*future internet*

MDPI

*Article*

# A Blockchain based PKI Validation System based on Rare Events Management

**Maurizio Talamo [1], Franco Arcieri [1], Andrea Dimitri [1,*] and Christian H. Schunck [1,2]**

[1] INUIT Foundation—University of Rome Tor Vergata, 00133 Rome, Italy; maurizio.talamo@inuitroma2.it (M.T.); franco.arcieri@inuitroma2.it (F.A.); christian.schunck@iao.fraunhofer.de (C.H.S.)

[2] Fraunhofer Institute for Industrial Engineering IAO, Nobelstraße 12, 70569 Stuttgart, Germany

* Correspondence: andrea.dimitri@uniroma2.it

Received: 19 December 2019; Accepted: 11 February 2020; Published: 14 February 2020


**Abstract:** Public key infrastructures (PKIs) are the cornerstone for the security of the communication layer of online services relying on certificate-based authentication, such as e-commerce, e-government, online banking, cloud services, and many others. A PKI is an infrastructure based on a hierarchical model, but the use of PKIs in non-hierarchical contexts has exposed them to many types of attacks. Here, we discuss weaknesses exploited in past attacks and we propose a solution based on an original consensus algorithm developed for use on blockchain technology. In this implementation we retain the full functionality around X.509 certificates, i.e., for the triad (server name, server address, X.509 server certificate), and demonstrate a mechanism for obtaining fast consensus. The main properties of the solution are that a consensus may be reached even when not all members of the involved PKI participate in a transaction, and that no advanced trust agreement among PKIs is needed. The proposed solution is able to detect PKI attacks and can distinguish errors from attacks, allowing precise management of anomalies.

**Keywords:** PKI; X.509 certificate; PKI attack; distributed ledger technology; blockchain; hyperledger; smart contract; rare event

## 1. Introduction

Public key infrastructures (PKIs) are the cornerstone for the security of the communication layer of online services relying on certificate-based authentication, such as e-commerce, e-government, and online banking. Their use is increasing, and propositions can be found for e-mail, social networking, cloud services, and many others.

The main goal of PKIs is to provide a secure means of authenticating identities over the Internet. It defines the policies and procedures needed to issue, manage, validate, and distribute digital certificates, and then allows the correct use of public-key encryption [1]. PKI management of public keys is usually based on the certificate standard X.509, which provides verification of ownership of a private key by some external entity (certificate authority). The X.509 certificate is defined as a data structure that binds public key values to subjects (e.g., domain names).

The binding is asserted by trusted certificate authorities (CAs), which digitally sign each certificate. The CA, before this assertion, has to strongly validate the identity of the private certificate holder [1].

An X.509 certificate [2] usually contains not only the pair (public key, subject), but other fields related to the role of the authenticated subject (for example, a web server with a public domain name) and the use or scope of the certificate (for example digital signature, authentication, and others).

The key task of the certificate and the associated PKI infrastructure is to establish trust in the certified information. This fails if false certificates are issued erroneously or maliciously and if

revocation mechanisms are weak or slow. Centralized solutions, such as certificate revocation lists (CRLs), online certificate status protocol (OCSP), and creating artificial groups of CAs, specifically designed to address these issues, have not resolved them completely. Here, we suggest a distributed approach, based on blockchain technology where trust in a certificate is established by the fact that many distributed users express trust in it. This is supported by appropriate smart contracts and a suitable consensus algorithm. Performance is a critical issue in blockchain solutions [3–5], and our solution considers it.

In the following, we analyze, in detail, attacks on PKIs in Section 2, and existing solutions in Section 3. Then, we present our proposal. In Section 4, we analyze the problem and our proposed solution. In Section 5, we describe the experiment done and evaluation of our solution and its performances. In the last paragraph, conclusions, we explain what happens for explored attacks on PKI with our solution.

The validation phase considers two aspects: performances and functionalities.

## 2. Attacks to PKI

PKI is exposed to risks due to potential failures of certificate authorities (CAs) that may be used to issue unauthorized certificates for end-users. Many recent breaches show that if a CA is compromised, the security of the corresponding end-users will be in risk. There are many cases where a CA's errors or breaches have resulted in unauthorized certificates being issued. One of the most famous events is the security breach of Dutch CA operated by DigiNotar, which resulted in the company's infrastructure being used to issue hundreds of fake digital certificates for high-profile domains, including a certificate for "google.com." After more than 500 fake certificates were discovered [6], web browser vendors were forced to blacklist all the certificates issued by the DigiNotar.

Some other security incidents have shown that CAs do not need to be breached at all. For example, in early 2012, TrustWave admitted to issuing a CA signing certificate to one of its corporate customers, allowing the customer to eavesdrop on all communications (even HTTPS) from its internal network [7]. Although TrustWave has revoked the certificate and stated that it will no longer issue subordinate root certificates to customers, it illustrates just how easy it is for CAs to make mistakes and how severe the consequences of those mistakes might be.

