the entire ILS data structure and performing consistency checks. Additionally, AKI certificates contain several extensions over standard X.509 certificates. Among others, it recommends multiple CAs to sign a single certificate and the domain can specify in its certificate which CAs and logs are allowed to attest to the authenticity of the certificate. Attack Resilient Public-key Infrastructure (ARPKI) [8] is a system inspired by AKI, which redesigns and improves many aspects of AKI. PoliCert [32] extends the AKI by giving the domain owner a way to describe its own certificates and properties of TLS connections.

The crucial property that many of these log-based approaches seek is the detectability of a log's misbehavior, i.e., if the log presents a different set of certificates to different clients at a given point of time, or if, it does not respect the append-only property. A somewhat necessary solution to this problem would be verifying whether all (possibly worldwide) clients share a consistent view of the log. A couple of gossip protocols for Certificate Transparency that aim at detecting several types of attacks by log servers were proposed in [10].

The ideas underpinning the requirement of consistent view of logs are strongly related to blockchain technology that is widely regarded as a promising technology to operate distributed applications. A blockchain can be defined as an immutable ledger for recording transactions, maintained within a distributed network of mutually untrusting peers. Every peer maintains a copy of the ledger. Multiple copies of this ledger are kept consistent with every other copy through a process called consensus.

Namecoin blockchain [3] is one of the first forks of Bitcoin [25] and it was created to provide alternate DNS-like system that replaces DNS root servers with a blockchain for mapping domain names to DNS records. Namecoin readily provides the infrastructure to be used as a PKI. Blockstack [5] leverages the Bitcoin blockchain to provide a name registration service that also allows entities to bind public keys to their names. However, Blockstack uses its own namespace and a pricing rule based on the name length and the presence of nonalphabetic characters, and does not attempt to secure names that exist in today's DNS. IKP [23] extends the standard TLS architecture with the Ethereum blockchain [35]. Central to the IKP Ethereum network is its smart contract for detecting, publicizing, and automatically responding to CA misbehavior. To use IKP, certification authorities and domain owners must register themselves with its network. Domain owners also register domain certificate policies (DCP) (available as Ethereum contract accounts) with the IKP, providing a public policy that defines CA misbehavior (i.e., issuing an unauthorized certificate). Any violation of these policies constitutes CA misbehavior and consequently triggers automatic execution of reaction policies (RPs) (available as Ethereum contract accounts) which specify financial transactions. IKP allows participating CAs to sell, in addition to certificates, reaction policies (RPs) to domains. Domain affected by an unauthorized certificate, the detector (these are entities responsible for reporting suspicious certificates), and the CA receive payments via these transactions. The payment amounts are set such that CAs expect to lose money by issuing unauthorized certificates, and detectors expect to gain money by reporting unauthorized certificates.

1.2 Our Contribution

The *transparency* in Google's Certificate Transparency (CT) refers to the end goal that it should be impossible for a CA to issue a certificate for a domain without it being publicly visible. Subsequent proposals, in order to achieve better transparency, concentrated fundamentally on the role of domain owners in determining validation of their certificates. This line of thought is apparent through concepts such as Domain Certificate Policies (DCP) in [23] and Subject Certificate Policies (SCP) in [32]. But, like CT, these proposals follow a reactive approach. Unauthorized certificates may be accepted by browsers before its validity is finally determined - quicker and with certainty in advanced systems [23, 32] . In this paper *we propose for domain owners to have absolute control in authorizing certificates for their domains.* Domain owners are directly involved in the certificate issuance process for their domains - thus following an pre-emptive approach.

We observed that it is important for any new system design to exhibit flexibility in coexisting with the current X.509 PKI based certificate validation. Indeed, X.509 PKI is nearly impossible to replace even though it has many issues. Since its introduction in 1988, the X.509 PKI has become ubiquitous, serving important networking protocols and applications including SSL/TLS, IPSec, S/MIME, document authentication, and software updates. In part because of this ubiquity, the X.509 PKI is averse to quick change. Given this structural inertia it is important that any new system design must offer flexibility to *augment-and-not-replace* the existing system. This is further evident with the widespread deployment of CT. Since its initial March 2013 deployment, CT has publicly logged over 2 billion certificates [12]. This wouldn't have been possible without CT's ability to work on top of the existing certificate validation infrastructure. Sharing the same spirit, our proposed system offers seamless integration with the current X.509 PKI assisted certificate validation in SSL/TLS.

Since April 2018, Google's Chrome browser requires all new certificates to be published in a CT LogServer [30]. Though it is not far-fetched to imagine further increase in the deployment of CT (perhaps because of Google's influence in the WWW), but it is also true that CT is complex, inefficient on several counts such as log server's communication cost, inefficient revocation mechanism, and the fact that its security analysis is still a work in progress. With this in mind, we review Certificate Transparency in detail and propose to evaluate our system in comparison with CT on all these counts.

In summary, this paper makes the following contributions:

- We design and propose a simple blockchain-based certificate validation mechanism (§ 4.). Our proposal is called *Certificate Transparency using Blockchain* (CTB).

- We use Hyperledger Fabric blockchain to instantiate CTB. The instantiation is denoted by $\mathsf{CTB}^{\mathsf{hf}}$ (§ 4.1).
- $\mathsf{CTB}^{\mathsf{hf}}$ works on top of the current certificate validation mechanism present in X.509-assisted SSL/TLS system.
- The underlying blockchain platform provides domain owners absolute authority over their certificates. Certificate updates are not allowed unless consent from its domain is presented. The overall security of this process is bootstrapped by the underlying blockchain, in particular the CTB^{hf} smart contract (chaincode).
- We further modify $\mathsf{CTB}^{\mathsf{hf}}$ to enable it with certificate revocation mechanism (§ 4.2). This is our final proposal combining both transparency and revocation. For completeness the underlying chaincode (written in Go) is hosted at `https://www.dropbox.com/sh/vne21wpusk6yaq1/AABy8pB4jd14tIXdo1WFyO8Ra?dl=0&preview=ca-blockchain.go`.

2 Preliminaries

2.1 Certificate Transparency (CT)

Certificate transparency is a technique invented by Google that aims to prevent SSL/TLS *certificate authorities from issuing certificates for a domain without being visible to the owner of the domain*. It was among the first logbased approaches to employ a Merkle hash tree, in order to build a publicly verifiable database of certificates. The technology is being built into Google Chrome.

Certificate Transparency adds three new functional components to the current SSL certificate system: LogServers, Auditors, and Monitors.

LogServer lie at the center of the CT system. A LogServer is a simple network service that maintains a record of X.509 certificates. These certificates mostly will be submitted by certificate authorities. LogServers exhibit the following important properties: they are *append-only* (certificates can only be added to a log; certificates cannot be deleted, modified, or retroactively inserted into a log); and they are *publicly auditable* - anyone can query a LogServer for a *consistency proof* and verify that it is well behaved, or verify that a certificate has been legitimately appended to the log by querying an *audit path*. For every submitted certificate (usually by CAs), the log server responds with a promise to incorporate the certificate into the log within a time period referred to as the *maximum merge delay* (MMD). This promise takes the form of a so-called *signed certificate timestamp* (SCT). A SCT is later issued to the server which is hosting a domain for which the certificate was issued.

The SCT accompanies the certificate throughout the certificate's lifetime. In particular, a server hosting a domain must deliver the SCT with the certificate during the SSL/TLS handshake to any client browser that has initiated https to access this domain. As usual, the client validates the certificate as per the current practice. In addition, it can verify that the corresponding certificate is indeed present in the log by asking for an *audit path* to the LogServer.

Auditors verify the overall integrity of LogServers. Auditors can use consistencyproofs (provided by LogServers) to verify that a log's new entries have always been added to the log's old entries, and that nobody has ever corrupted a log by retroactively inserting, deleting, or modifying a certificate. In particular *consistency proofs* allow an Auditor to verify that its current view of a particular log is consistent with its past view.

Monitors watch for suspicious certificates in the logs of LogServers. Monitors also verify that all logged certificates are visible in the log. They do this by periodically fetching all the new entries that have been added to a log. As a result, most Monitors have complete copies of the logs they monitor.

In the following figure, we provide a specific configuration flow of the CT assisted SSL/TLS system.

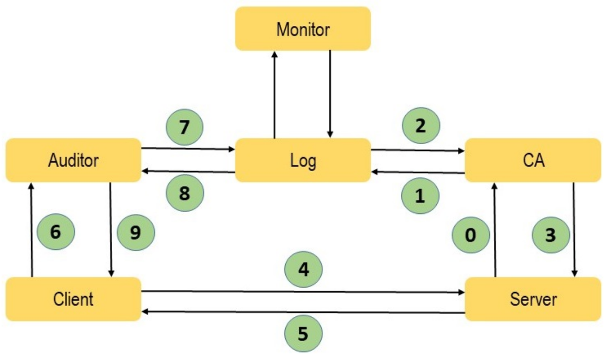

Figure 1: Certificate Transparency

0. Certificate Signing Request (CSR).
1. Request to append $\mathsf{cert}_{T\to \mathsf{d}/\mathsf{pk}_\mathsf{d}}$ to Log.
2. SCT Issuance.
3. Returning SCT + $\mathsf{cert}_{T\to \mathsf{d}/\mathsf{pk}_\mathsf{d}}$.
4. SSL/TLS hello.
5. Return $\mathsf{cert}_{T\to \mathsf{d}/\mathsf{pk}_\mathsf{d}}$ + SCT for Server Authentication.
6. Query Auditor an *audit path* for $\mathsf{cert}_{T\to \mathsf{d}/\mathsf{pk}_\mathsf{d}}$.
7. Query LogServer an *audit path* for $\mathsf{cert}_{T\to \mathsf{d}/\mathsf{pk}_\mathsf{d}}$.
8. Returning a valide *audit path*.
9. Verification of *audit path*.

Finally, as discussed above - LogServers have three important functionalities to implement, and in the following implementation details of these, i.e., how LogServers maintain certificates, produce *audit paths* and *consistency proofs*, are provided for completeness.

Appending Certificates to Log: LogServers in CT use Merkle hash trees to facilitate public auditing of certificates. A Merkle hash tree is a binary tree in which each leaf node contains hashes of individual certificates that have been appended to the log, and each non-leaf node contains the hashes of its child nodes [24]. In the following the hashing algorithm used is SHA-256. The input to the Merkle hash tree is a list of data (certificates) entries. We denote by $\mathsf{cl}[n]$ an ordered list of certificates. Indexing is 0-based: $\mathsf{cl}[n] = (\mathsf{cl}_0, \ldots, \mathsf{cl}_{n-1})$. These entries will be hashed to form the leaves of the Merkle hash tree. Each intermediate node is the hash of its two child nodes; the root of the tree, denoted by $\mathsf{MTH}(\mathsf{cl}[n])$, acts as a fingerprint of all the data contained in the tree and is the output of the algorithm in Figure 2. A LogServer appends, periodically, all of its newly acquired certificates by creating a new Merkle hash tree combining the existing and the newly acquired certificates. This process continues over and over, creating an ever-growing Merkle tree of all certificates ever submitted to the log. Figures 3–6 illustrate an instance of the above incremental process of Merkle hash tree construction.

Audit Path Generation: A Merkle audit proof lets you verify that a specific certificate has been included in a log. The audit path generation algorithm is shown in Figure 7. The Merkle audit path for a leaf node is the missing node hashes required to compute all of the nodes between the leaf and the tree root. For example, as per this algorithm, the audit path for the certificate x_3 in Stage 4 Merkle tree (Figure 6) is the tuple $(h(x_2), a, g)$. Consequently, the verification works by computing $h(x_3)$, $\ell_2 = h(h(x_2), h(x_3))$, $\ell_1 = h(a, \ell_2)$, and checking if $MTH(\mathsf{cl}[7]) \overset{?}{=} h(\ell_1, g)$.

Consistency Proof Generation: A Merkle consistency proof lets you verify that any two ver-

```
input  : $\mathsf{cl}[n] = (\mathsf{cl}_0, \ldots, \mathsf{cl}_{n-1})$
output: Returns the Merkle hash tree head $\mathsf{MTH}(\mathsf{cl}[n])$
return $\mathsf{MTH}(\mathsf{cl}[n])$

Function $\mathsf{MTH}(\mathsf{cl}[n])$:
    if $\mathsf{cl}[n] = \{\}$ then
        return $\mathsf{MTH}(\mathsf{cl}[n]) = \mathsf{SHA256}()$
    else if $|\mathsf{cl}[n]| = 1$ then
        return $\mathsf{MTH}(\mathsf{cl}[n]) = \mathsf{SHA256}(0\text{x}00 \| \mathsf{cl}[n](0))$
    end
    Compute $k = 2^{\lceil (\log_2 n) - 1 \rceil}$ [a]
    return $\mathsf{MTH}(\mathsf{cl}[n]) = \mathsf{SHA256}(0\text{x}01 \| \mathsf{MTH}(\mathsf{cl}[0:k]) \| \mathsf{MTH}(\mathsf{cl}[k:n]))$ [b]
```

[a]The expression $2^{\lceil (\log_2 n) - 1 \rceil}$ corresponds to setting k to be the largest power of two less than n

[b]$\mathsf{cl}[r:s]$ denotes the list $(\mathsf{cl}_r, \mathsf{cl}_{r+1}, \ldots, \mathsf{cl}_{s-1})$ of length $(s-r)$ and $\|$ is concatenation