In 2012, TurkTrust mistakenly issued two CA signing certificates instead of end-entity certificates (which cannot issue further certificates), allowing the holders to issue rogue certificates for Google [8].

Another important weakness of PKIs concerns the reliability and security of certificate revocation lists (CRLs or associated OCSP servers), which are used to ensure proper lifecycle management of certificates, particularly the revocation, and should be queried any time a certificate is used. Classically, the CRL for a set of certificates is maintained by the same (and sole) certification authority (CA) that issued the certificates, and this introduces a single point of failure into the system. On the Internet, a client browser reads the CRL address in the server certificate that it must verify; a fake certificate will contain a coherent fake CRL address. The second critical point about CRLs is the speed of CRL updates. In fact, CRLs do not operate in real time; they are commonly updated periodically by the issuing CA, and there may be a delay between a security breach and the subsequent CRL update, resulting in the temporary use of compromised certificates. The theft of a certificate (with its associated private key) could be unknown to the CA, and the certificate is not revoked in this case. We will show that this situation can be managed and solved using our approach.

The discussed weaknesses have also been exploited in "man in the middle" (MITM) attacks. Usually, web servers offer their services using a Transport Layer Security (TLS) secure channel without mutual authentication. There are two ways to carry out MITM attacks: to obtain access to a fake web server certificate (containing the same domain name as the original web server), issued by a trusted CA, or to directly create a fake CA certificate and put it in the list of trusted CA certificates stored on the computer of the user under attack. Most browsers allow users/programs to do this. There are two

troubling aspects linked to these attacks: Client credentials can be read and memorized, and re-used later on by the attacker.

Finally, when detecting an anomaly in a server certificate, the browser warns users that the website's digital certificate is not valid and recommends not visiting the website itself. However, the user can ignore the warning, rendering this anomaly handling system weak.

Table 1 contains a summary of attacks grouped into classes, where the class represents the attacked entity.

**Table 1.** Summary of attacks grouped into classes. CA: certificate authority; CRL: certificate revocation list.

| Class | Attack | Consequence |
|---|---|---|
| **CA side** | Fake CA | Issuing and distribution of fake server certificate |
| | Fake CA in the client store | Man in the middle attack |
| **Server side** | Certificate/private key stolen with server awareness | Server duplicate/Man in the middle attack |
| | Certificate/private key stolen without server awareness | Server duplicate/Man in the middle attack |
| **CRL/OCSP** | Delay in CRL issuing | Fake server |
| | Fake CRL address in fake server certificate | Fake server |

## 3. Existing Solutions

Existing solutions to the listed problems can be grouped in three main classes, as shown in the following.

In the first class, there is a group of solutions based on the creation of one or more servers that operate in parallel to already existing servers (for example, OCSP servers). Their task is to answer questions about the validity of a certificate. They have to be efficient, fast, secure, and simple. In this class falls the so-called log-based PKI schema, where highly available public log servers monitor and publish a log of the certificates issued by the CAs, ensuring that only the certificates listed in the published logs shall be accepted and trusted by end-customers. Google's certificate transparency (CT) [9] is the most widely deployed log-based PKI (it is available in Chrome and Firefox). Proposals have been done to extend the log also to the revoked certificates, without reaching widely adoptable solutions [10]. This class also includes OCSP solutions that try to support CRLs or other error handling solutions supporting certificate revocation. The idea here is to create a super-CA that coordinates and supports the tasks of the other CAs. However, these solutions suffer from the same fragilities of standard, traditional PKIs because the underlying architecture is essentially the same.

In the second class, there is a group of solutions that come from decentralized networks of peer-to-peer certification, known as the web of trust (WoT). The main example of a WoT is PGP (pretty good privacy) where a trust function built on a social network replaces the concept of authority and the classical CA. In our contest, this trust function can be used to enforce a certificate in its link between a subject and a public key. Like PGP, users are able to designate others as trustworthy by signing their public key certificates. By doing so, a user accumulates a certificate containing his public key and digital signatures from entities that have deemed him trustworthy. Problems exist with new users and with CRL management. We think that this is a promising solution that has not yet been fully explored.

In [11], PKI interoperability is studied in detail. The paper covers all aspects behind the existing scenarios and the conclusive thesis is that there is a **trust management problem**. Only a new authority could solve this problem, but there is not a hierarchical structure that keeps together root certification authorities. In [11] the proposed solution is to define a trust broker, a new actor with the goal to evaluate the CAs objectively, and their certificates, and to send recommendations to relying parties (RP) to help them to make informed decisions about these certificates.

The paper specifies that the relation between the trust broker and the RPs must be regularized by explicit agreements. In such agreements, the trust broker recognizes its responsibility to the RPs about the provided recommendations and requires itself to respect and to protect the privacy of the RPs. On the other hand, the trust broker must be independent from the CAs. Its relationship with CAs must also be regularized by explicit agreements, so that the trust broker can transfer the responsibility to a CA when a false recommendation is made resulting from incorrect information provided by the CA.