Figure 2: Adding Certificates to Log

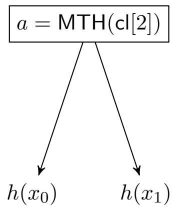

Figure 3: Stage 1

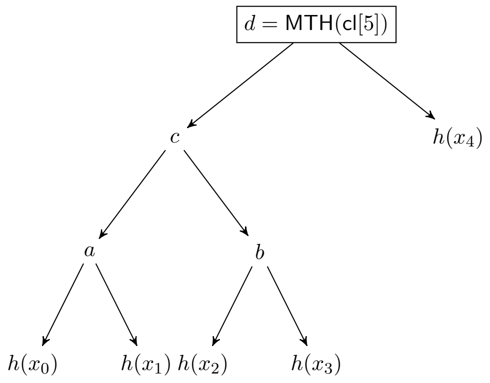

Figure 4: Stage 2

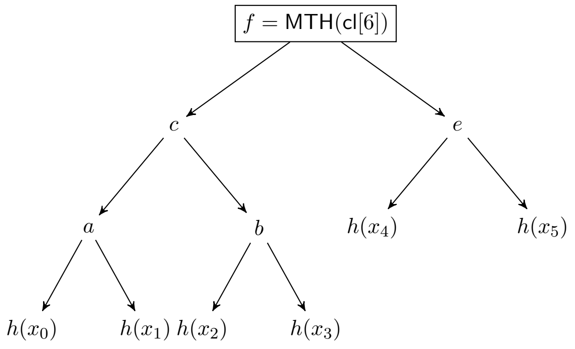

Figure 5: Stage 3

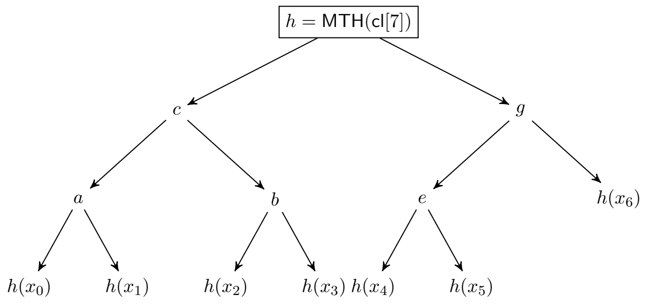

Figure 6: Stage 4

```
   input  : $\mathsf{cl}[n], m$ $(0 \le m \le n-1)$
   output: Returns a membership witness for $(m+1)$th certificate $\mathsf{cl}_m$
 1 return $\mathsf{wit}(m, \mathsf{cl}[n])$
 2
 3 Function $\mathsf{wit}(m, \mathsf{cl}[n])$:
 4     if $|\mathsf{cl}[n]| = 1$ then
 5         return $\mathsf{wit}(m, \mathsf{cl}[n]) = \{\}$
 6     Compute $k = 2^{\lceil \log_2 n-1 \rceil}$
 7     if $m < k$ then
 8         return $\mathsf{wit}(m, \mathsf{cl}[n]) = \mathsf{wit}(m, \mathsf{cl}[0:k]))\|\mathsf{MTH}(\mathsf{cl}[k:n])$
 9     else if $m \ge k$ then
10         return $\mathsf{wit}(m, \mathsf{cl}[n]) = \mathsf{wit}(m-k, \mathsf{cl}[k:n]))\|\mathsf{MTH}(\mathsf{cl}[0:k])$
11     end
```

Figure 7: Issuance of Audit Path

```
input  : $\mathsf{cl}[n], m\ (0 < m \leq n), \mathsf{MTH}(\mathsf{cl}[0:m]), \mathsf{MTH}(\mathsf{cl}[0:n])$
output: Consistency Proof
Set $\mathsf{ogl} = m$
return $\mathsf{CP}(m, \mathsf{cl}[n]) = \mathsf{CSP}(m, \mathsf{cl}[n], \mathsf{true}, \mathsf{ogl})$

Function CSP($m$, $\mathsf{cl}[n]$, $b$, $\mathsf{ogl}$):
    if $(m == n) \wedge (m == \mathsf{ogl}) \wedge (b == \mathsf{true})$ then
        return $\mathsf{CSP}(m, \mathsf{cl}[n], b, \mathsf{ogl}) = \{\}$
    else if $(m == n) \wedge (m \neq \mathsf{ogl}) \wedge (b == \mathsf{true})$ then
        return $\mathsf{CSP}(m, \mathsf{cl}[n], b, \mathsf{ogl}) = \mathsf{MTH}(\mathsf{cl}[0:m])$
    else if $(m == n) \wedge (b == \mathsf{false})$ then
        return $\mathsf{CSP}(m, \mathsf{cl}[n], b, \mathsf{ogl}) = \mathsf{MTH}(\mathsf{cl}[m])$
    end
    Compute $k = 2^{\lceil \log_2 n - 1 \rceil}$
    if $m \leq k$ then
        return $\mathsf{CSP}(m, \mathsf{cl}[n], b, \mathsf{ogl}) = \mathsf{CSP}(m, \mathsf{cl}[0:k], b, \mathsf{ogl}) \parallel \mathsf{MTH}(\mathsf{cl}[k:n])$
    else if $m > k$ then
        return $\mathsf{CSP}(m, \mathsf{cl}[n], b, \mathsf{ogl}) = \mathsf{CSP}(m-k, \mathsf{cl}[k:n], \mathsf{false}, \mathsf{ogl}) \parallel \mathsf{MTH}(\mathsf{cl}[0:k])$
    end
```

Figure 8: The Consistency Proof Algorithm: $\mathsf{CP}(\mathsf{cl}[n], m, \mathsf{MTH}(\mathsf{cl}[0:m]), \mathsf{MTH}(\mathsf{cl}[0:n]))$

sions of a log are consistent. Algorithm in Figure 8 documents how LogServers in CT generates consistency proofs. A consistency proof, say between the log in Figure 4 and the log in Figure 6, is the minimum set of intermediate nodes in the Merkle hash tree (MHT) of Figure 6 that lets you verify that 1. MHT of Figure 4 is a subset of Figure 6 MHT, and 2. Figure 6 MHT is the concatenation of Figure 4 MHT and all the new certificates. In particular, the consistency proof here is the following nodes from Figure 6 MHT: $(c, h(x_4), h(x_5), h(x_6))$. Verification works by first checking if $\mathsf{MTH}(\mathsf{cl}[5]) \overset{?}{=} h(c, h(x_4))$, thereby verifying that the old tree exists and is unchanged. Finally, compute $\theta_2 = h(h(x_4), h(x_5))$ and $\theta_1 = h(\theta_2, h(x_6))$ and check if $\mathsf{MTH}(\mathsf{cl}[7]) \overset{?}{=} h(c, \theta_1)$, thereby verifying that the current log has been consistently built.

3 Blockchain and Hyperledger Fabric

In the current section, we provide a short overview of Hyperledger Fabric blockchain network, which is necessary to follow the ideas that we develop further.