The tasks that the trust broker has to perform are not simple to implement and maintain centrally. In this context a new class of blockchain-based solutions emerged. The starting point is the distributed ledger paradigm, which is used to secure a transaction [12,13]; there is no single point where secure data are stored. Data are shared between many users in a distributed, peer-to-peer network. This means that an attacker has to corrupt at least a significant fraction of them, and this is not easy. Many papers have proposed a blockchain to enforce PKIs [3–5,10,14–16] where the goal is to carry out efficient certificate validation and solve problems related to CRLs and to the existence of many CAs. Reference [10] analyzed log-based PKIs in an approach to protect PKI publishing CA logs. The paper suggested that log-based PKIs require a centralized, consistent source of information to operate securely when analyzing these logs (for example, to recognize that more than one entity received the same certificate). In particular, there are no incentives to synchronize CA logs and to capture misbehavior. The Ethereum blockchain is described in [10] as a decentralized architecture to provide natural financial incentives and a transaction framework to participants. The work does not address full X.509 certificate parsing since this is prohibitively expensive on Etherum and does not measure processing speeds.

In [14], the authors designed and developed a blockchain-based PKI management framework for issuing, validating, and revoking X.509 certificates. In particular, this blockchain-based PKI framework supports the revocation of a certificate, which is a real issue in traditional PKI systems. Moreover [14], as it is impossible to delete information from the blockchain, only a parent CA can mark a certificate issued by him as revoked. Thus, any misbehavior of a CA regarding certificate revocation will be also traced and noticed by all other entities. In [15], the approach is the same: the proposal is a solution in which multiple CAs share a public, decentralized, and robust ledger where CRLs are collected. Reference [16] investigates a web-of-trust based PKI Model. The approach is interesting, but in many ways has already been outdone by the developments around distributed and self-sovereign identity management solutions, which do not have the privacy limitations of this approach. The paper does not address X.509 certificates or processing times. Reference [5] presents a bootstrapped, Bitcoin-like blockchain protocol relying on proofs of work and shows theoretically how this could be used to bind public keys to identities "while guaranteeing that the majority of them are assigned to honest parties". A PKI based on this approach does not provide most features of X.509 based PKIs. No practical implementation was realized, nor were estimates of processing speeds provided.

The common limitation of these approaches [3–5,10,14–16] is that they propose a cooperative platform that works if all CAs agree. This is a very strong assumption that requires a set of common rules provided by another party that defines under what conditions a CA is accepted into the blockchain.

While promising, none of the proposed solutions have been implemented in such a way that actual X.509 certificates were processed to solve the trust management problem discussed above. Additionally, no performance benchmarks were carried out to assess whether processing speeds were realistic.

## 4. A New Approach to X.509 Certificate Management Using Distributed Ledger Technology (DLT)

Here we use DLT to build a new implementation for blockchain based management of X.509 certificates that is able to address the trust management problem while maintaining most of the existing PKI infrastructure.

The solution requires:

- A community of independent peers where everyone checks the certificates;
- The results of the checking phase must be shared in the community, by approval;
- The obtained results must be kept unchanged and tamper-proof.

With this approach, a precise detection of any misbehavior of PKI actors can be achieved. Errors can be distinguished from attacks and the previously mentioned attacks can be mitigated.

In the following we describe our solution:

### *4.1. The Chosen Blockchain Framework*

Hyperledger fabric is a permissioned blockchain framework infrastructure, originally designed by IBM and Digital Asset, which provides a modular architecture with a delineation of roles between the nodes in the infrastructure, execution of smart contracts (called "chaincodes" in fabric), and configurable consensus and membership services. A fabric network comprises "peer nodes", which execute chaincodes, access ledger data, endorse transactions, and interface with applications; "orderer nodes", which ensure the consistency of the blockchain and deliver the endorsed transactions to the peers of the network; and "MSP services", which are generally implemented as a "certification authority" that manage X.509 certificates which are used to authenticate member identities and roles [12]. To avoid misunderstanding, in the following text, "certification authority" refers to the issuers of digital certificates used internally in the fabric blockchain to identify peers, while "Certification Authority" or "CA" with upper case initials is used to identify the issuers of digital certificates discussed in this research paper.

Nowadays, we are used to operating on already instantiated blockchains, such as Bitcoin or Ethereum, where there are a number of peers that run specific programs (for example the program named GETH for Ethereum) that make them able to locally manage all the blocks of the blockchain to propose transactions and to give consensuses. One does not need to know how many peers are active at a particular time on the blockchain, but the bigger that number is, the more secure transactions are.

Using the hyperledger fabric framework, instead, a blockchain developer can instantiate his personal blockchain over a given number of organizations. Hyperledger fabric, in fact, is a completely configurable blockchain framework in which it is possible to specify the numbers of organizations and peers participating in the blockchain, and also, to modify the basic modules of the blockchain, such as the certification authority and also the consensus algorithm. In version 1.4.2 of the hyperledger fabric, there are three different consensus algorithms: Solo, Kafka, and Raft.