3.1 Hyperledger Fabric (HF)

Hyperledger Fabric (HF) [6,15] is an open-source blockchain platform. HF is one of the projects within the Hyperledger umbrella project [17]. In the following we describe entities that constitute Fabric network; ledger; chaincodes and endorsement policies that together defin any distributed application running on Fabric network; and the transaction flow in Fabric network following execute-order-validate cycle.

3.1.1 Network

A Hyperledger Fabric (HF) blockchain consists of a set of nodes that form a network. As HF is permissioned, all nodes that participate in the network have an identity, as provided by a modular membership service provider (MSP). Nodes in a Fabric network take up one of three roles: clients, peers, and ordering service nodes (OSN) (or, simply, orderers).

3.1.2 Ledger

The distributed ledger in HF is a combination of the world state database and the transaction log history. Each network node has a copy of the ledger. The world state component describes the state of the ledger at a given point in time. The state is a database and is modeled as a versioned key/value store. The transaction log component records all transactions which have resulted in the current value of the world state; it's the update history for the world state.

3.1.3 Distributed Application

A distributed application for HF must consists of two parts:

- Chaincode: A smart contract, called chaincode [6], is a program code that implements the application logic. It is a central part of a distributed application in HF and is deployed on the HF network, where it is executed and validated by a specific set of peers, who maintain the ledger.
- Endorsement Policy: A typical endorsement policy lets the chaincode specify the endorsers for a transaction in the form of a set of peers that are necessary for endorsement.

3.1.4 Transaction Flow

We now explain the transaction flow in HF and illustrates the steps of the execution, ordering and validation phases.

- Execution Phase: In this phase, a client sends transactions to a specific set of peers specified by the endorsement policy. Such a message is a signed request to invoke a chaincode function. It must include the chaincode id, time stamp and the transaction's payload. Each transaction is then executed by specific peers and its output is recorded.
  Endorsing peers simulate/execute transactions against the current state. Peers transmit to the client the result of this execution (read and write sets associated to the their current state) alongside the endorsing peer's signature. No updates are made to the ledger at this point.
  Clients collect and assemble endorsements into a transaction. The client verifies the endorsing peer's signatures, determine if the responses have the matching read/write set and checks if the endorsement policies has been fulfilled. If these conditions are met, the client creates a signed envelope with the peer's read and write sets, signatures and the Channel id. The aforementioned envelope represents a transaction proposal.
- Ordering Phase: After execution, transactions enter the ordering phase where clients broadcast the transaction proposal to the ordering service. The ordering service does not read the contents of the envelope; it only gathers envelopes from all channels in the network, orders them using atomic broadcast, and creates signed chain blocks containing these envelops. These are broadcast to *all* peers, with the (optional) help of gossip.

- ValidationPhase: In this phase, each peer then validates the state changes from endorsed transactions with respect to the endorsement policy and the consistency of the execution. All peers validate the transactions in the same order and validation is deterministic. Finally, each peer appends the block to the chain, and for each valid transaction the write sets are committed to current state database. An event is emitted, to notify the client application that the transaction (invocation) has been immutably appended to the chain, as well as notification of whether the transaction was validated or invalidated.

4 Our Proposal: Certificate Transparency using Blockchain (CTB)

CTB adds a new functional component to the current SSL/TLS certificate system: the CTB `network` - a *blockchain network*. It is not a replacement for, or an alternative to, the current SSL/TLS certificate system. Indeed, it does not change the fundamental chain-of-trust model that lets clients validate a domain and establish a secure connection with a server. Instead, it augments the chain-of-trust model by providing support for an extra layer of scrutiny of the entire existing certificate system.

The flow of the CTB assisted SSL/TLS certificate system begins with a certificate issuance request by a domain owner D (having the key pair $(\mathsf{pk}_D, \mathsf{sk}_D)$) on domain d to a certification authority T. Please note that the above request is being addressed as per the existing SSL/TLS certificate system and results into a X.509 certificate $\mathsf{cert}_{T\to\mathsf{d}/\mathsf{pk}_D}$.

At this phase, T, who is also required to be a member of CTB `network`, submits $\mathsf{cert}_{T\to\mathsf{d}/\mathsf{pk}_D}$ to this network for it be added to the distributed ledger. The submitted transaction will be validated by invoking the corresponding smart contract, and if successful, resulted into the addition of $\mathsf{cert}_{T\to\mathsf{d}/\mathsf{pk}_D}$ to the ledger. Finally, T also returns $\mathsf{cert}_{T\to\mathsf{d}/\mathsf{pk}_D}$ to D.

Later, during a SSL/TLS handshake, any client browser who has initiated https to the domain d, receives the certificate $\mathsf{cert}_{T\to\mathsf{d}/\mathsf{pk}_D}$ (also additional certificates forming a traditional certificate chain) from the server hosting d. As usual, the client validates the certificate and its signature chain as per the existing system. In addition, client can now verify that the $\mathsf{cert}_{T\to\mathsf{d}/\mathsf{pk}_D}$ is not malicious and indeed has been validated by the CTB `network` by querying the blockchain ledger of this network.

In the following we provide complete details of our CTB `network` by instantiating it on Hyperledger Fabric blockchain platform, and denote it by $\mathsf{CTB}^{\mathsf{hf}}$ `network`.

4.1 CTB Instantiated on Hyperledger Fabric: $\mathsf{CTB}^{\mathsf{hf}}$

$\mathsf{CTB}^{\mathsf{hf}}$ proposes a Hyperledger Fabric (HF) network among all certification authorities by requiring each certificate authorities to play the role of endorsing peers. In particular, let us assume there are n public certification authorities, say $T_1, \ldots, T_n$. In the $\mathsf{CTB}^{\mathsf{hf}}$ `network` we define, for every i in $1 \le i \le n$, a pair (P_i, C_i), where P_i denotes a peer and C_i denotes a client. Each such pair (P_i, C_i) represent the certification authority T_i. In addition, there exists a separate client, say B, representing internet browsers.

4.1.1 $\mathsf{CTB}^{\mathsf{hf}}$ Ledger, Chaincode and Endorsing Policy

We recall that a typical HF ledger consists of two distinct, though related, parts - a *world state* and a *blockchain*. The world state, a database, holds the current values of a set of *ledger states*. Ledger states are, by default, expressed as key-value pairs and they represent *assets*. HF provides the ability to *modify assets* (also referred as *state transitions* - i.e., states can be created, updated and deleted) through *chaincode invocations* - referred to as *transactions* and are submitted by clients. Accepted transactions are collected into blocks and then appended to the *blockchain*.

A typical state in $\mathsf{CTB^{hf}}$ ledger is a tuple of the form $(\mathsf{key}, \mathsf{val}) = (\mathsf{d}, \mathsf{cert}_{T \to d/\mathsf{pk}_D})$, where d represents a domain and $\mathsf{cert}_{T \to d/\mathsf{pk}_D}$ is a X.509 certificate attesting to the binding between d and pk_D (public key of the owner D of d) and is issued by a certification authority T.