It is for these reasons that we choose the hyperledger fabric framework. In particular, we decided to choose the Raft module, and we modified the way a "consenter set" is chosen. Note that in Raft, a consenter set is the set of ordering nodes actively participating in the consensus mechanism for a given channel and receiving replicated logs for the channel. This can be all of the nodes available (either in a single cluster or in multiple clusters contributing to the system channel), or a subset of those nodes.

Furthermore, one of the main characteristics of hyperledger fabric is the *permissioned* approach to the blockchain developed. Only users owning a digital certificate issued by one of the certification authorities configured in the blockchain can launch smart contracts (chaincode methods in the fabric terminology). This is not an implementation limit, and all the users (both CAs and end-users) will have a private key and a public certificate, issued or recognized by the blockchain.

In the following, we use the word server instead of domain server; for example, a web server, or in general, the server of a TLS channel, identified by the subject attribute of the X.509 server public key certificate. The aim is to secure the TLS channel establishment between the client and server.

With these elements in mind, our blockchain has three classes of peers:

(a) Clients, who, as usual, activate TLS secure connections with a server, receiving the server certificate;
(b) Servers with a domain name;
(c) CAs.

Events that cause transactions are as follows:

- Client:

- Certificate validation check to establish a TLS channel with a server.

- Server
  - New certificate issuance when the server publishes a new X.509 certificate;
  - Certificate revocation when the server certificate is revoked.

- For the CA
  - New server certificate issuance: a new server X.509 certificate has been given to a server;
  - New CA certificate issuance: a new CA (sub-ca) X.509 certificate has been issued to the CA itself;
  - Server certificate revocation: a server X.509 certificate has been revoked;
  - CA certificate revocation: a CA X.509 certificate has been revoked.

### *4.2. The Chosen Consensus Algorithm (CONS)*

Our proposal is based on a fundamental assumption: the rare event assumption. This assumption defines an attack as a rare event in a context or in a network where the majority of the transactions are correct ones [17]. Starting from this assumption, we defined a novel consensus algorithm with one requirement: considering that the aim is to certify to a user/browser that the X.509 public key certificate received (i.e., TLS protocol) from a server is valid (correctly issued, not stolen, not cracked), even a small delay (20 seconds) is not acceptable [18]. The delay spent in certificate verification is added to the loading delay of the first page of the site: when this delay only exceeds several tens of seconds, it makes the site practically unusable.

Following that requirement, the novel consensus algorithm has the following main characteristic: the peers that participate in the consensus are a randomly chosen subset of all the peers connected to the blockchain. There are two reasons behind this assumption: first, we need fast certificate validation; second, the validation has to be secure. The random selection of peers protects them from attacks, because no one knows who will be in the next subset for certificate validation. The definition of our consensus algorithm draws inspiration from the consensus algorithm recalled in Algorand [19], but Algorand's performances do not meet our previously recalled requirements [19]. Thanks to our approach, it is possible to limit certificate verification delay to a few seconds. In the following we explain how this is achieved.

Details of the CONS Algorithm

Assume that a web server is S and the server certificate is C, and the i-th client/browser ($CB_i$) is associated with the i-th peer of our blockchain. When the $CB_i$ has to decide to accept or not the certificate C sent by server S, it will accept the certificate only if the execution of the smart contract that checks the certificate validity returns OK or KO, meaning that all the other peers (the random subset chosen using our novel consensus algorithm) involved in the consensus gave OK or KO, and the tuple (ipServer, server name, certificate) is appended to the ledger with a positive/negative attribute.

In our assumption, when the $CB_i$ connects to a fake server S that pretends to be S using a cracked certificate C containing the same attributes of C with, obviously, a different private key signature, the smart contract that has to verify the validity of the certificate will advise that the couple S, C':

- Is already in the ledger with a negative attribute and then is an anomaly;
- Alternatively, another couple (S,C) is in the ledger with a positive attribute, and then (S,C') has to be rejected;
- Alternatively, there is nothing stored in the ledger, and then a consensus phase has to start.

The CONS algorithm then selects randomly, the peers that will participate in the next consensus phase. In Appendix A, we explain the algorithm used in our experiment to decide the random set

of peers (RandIpAlgo) and we show that if a peer of the blockchain is periodically elected with a random algorithm responsible for selecting the subset of the other peers that will participate in the next consensus, any probabilistic attack has a very low probability of success, and this probability decreases exponentially with the subset dimension and the periodicity of the elections. The random nature of the selection ensures that an attacker has a very low probability to corrupt all the participants of the consensus. In Appendix A we discuss the minimum dimension for a random subset of the peers.

The selected peers check the tuple (ipServer, S, C') and share the results. Therefore, an attacker who does not know which peers have participated to the consensus, must attack a large number of peers (specifically, it should attack the blockchains stored inside the peers) to be sure that his attack will not be detected.

Figure 1, help the reader to better understand the sequence behind the Consensus Algorithm.

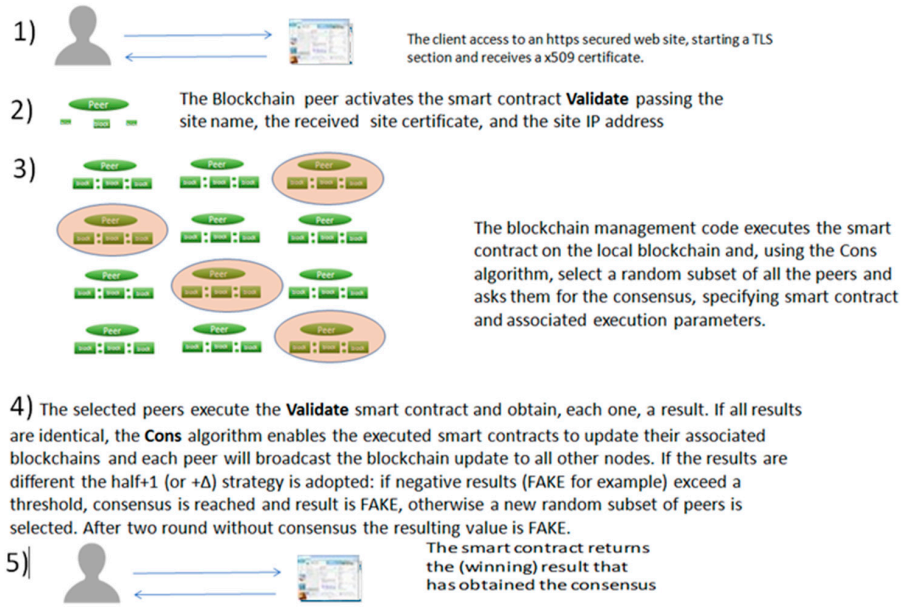


**Figure 1.** Explanation of the Cons algorithm. The size and the selection procedure of the subset of peers are crucial steps in our proposal.

### *4.3. The Smart Contract Structure and Its Relationships with the Consensus Algorithm*

The structure of the smart contracts to be implemented is given by the types of transaction executed by the peers (see Section 4.1). The most important one is the smart contract used to validate a public key certificate.

The primary requirement we have decided to adopt is that the behavior of the system must be as adherent as possible to the current validation procedures done during web sessions.

In order to do so, let us examine the process flow followed by a browser when it connects to a server secured by HTTPS protocol:

(1) Check if the certificate is structurally valid and not expired;
(2) Check if the certificate received from the server has been signed (directly or indirectly via a secure chain) by a root CA listed in some locally stored archives (trusted root CA or trusted intermediate CA);
(3) If previous checks are OK, check the revocation against locally stored CLRs or against OCSP or using CA logs, as described in a previous section of this paper.
(4) Alternatively, if one or more of the checks give KO:

   a. If allowed by the local policies of the browser and the O.S. (operating system), the user is asked, with an alert, if he wants to continue with the insecure server;

b. Otherwise, the operation is interrupted, and the connection is closed.

This is the usual behavior of browsers when we try to connect to known or unknown HTTPS servers. To be considered acceptable, our solution must have a similar behavior.

Let us examine the implementation of the smart contract used to validate a public key certificate. The name of the smart contract is Validate, and the required parameters are the Public key certificate, server domain name, and server IP number. Note that the IP address is not always unique, since, in some organizations, security policies require the use of NAT (network address translation) solutions that mask the public IP numbers of the servers, making them useless. However, in most cases, servers' IPs are unique, at least over the Internet.

A short implementation of the Validate smart contract follows (Algorithm 1). Consider that the status of a certificate inside the blockchaincould has been set by other smart contracts (i.e., when a CA revokes a certificate, the information is stored in the blockchain using a specific smart contract).

**Algorithm 1** Pseudo code of the smart contract procedure used from a client to validate an X.509 certificate received in the starting phase of a TLS connection. It is important to note that the algorithm also manages the case of a new server and a new associated certificate when the CA and the server involved are not in the blockchain. In this case, the agreement comes from a smart contract that checks that many unrelated clients with sparse locations start the TLS connection with the same server using the same certificate and the same IP address

```
1. Procedure Validate (certificate, domainName, ipAddress)
2. {   //Return values: VALID, REVOKED, EXPIRED, FAKE
3.   find value for key=(certificate,domainName,ipAddress)
4.   if(found)
5.     {
6.       If value is in (REVOKED,EXPIRED,FAKE)
7.             return value
8.      else if (the tuple =(certificate,domainName,ipAddress) is in the blockchain and has been
9.                  flagged as correct )
10.                    return valid
11.            else
12.                {
13.                    ok = check the certificate on internet (try to connect to the server) and verify
14.                        that the certificate sent by the server is identical to the checked certificate
15.                        if (ok)
16.                              store key in the blockchain with value=VALID
17.                            return VALID
18.                         else
19.                              store key in the blockchain with value=FAKE
20.                              return FAKE
21. }
22.      }
23.      else {
24.            ok = check the certificate on internet (try to connect to the server) and verify
25.                  that the certificate sent by the server is identical to the checked certificate
26.                    if (ok)
27.                              store key in the blockchain with value=VALID
28.                            return VALID
29.                       else
30.                              store key in the blockchain with value=FAKE
31.                              return FAKE
32.        }
33. }
```