State Transitions: The chaincode for $\mathsf{CTB^{hf}}$, denoted by $\mathsf{SContract.CTB^{hf}}$, contains the logic and the endorsement policy $\mathsf{EP.CTB^{hf}}$ that helps execute the following two state transitions:

State Query - A state query is made to retrieve the current value of a state. A client can invoke the chaincode against a state query by providing domain name $\mathsf{key} = \mathsf{d}$ as input. The chaincode $\mathsf{SContract.CTB^{hf}}$ has a function $\mathsf{FetchLedger}(\cdot)$ that will return $\mathsf{cert}_{T \to \mathsf{d}/\mathsf{pk}_D} \leftarrow \mathsf{FetchLedger}(\mathsf{d})$ as output.

State Update - A call to state update function from the $\mathsf{SContract.CTB^{hf}}$ allows a certification authority T to issue a fresh certificate $\mathsf{cert}_{T \to \mathsf{d}/\mathsf{pk}_D}$ attesting to the binding between domain d and possibly a new public key pk_D. The transaction payload, for such a state update call to $\mathsf{SContract.CTB^{hf}}$, consists of three inputs: $\Big(\mathsf{cert}_{T \to \mathsf{d}/\mathsf{pk}_D},\ \mathsf{cert}_{T \to T/\mathsf{pk}_T},\ \mathsf{sign}_{\mathsf{sk}^{\mathsf{c}}_D}(\mathsf{cert}_{T \to \mathsf{d}/\mathsf{pk}_D})\Big)$. The first input proposes a fresh certificate $\mathsf{cert}_{T \to d/\mathsf{pk}_D}$ attesting to the binding between d and pk_D. The validity testing of this certificate requires the public key of T which is provided by the second input - a self signed certificate $\mathsf{cert}_{T \to T/\mathsf{pk}_T}$ by T - committing to the binding between T and its public key pk_T. The third input is required if the domain d is already having a certificate issued to its name. Suppose the current state for d is: $(\mathsf{key} = \mathsf{d}, \mathsf{val} = \mathsf{cert}_{T' \to \mathsf{d}/\mathsf{pk}^{\mathsf{c}}_D})$, where T' is possibly a different CA from T, and $\mathsf{cert}_{T' \to \mathsf{d}/\mathsf{pk}^{\mathsf{c}}_D}$ is the current certificate attesting to the binding between d and the current public key $\mathsf{pk}^{\mathsf{c}}_D$. The third input, to this effect, represents a consent from D - the required input $\mathsf{sign}_{\mathsf{sk}^{\mathsf{c}}_D}(\mathsf{cert}_{T \to \mathsf{d}/\mathsf{pk}_D})$ is a signature on the proposed certificate $\mathsf{cert}_{T \to \mathsf{d}/\mathsf{pk}_D}$, signed using the current secret key $\mathsf{sk}^{\mathsf{c}}_D$ corresponding to the current public key $\mathsf{pk}^{\mathsf{c}}_D$. Clearly, a valid third input signals D's approval in updating certificates for its domain d. In absence of a third input, a proposal will still be considered valid, if no certificate against d is currently registered with $\mathsf{CTB^{hf}}$.

The endorsement policy $\mathsf{EP.CTB^{hf}}$ parametrized by $\mathsf{SContract.CTB^{hf}}$ requires a consistent endorsement by at least $\frac{n}{2}$ certification authorities in favor of a transaction for it to be able to cause a state update. In Figure 9 concrete details are provided for our smart contract $\mathsf{SContract.CTB^{hf}}$.

4.1.2 Security Analysis of $\mathsf{CTB^{hf}}$

We now conduct an informal security analysis of $\mathsf{CTB^{hf}}$ to show how it prevents CA misbehaviour. The primary objective of $\mathsf{CTB^{hf}}$ is to provide domain owners with *absolute* control over their certificates.

We consider an adversary $\mathcal{A}$, who is able to capture the trusted element of the current SSL/TLS system, i.e., certification authorities, and whose goal is to impersonate a domain owner D's domain (website) d. Let U be the compromised CA and $\mathsf{cert}_{T \to \mathsf{d}/\mathsf{pk}_D}$ be the current certificate, issued by another CA T and is registered with $\mathsf{CTB^{hf}}$, attesting to the binding between d and D's public key pk_D (sk_D is the corresponding secret key). $\mathcal{A}$ can use U's secret key to obtain a certificate $\mathsf{cert}_{U \to \mathsf{d}/\mathsf{pk}_{\mathsf{fake}}}$ with the corresponding secret key $\mathsf{sk}_{\mathsf{fake}}$ available to U. With this, the current SSL/TLS system will allow $\mathcal{A}$ to impersonate d to clients by performing active man-in-the-middle (MITM) attacks. But with $\mathsf{CTB^{hf}}$ working on top of SSL/TLS, $\mathcal{A}$ is required to overcome one more hurdle before it can successfully impersonate d. $\mathcal{A}$ must register $\mathsf{cert}_{U \to \mathsf{d}/\mathsf{pk}_{\mathsf{fake}}}$

```
Function StateUpdate.CertIssue($\mathsf{cert}_{T\to\mathsf{d}/\mathsf{pk}_D}$; $\mathsf{cert}_{T\to T/\mathsf{pk}_T}$; $\mathsf{sign}_{\mathsf{sk}^{\mathsf{c}}_D}(\mathsf{cert}_{T\to\mathsf{d}/\mathsf{pk}_D})$):
    Compute $\mathsf{cert} \leftarrow \mathsf{FetchLedger}(\mathsf{d})$
    Compute $\mathsf{pk}_1 \leftarrow \mathsf{PkExtract}(\mathsf{cert}_{T\to T/\mathsf{pk}_T})$
    if ($\mathsf{cert} \neq \phi \wedge \mathsf{Status}(\mathsf{cert}) = \mathsf{active}$) then
        Compute $\mathsf{pk}_2 \leftarrow \mathsf{PkExtract}(\mathsf{cert})$
        if
            $\mathsf{Verify}_{\mathsf{pk}_2}(\mathsf{cert}_{T\to\mathsf{d}/\mathsf{pk}_D}, \mathsf{sign}_{\mathsf{sk}^{\mathsf{c}}_D}(\mathsf{cert}_{T\to\mathsf{d}/\mathsf{pk}_D})) =$
            $\mathsf{True} \wedge \mathsf{Verify}_{\mathsf{pk}_1}(\mathsf{cert}_{T\to\mathsf{d}/\mathsf{pk}_D}) = \mathsf{True}$
         then
            return $\mathsf{cert}_{T\to\mathsf{d}/\mathsf{pk}_D}$
        else
            return Invalid
        end
    else
        if $\mathsf{Verify}_{\mathsf{pk}_1}(\mathsf{cert}_{T\to\mathsf{d}/\mathsf{pk}_D}) = \mathsf{True}$ then
            return $\mathsf{cert}_{T\to\mathsf{d}/\mathsf{pk}_D}$
        else
            return Invalid
        end
    end

Function KvExtract(x):
    Query world state for key $\mathsf{key} = \mathsf{x}$
    if ($\mathsf{key} = \mathsf{x}, \mathsf{val} = \mathsf{cert}_{T\to\mathsf{x}/\mathsf{pk}_X}$) exists in world state then
        return $\mathsf{cert}_{T\to\mathsf{x}/\mathsf{pk}_X}$

Function PkExtract($\mathsf{cert}_{T\to\mathsf{x}/\mathsf{pk}_X}$):
    return $\mathsf{pk}_X$

Function Status($\mathsf{cert}_{T\to\mathsf{x}/\mathsf{pk}_X}$):
    return active, if $\mathsf{cert}_{T\to\mathsf{x}/\mathsf{pk}_X}$ hasn't expired
```