The main steps of the Validate smart contract are as follows:

(0) The smart contract checks the tuple, particularly the couple (certificate, domain name), locally, like a browser, to see if the certificate is malformed, if the issuing CA is a trusted CA, and if the other static and structural checks have been done. Validate returns (and appends a simple log to the blockchain to support possible litigations) the status FAKE or EXPIRED or MALFORMED for a negative result in one of these checks. Otherwise, the following steps are done:

(1) The smart contract checks if the tuple (certificate, domain name, IP address) is already in the blockchain. There are five possible cases where the tuple is already in the blockchain:

   (a) The CA and/or the server belong to the blockchain and have appended the tuple at the certificate issuing event;
   (b) The tuple is marked as revoked or expired (by the issuer CA, by the server that owns it);
   (c) The tuple has been checked in the past (validated) with a positive result;
   (d) The tuple has been checked in the past (validated) with a negative result;
   (e) The tuple (certificate, domain name, IP) is in the blockchain with a different IP address and with a positive validation (Case c).
   (f) The tuple (certificate, domain name, IP) is in the blockchain with a different IP address and with a negative validation (Case d).

In all these instances the Validate procedure terminates—in (a, c, e) with a positive result; in the other situations with a negative result.

If the tuple is not already available in the blockchain, it needs a "further check". The "further check" comprises the connection of the peer to the server, receipt of the certificate, and making a local check of the tuple itself together with the received certificate. If all the checks are positive, the Validate procedure terminates with a VALID return value (and the tuple is appended to the blockchain with a positive evaluation).

To better understand the step "further check", let us consider a man in the middle attack. The tuple to be further checked is shared by the client peer to the random subset of the peers chosen in by the consensus algorithm. Each one of these peers executes the validation smart contract on the tuple, and if only one of these peers is not under attack (with the man in the middle), he will receive the correct certificate from the real server (solved by a real DNS server), and its validation smart contract returns the value of FAKE. Due to the fact that all the peers of the chosen subset must give the same result, here it is possible to apply different strategies, bases on the ratio among FAKE and VALID results.

When we say, "The tuple is in the blockchain", we intend it to be in the ledger of each peer in an architecture where each peer shares results with all other peers.

An important aspect to be stressed is that our smart contracts do not look exclusively inside the blockchain nodes to complete their execution but also use information coming from the network. This could sound strange considering that in a blockchain application the only trusted information is that stored in the blockchain itself. In our case this "strange" behavior is acceptable, because the network itself is not trusted, but the result given to the smart contracts and shared among nodes by the Cons algorithm become trusted by its numerosity.

## 5. Experimental Results

The checking of X.509 certificates has to be done in the middle of the TLS session and has to be fast. The issue is crucial: one of the main weaknesses of blockchain solutions is the time delay between the execution of a smart contract and the consequent updating of the blockchain. To deal with this issue, and to check that the time delay introduced by our solution is acceptable (just few seconds; less than 6 s), we implemented a prototype of a blockchain in a network with five CAs, 100 web servers, and 100,000 users. Each user connects to the web server with a maximum frequency of 10 requests/minute.

The goal of the experimental phase was to test the response times behind the approach. We had to measure the delay introduced by the blockchain to check the server certificate and to select blockchain parameters to obtain acceptable delays. The functional properties are described in the next paragraph. We implemented a prototype of the blockchain using the Go programming language (GOLANG). The communication channels were managed using the gRPC protocol (the same as hyperledger fabric). Each peer knew four to eight adjacent peers.

We ran a simulation comprising five CAs, 100 web servers, and 100,000 users, where each one connects to the web servers with a frequency of 10 pages/minute. Client requests were serial: to start a new request, the client had to terminate the previous one. Each experiment, conducted in our lab, consisted of 5 min of all entities working. In one of the experiments, for example, we registered 4,102,728 trials (client requests with full responses). This means that, on average, each client made 41 requests, and each server received 41,027 requests. Figure 2 presents the results of the experiment, where the M ($x$-axis) is the minimum number of peers that checked an X.509 certificate, and RelativeAvgRandomness ($y$-axis) is a relative index showing the sparsity of IP addresses. We can suggest some reasons behind these numbers.

- First, the distance function uses the IP addresses of peers and then has an upper bound related to the physical structure of the network. Two peers are far each other if randomly selected in the two different subnetworks. The sparsity index grows linearly with M.
- Secondly, network times increase with the growth of network traffic and the cost function. For high network times, growth is exponential.