Figure 9: $\mathsf{SContract.CTB}^{\mathsf{hf}}$

with $\mathsf{CTB}^{\mathsf{hf}}$ by updating the current authentic certificate $\mathsf{cert}_{T\rightarrow\mathsf{d}/\mathsf{pk}_D}$. Clearly, $\mathcal{A}$ cannot do this even with the help of U. Registering $\mathsf{cert}_{U\rightarrow\mathsf{d}/\mathsf{pk}_{\mathsf{fake}}}$ with $\mathsf{CTB}^{\mathsf{hf}}$ requires U to obtain the following signature $\mathsf{sign}_{\mathsf{sk}_D}(\mathsf{cert}_{U\rightarrow\mathsf{d}/\mathsf{pk}_{\mathsf{fake}}})$. Because this must be signed by D's current secret key sk_D, $\mathcal{A}$/U without that key cannot create this signature. Another way for $\mathcal{A}$ to succeed is by mounting an attack on the underlying HF blockchain network with the help of more than half of the CAs (who are registered with $\mathsf{CTB}^{\mathsf{hf}}$) endorsing this invalid transaction. But, in addition to the fact that the requirement of an incredibly big size of the colluding CAs makes it a difficult adversarial task, there are works in the blockchain community to design more robust networks/consensus algorithms addressing colluding attacks [14].

4.2 Enabling $\mathsf{CTB}^{\mathsf{hf}}$ with Certificate Revocation

When a certificate is issued, it is expected to be in use for the entire validity period. However, there are circumstances where a certificate must be revoked prior to its expiration date. Such circumstances include change of domain name, change of association between domain and the certificate issuing CA, and (suspected)compromise of the domain owner's secret key [9]. Thus, the existence of a certificate is a necessary but not sufficient evidence for its validity, and a mechanism for determining whether a certificate was revoked is needed. Revocation techniques in the traditional PKI-based SSL/TLS model is supported via certificate revocation lists (CRL) and online certificate status checking protocol (OCSP). But, checking for revocation with these approaches remains brittle. In principle the current methods involve CAs periodically pushing revocation lists to browsers enabling revocation checking. However, this solution creates a window during which the browser's revocation lists are out of date until the next push. The core proposal in the IETF draft for Certificate Transparency does not specify any revocation mechanism. An informal proposal for handling revocation exists [20], but adopting it has the side-effect of dramatically reducing the efficiency of certificate transparency. Results enabling CT with revocation and further improvements were proposed in [28, 29].

We now propose to enable revocation mechanism in CTB by including an additional functionality in $\mathsf{SContract.CTB}^{\mathsf{hf}}$. With this modification, world states in CTB needs better representation. We now propose to store a typical state in $\mathsf{CTB}^{\mathsf{hf}}$ ledger as a tuple of the form $(\mathsf{key}, \mathsf{val}) = (\mathsf{d}, (\mathsf{cert}_{T\rightarrow d/\mathsf{pk}_D}, \mathsf{revoked/notrevoked}))$, where now val to a key itself is a tuple and the second entry of a val tells us if the first entry, i.e., the certificate $\mathsf{cert}_{T\rightarrow d/\mathsf{pk}_D}$ is currently revoked or not. Next, add an additional state update function $\mathsf{StateUpdate.Revoke}$, to the $\mathsf{CTB}^{\mathsf{hf}}$ chain code $\mathsf{SContract.CTB}^{\mathsf{hf}}$, that works to update the second entry of the val corresponding to a key. A call to $\mathsf{StateUpdate.Revoke}$ allows a certification authority T to revoke an existing certificate. The transaction payload for such a call consists of four inputs: $(\mathsf{cert}_{T\rightarrow\mathsf{d}/\mathsf{pk}_D}; \mathsf{cert}_{T\rightarrow T/\mathsf{pk}_T}; \mathsf{sign}_{\mathsf{sk}_T}(\mathsf{cert}_{T\rightarrow\mathsf{d}/\mathsf{pk}_D}); \mathsf{revoke})$. The certificate $\mathsf{cert}_{T\rightarrow\mathsf{d}/\mathsf{pk}_D}$ is called for revocation by the CA T. For revocation to be applied, $\mathsf{StateUpdate.Revoke}$ checks if the following holds true: Check if the current certificate registered with $\mathsf{CTB}^{\mathsf{hf}}$ for the domain d is $\mathsf{cert}_{T\rightarrow\mathsf{d}/\mathsf{pk}_D}$ and check if both verifications $\mathsf{Verify}_{\mathsf{pk}_T}(\mathsf{cert}_{T\rightarrow\mathsf{d}/\mathsf{pk}_D})$ and $\mathsf{Verify}_{\mathsf{pk}_T}(\mathsf{sign}_{\mathsf{sk}_T}(\mathsf{cert}_{T\rightarrow\mathsf{d}/\mathsf{pk}_D}))$ hold true under the public key pk_T. A valid call to $\mathsf{StateUpdate.Revoke}$ will update the existing $(\mathsf{key}, \mathsf{val}) = (\mathsf{d}, (\mathsf{cert}_{T\rightarrow d/\mathsf{pk}_D}, \mathsf{notrevoked}))$ for d to $(\mathsf{d}, (\mathsf{cert}_{T\rightarrow d/\mathsf{pk}_D}, \mathsf{revoked}))$.

Having revoked a registered certificate $\mathsf{cert}_{T\rightarrow d/\mathsf{pk}_D}$, $\mathsf{CTB}^{\mathsf{hf}}$ enforces an *idle* period (couple of days/a week) before a fresh certificate can again be registered with $\mathsf{CTB}^{\mathsf{hf}}$ for the domain d as described by $\mathsf{StateUpdate.CertIssue}$ functionality in $\mathsf{SContract.CTB}^{\mathsf{hf}}$. In this period no fresh certificate can be registered with $\mathsf{CTB}^{\mathsf{hf}}$ for the domain d. This enables protection against a compromised T. In the absence of such an *idle* period, a compromised T, after revoking a genuine certificate, could quickly issue and register a fresh certificate attesting to binding between d and a fake public key.