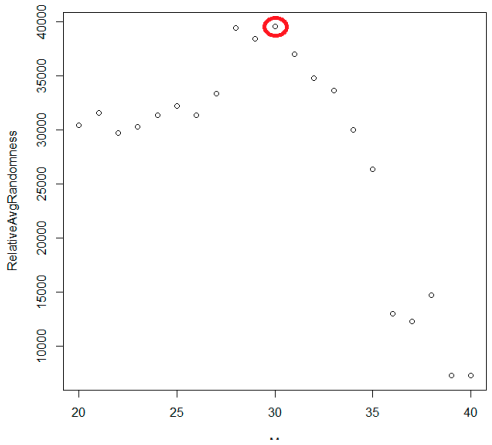


**Figure 2.** Considering the structure of an IP address, we defined an index of sparsity based on a random sample of IP addresses. This absolute index was relativized using a cost function exponential in the sample size. Repeated experiments showed that 30 is a good number for the size of a sample.

For these reasons, the ratio between the sparsity index and time costs grows initially and then decreases, reaching the maximum value for M = 30.

The experimental results tell us that the optimal configuration of the Cons algorithm occurs when the value for N, the size of the subset of the peers that participate to the consensus, is 30.

In this case, the blockchain delay is less than 6 s, a negligible delay for the first page during HTTPS web browsing. This is the first important result of our analysis—a delay of 6 s is introduced to the certificate validation procedure based on a distributed approach. Appendix A contains technical and numerical details about the experiment.

## 6. Conclusions

In this paper has been proposed a blockchain solution to avoid some weaknesses introduced by the usage of PKIs. We started by analyzing these weaknesses. Their origin comes from the usage of a

hierarchical architecture (the PKI) in a non-hierarchical context, where there are many CAs with no hierarchical constraints between them and the certified subjects. Furthermore, software architecture is open, allowing clients to connect the CAs from their computer. This is also not coherent with a hierarchical context. The analysis suggested that the solution had to be found in a distributed context, outside of the PKIs. This context is a blockchain where the peers are all actors involved in the PKI.

The first weakness we analyzed was that the PKIs are exposed to risks due to potential failures of CAs that may be used to issue unauthorized certificates for end-users. We defined a solution where an unauthorized certificate, issued by a CA, fails to work. In fact, if a previous authorized certificate for the same server is issued and declared in the blockchain by a CA or by a server, then each event regarding the new fake certificated does not obtain consensus in the blockchain. A server that wants to be preserved from this type of attack is motivated to enter the blockchain as a server peer. There is one aspect to underline.

The proposed solution allows an error to be distinguished from an attack. In the first case, the CA issuer of the unauthorized certificate will declare its existence and the issued certificate. Additionally, the involved server can declare the certificate. Then, the error emerges. If these events do not happen, then there will be an attack.

Analyzing past attacks to CAs, we note that more than the breach, the fragility of the countermove that involves clients, and then protects only updated browsers, has to be underlined. This is another weakness in PKIs that our blockchain approach resolves. A fake CA or a fake certificate can be discovered also by a non-updated browser.

Another class of fragilities relates to the efficient management of CRLs. In the proposed blockchain, a server can directly insert an event of certificate revocation for his certificate in the blockchain. All clients will immediately reject the revoked certificate (if proposed). The expired or revoked certificate is also rejected if the server is not a peer of the blockchain. If a fake server learns the private key associated with the expired certificate (for example, with a brute force attack) and tries to use it, when a client receives this fake certificate during a TLS transaction, a random sample of peers (consensus subset) will execute the Validate smart contract. For peers under attack by the fake server, the Validate procedure will return VALID, even if the certificate is a revoked certificate. However, the consensus algorithm will reject the revoked certificate because the majority of peers will connect with the correct server and will receive the updated certificate (see Figure 1).

Man in the middle attacks fail because there cannot be two certificates associated with the same subject. The set of clients that executes the smart contract is randomly selected. In this set, only a limited number of clients will be involved in the man in the middle attack. All the other clients checking the server certificate will receive the correct certificate, and then the Validate procedure will return the status FAKE. For the consensus algorithm (Figure 1, Steps 4 and 5), the tuple will be inserted into the blockchain with the value FAKE. There are two possibilities: attacked clients are a minority in the set of peers selected for tuple validation, which is the more probable scenario, and a certificate will be inserted in the blockchain as FAKE after one consensus check. If their number is close to the half of the sample, then two steps are needed to reach a consensus and to reject the certificate. Additionally, in this case, we can repeat the analysis that allows an error to be distinguished from an attack.

Extended data acquired by the client peer (IP address of the server) and sent together with the transactions allow management of the case when a pair (certificate, private key) is stolen by a server, but the server is not aware of it. In this scenario, the fake server must convince an attacked client that his IP address is the correct IP associated with the domain name of the certificate (for example, with a Domain Name Server (DNS) attack). When the attacked client uses the Validate smart contract to check the tuple (certificate, domain name, IP address), he will receive FAKE as the return value because other peers solving the IP address associated with the domain name will obtain a different value. This is true if the rare event assumption is valid: the fake server cannot attack all DNS servers on the internet but only a small subset. Thus, many clients, peers of our blockchain, will continue to receive the correct IP address associated with the domain name of the certificate.