The new chaincode $\mathsf{SContract.CTB^{hf}}$ with the above changes is given in Figure 10. For current state $(\mathsf{key}, \mathsf{val}) = (\mathsf{d}, (\mathsf{cert}_{T\to d/\mathsf{pk}_D}, \mathsf{revoked/notrevoked}))$, the following calls on key d, $\mathsf{FetchLedger.val1(d)}$ and $\mathsf{FetchLedger.val1(d)}$, will return $\mathsf{cert}_{T\to d/\mathsf{pk}_D}$ and $\mathsf{revoked/notrevoked}$ respectively.

4.3 Implementation Details of $\mathsf{CTB^{hf}}$

In the following we provide details on our implementation of $\mathsf{CTB^{hf}}$. Our implementation runs on Fabric version v1.1 [1]. There are three types of organizations in total: org_i $(1 \le i \le n)$, $\mathsf{C.org}$, and $\mathsf{O.org}$. Every org_i $(1 \le i \le n)$ has following entities: an admin, a peer, a client, and a membership provider. Organizations are hosted on a single MacBook Pro (Mid 2012) as dedicated docker containers (Docker Version: Community Edition, Version: 18.06.0-ce-mac70 (26399)). Admin is responsible for registering a client to an organization. The membership provider is responsible for issuing cryptographic key-pairs to peers and clients. Each org_i $(1 \le i \le n)$ represents a certification authority - they can access the network through their respective clients. Peers presenting CAs in these organizations also play the role of endorsers as per the endorsement policy. The organizations $\mathsf{C.org}$ and $\mathsf{O.org}$ represent client browsers and the ordering service of the network respectively. The peer in $\mathsf{C.org}$ does not play the role of an endorsing peer. All peers in org_i's and $\mathsf{C.org}$ have a local copy of the ledger.

The chaincode $\mathsf{SContract.CTB^{hf}}$, as described in Figure 10, is written in Go and is available at `https://www.dropbox.com/sh/vne21wpusk6yaq1/AABy8pB4jd14tIXdo1WFyO8Ra?dl=0&preview=ca-blockchain.go`.

5 Conclusions

The idea of using public logs to monitor CA behavior is common to several proposals for current Public-Key Infrastructure. But almost all of these proposals require global coordination of logs in order to ensure that they have a consistent view of valid certificates. And the methods proposed to achieve logs-consistency are either works-in-progress or inefficient. Blockchain data structure with its distributed consensus have proved to be a valuable technology in running distributed applications efficiently and securely.

In this paper we propose CTB, a blockchian based X.509 certificate registration and validation methodology into SSL/TLS. CTB is not a replacement for, or an alternative to, the current certificate validation mechanism. It augments the chain-of-trust model by providing support for multi layer scrutiny of the entire existing certificate system. Furthermore, CTB can drop seamlessly into existing SSL/TLS protocols and is much simpler than the existing public logs based approaches. We recalled CT's design components thoroughly to compare them vis-a-vis CTB's simple and modular design. Central to CTB is its smart contract that holds logics for operations involving certificates. The security of these logics is bootstrapped by cryptography and allows domain owners to have absolute control over their certificates. We further enable CTB with an efficient certificate revocation mechanism. CTB is successfully developed using Hyperledger Fabric blockchain platform. For completeness, a link (anonymous) to the underlying chaincode written in Go is given in the paper.

[1] http://hyperledger-fabric.readthedocs.io/en/release-1.1/

```
Function StateUpdate.Revoke($\mathsf{cert}_{T\to\mathsf{d}/\mathsf{pk}_D}$; $\mathsf{cert}_{T\to T/\mathsf{pk}_T}$; $\mathsf{sign}_{\mathsf{sk}_T}(\mathsf{cert}_{T\to\mathsf{d}/\mathsf{pk}_D})$; revoke):
    Compute $\mathsf{cert} \leftarrow \mathsf{FetchLedger.val1(d)}$
    Compute $\mathsf{pk}_1 \leftarrow \mathsf{PkExtract}(\mathsf{cert}_{T\to T/\mathsf{pk}_T})$
    if
        $$\mathsf{cert} == \mathsf{cert}_{T\to\mathsf{d}/\mathsf{pk}_D}$$
        $$\wedge \mathsf{Verify}_{\mathsf{pk}_1}(\mathsf{cert}_{T\to\mathsf{d}/\mathsf{pk}_D}) = \mathsf{True}$$
        $$\wedge \mathsf{Verify}_{\mathsf{pk}_1}(\mathsf{sign}_{\mathsf{sk}_T}(\mathsf{cert}_{T\to\mathsf{d}/\mathsf{pk}_D})) = \mathsf{True}$$
     then
        return revoke
    else
        return Invalid
    end

Function StateUpdate.CertIssue($\mathsf{cert}_{T\to\mathsf{d}/\mathsf{pk}_D}$; $\mathsf{cert}_{T\to T/\mathsf{pk}_T}$; $\mathsf{sign}_{\mathsf{sk}^{\mathsf{c}}_D}(\mathsf{cert}_{T\to\mathsf{d}/\mathsf{pk}_D})$):
    Compute $\mathsf{cert} \leftarrow \mathsf{FetchLedger.val1(d)}$
    Compute $\mathsf{rstat} \leftarrow \mathsf{FetchLedger.val2(d)}$
    Compute $\mathsf{pk}_1 \leftarrow \mathsf{PkExtract}(\mathsf{cert}_{T\to T/\mathsf{pk}_T})$
    if ($\mathsf{cert} \neq \phi \wedge \mathsf{Status(cert)} = \mathsf{active} \wedge \mathsf{rstat} = \mathsf{notrevoked}$) then
        Compute $\mathsf{pk}_2 \leftarrow \mathsf{PkExtract(cert)}$
        if
            $$\mathsf{Verify}_{\mathsf{pk}_2}(\mathsf{cert}_{T\to\mathsf{d}/\mathsf{pk}_D}, \mathsf{sign}_{\mathsf{sk}^{\mathsf{c}}_D}(\mathsf{cert}_{T\to\mathsf{d}/\mathsf{pk}_D}))$$
            $$= \mathsf{True} \wedge \mathsf{Verify}_{\mathsf{pk}_1}(\mathsf{cert}_{T\to\mathsf{d}/\mathsf{pk}_D}) = \mathsf{True}$$
         then
            return $\mathsf{cert}_{T\to\mathsf{d}/\mathsf{pk}_D}$
        else
            return Invalid
        end
    else if ($\mathsf{cert} = \phi \vee \mathsf{Status(cert)} \neq \mathsf{active}$) then
        if $\mathsf{Verify}_{\mathsf{pk}_1}(\mathsf{cert}_{T\to\mathsf{d}/\mathsf{pk}_D}) = \mathsf{True}$ then
            return $\mathsf{cert}_{T\to\mathsf{d}/\mathsf{pk}_D}$
        else
            return Invalid
        end
    else if ($\mathsf{cert} \neq \phi \wedge \mathsf{Status(cert)} = \mathsf{active} \wedge \mathsf{rstat} = \mathsf{revoked}$) then
        if
            $$\mathsf{time.current} \geq \mathsf{time.revoked} + \mathsf{IdlePeriod}$$
            $$\wedge \mathsf{Verify}_{\mathsf{pk}_1}(\mathsf{cert}_{T\to\mathsf{d}/\mathsf{pk}_D}) = \mathsf{True}$$
         then
            return $\mathsf{cert}_{T\to\mathsf{d}/\mathsf{pk}_D}$
        else
            return Invalid
        end
    end
```

Figure 10: StateUpdate.Revoke and StateUpdate.CertIssue functionalities in $\mathsf{SContract.CTB}^{\mathsf{hf}}$

References

[1] Convergence `https://convergence.io/`.

[2] Electronic frontier foundation. ssl observatory `https://www.eff.org/observatory`.

[3] Namecoin `https://namecoin.org/`.

[4] Freier A., Karlton P., and Kocher P. The secure sockets layer (ssl) protocol version 3.0. 2011.

[5] Muneeb Ali, Jude C. Nelson, Ryan Shea, and Michael J. Freedman. Blockstack: A global naming and storage system secured by blockchains. In *2016 USENIX Annual Technical Conference, USENIX ATC*, pages 181–194. USENIX Association, 2016.

[6] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolic, Sharon Weed Cocco, and Jason Yellick. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference, EuroSys 2018*, pages 30:1–30:15. ACM, 2018.

[7] J. Appelbaum. Detecting certificate authority compromises and web browser collusion. Tor Blog, 2011.

[8] David A. Basin, Cas J. F. Cremers, Tiffany Hyun-Jin Kim, Adrian Perrig, Ralf Sasse, and Pawel Szalachowski. ARPKI: attack resilient public-key infrastructure. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 382–393. ACM, 2014.

[9] RFC 5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. Page 69.

[10] Laurent Chuat, Pawel Szalachowski, Adrian Perrig, Ben Laurie, and Eran Messeri. Efficient gossip protocols for verifying the consistency of certificate logs. In *2015 IEEE Conference on Communications and Network Security*, pages 415–423. IEEE, 2015.

[11] P. Eckersley. Iranian hackers obtain fraudulent https certificates: How close to a web security meltdown did we get? `https://www.eff.org/deeplinks/2011/03/iranian-hackers-obtain-fraudulent-https`. Electronic Frontier Foundation, 2011.

[12] HTTPS encryption on the web: Certificate transparency. `https://transparencyreport.google.com/https/certificates`.

[13] C. Evans, C. Palmer, and R. Sleevi. Public key pinning extension for http. *RFC*, 7469:1–28, 2015.

[14] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security*, volume 8437 of *Lecture Notes in Computer Science*, pages 436–454. Springer, 2014.

[15] Hyperledger Fabric. `https://www.hyperledger.org/projects/fabric`.

[16] Phillip M. Hallam-Baker and Rob Stradling. DNS certification authority authorization (CAA) resource record. *RFC*, 6844:1–18, 2013.

[17] Hyperledger. `https://www.hyperledger.org/`.

[18] James Kasten, Eric Wustrow, and J. Alex Halderman. Cage: Taming certificate authorities by inferring restricted scopes. In *Financial Cryptography and Data Security - 17th International Conference, FC 2013*, volume 7859 of *Lecture Notes in Computer Science*, pages 329–337. Springer, 2013.

[19] Tiffany Hyun-Jin Kim, Lin-Shung Huang, Adrian Perrig, Collin Jackson, and Virgil D. Gligor. Accountable key infrastructure (AKI): a proposal for a public-key validation infrastructure. In *22nd International World Wide Web Conference, WWW*, pages 679–690. International World Wide Web Conferences Steering Committee / ACM, 2013.

[20] B. Laurie and E. Kasper. Revocation transparency. `www.links.org/files/RevocationTransparency.pdf`. Google Researcg, 2012.

[21] Ben Laurie, Adam Langley, and Emilia Käsper. Certificate transparency. *RFC*, 6962:1–27, 2013.

[22] J. Leyden. Trustwave admits crafting ssl snooping certificate: Allowing bosses to spy on staff was wrong, says security biz. `www.theregister.co.uk/2012/02/09/tustwavedisavowsmitmdigitalcert`. The Register, 2012.

[23] Stephanos Matsumoto and Raphael M. Reischuk. IKP: turning a PKI around with decentralized automated incentives. In *2017 IEEE Symposium on Security and Privacy, SP*, pages 410–426. IEEE Computer Society, 2017.

[24] Ralph C. Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*, volume 293 of *Lecture Notes in Computer Science*, pages 369–378. Springer, 1987.

[25] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system," whitepaper. 2008.

[26] Electronic Frontier Foundation. SSL Observatory. https://www.eff.org/observatory/.

[27] P. Roberts. Phony ssl certificates issued for google, yahoo, skype, others. `https://threatpost.com/phony-ssl-certificates-issued-google-yahoo-skype-others-032311/75061/`. Threat Post, 2011.

[28] Mark D. Ryan. Enhanced certificate transparency and end-to-end encrypted mail. `https://eprint.iacr.org/2013/595.pdf`. 2013.

[29] Abhishek Singh, Binanda Sengupta, and Sushmita Ruj. Certificate transparency with enhancements and short proofs. In *Information Security and Privacy - 22nd Australasian Conference, ACISP*, volume 10343 of *Lecture Notes in Computer Science*, pages 381–389. Springer, 2017.

[30] R. Sleevi. Certificate transparency in chrome - change to enforcement date: `https://groups.google.com/a/chromium.org/forum/#!msg/ct-policy/sz_3W_xKBNY/6jq2ghJXBAAJ`.

[31] T. Sterling. Second firm warns of concern after dutch hack. http://news.yahoo.com/second-firm-warns-concern-dutch-hack-215940770.html. Yahoo! News, 2011.

[32] Pawel Szalachowski, Stephanos Matsumoto, and Adrian Perrig. Policert: Secure and flexible TLS certificate management. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 406–417. ACM, 2014.

[33] Dierks T. and Rescorla E. The transport layer security (tls) protocol. tech. rep., ietf rfc 5246. 2008.

[34] Dan Wendlandt, David G. Andersen, and Adrian Perrig. *Perspectives:* improving ssh-style host authentication with multi-path probing. In *2008 USENIX Annual Technical Conference*, pages 321–334, 2008.

[35] Gavin Wood. Ethereum: A secure decentralized generalised transaction ledger. http://gavwood.com/paper.pdf.