A new type of attack could involve entering as a peer in the blockchain and generating events to block it. This is a denial of service attack. This can only happen if the attacker enters the blockchain as a CA or as a server without the right CA or the right server. The declaration of the right server exposes the fake peer, and using the permissioned property of the used blockchain, allows his exclusion from the blockchain.

An important property of the proposed blockchain is that it the participation of all CAs as peers of the blockchain itself is not required. The algorithm used for the consensus makes it adaptive and usable. An original smart contract allows the blockchain to work in open environments and defines new and shorter response times for the consensus. The proposed solution can detect man in the middle attacks and can distinguish errors from attacks, allowing precise anomaly management. Protection of clients that work with incomplete information is one of the goals of the proposed solution.

All described results have been verified with extensive experimentation that showed the effectiveness of the solution in terms of time performance, flexibility, and usability. The next steps are:

- Releasing and distributing developed codes as open source code; this step will help to check usability and consistency of developed tools in a larger community;
- Testing and validating the developed smart contract on different blockchains issuing interoperability problems;
- Performing a deeper analysis of usability of the proposed solution client side, with an experiment that includes all existing clients in terms of PC, operative systems, and browsers and their configurations;
- Submitting the developed experimental platform to penetration tests.

We also intend to extend our solution to other classes of problems. One example is the protection of other programs and environments in the client pc. One of them, for example, is a javascript library, loaded at runtime with the browser and the network. An important aspect is that the right version of the library can be learned considering the community of peers, using process mining [20].

**Author Contributions:** M.T. and A.D. conceived the presented idea and the experiment plan. F.A., A.D., and C.H.S. verified the analytical methods; and F.A. and A.D. carried out the experiments. All authors discussed the results and contributed to the final manuscript. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was funded by "Credito Sportivo" inside the project "Biomedicina," grant number CUP E86C18002330005.

**Acknowledgments:** We would like to thank Silvio Casagrande of the Inuit Foundation Security Lab for his technical support and for the availability of the hardware devices used during the experiments.

**Conflicts of Interest:** The authors declare no conflict of interest.

## Appendix A

We implemented a blockchain prototype in GOLANG language. The communication channels were managed using the GRPC protocol (the same as hyperledger fabric). Each peer knew 4 to 8 adjacent peers. We ran a simulation comprising 5 CAs, 100 web servers, and 100,000 users, where each one was connected to the web servers with a maximum frequency of 10 pages/minute.

Experimental trials were saved in a database of two relational tables:

Experiment (Experiment ID, M);

Experiment Trials (idExp, idTrial, M, Absolute Sparsity Index, Time, Time Cost Value, Relative Sparsity Index).

Let us consider single fields to explain the experiment details. The first table, Experiment, contains the list of experiments with a parameter M explained in the next lines. In the second table, ExperimentTrials, each record is a single section (certificate checked by a client and associated target peer of the blockchain). The idExp and idTrial columns uniquely identify this section.

M is the number of actual peers that check the target certificate in the current section. The optimal value of M is one of the goals of the experimentation.

**AbsoluteSparsityIndex**

Given a target peer (TP) and a target certificate (TC) to check, we had to select a subset of the 100,000 IP addresses. To do this task, we used the Raft consensus algorithm [21]. In the background, all peers of the blockchain agreed on a leader peer, known here as the RLp (Raft leader peer). The leader peer is re-negotiated and provides updates via the Raft consensus algorithm. The actual RLp receives a call from the TP. The RLp updates, periodically, in the background, a table with web servers and subsets of associated IP addresses (RandIpSet). The subset of IP addresses is of size M. The Raft consensus algorithm avoids the risk of a target peer being attacked during the procedure of deciding the random subset of IP addresses. A randomly selected peer decides, each time, this random subset.

We considered the structure of an IP address (classA.classB.classC.Y) and assumed that IPs of the same class have shorter distances between them than IPs of different classes. The experimental network had two groups of IP addresses:

192.168.x.y ($256 \times 250 = 64{,}000$ available IP addresses)
192.169.x.y ($256 \times 250 = 64{,}000$ available IP addresses)

We left out 6 IP addresses for network management.

We had 128,000 available IP addresses. A subset of them was statically assigned to our 100,000 clients. Half of them were assigned to the numbers 1–50,000 in the address set 192.168.x.y, and the other half were assigned to the numbers 1–50,000 in the address set 192.169.x.y. We defined the following distance function between two IP addresses:

$$IP1 = x1.x2.x3.x4$$

$$IP2 = y1.y2.y3.y4$$

$$D(IP1,IP2) = (IP1 \text{ xor } IP2).$$

**AbsoluteSparsityIndex** = Sum of distances of all couples in the sample/M

**RandIPAlgo**

Considering that it works in the background (by the RLp) without adjunctive costs, the RandIPAlgo that outputs the RandIpSet, is as follows:

Step 1 Generate a subset of M/2 random numbers in the set [1,50,000];
Step 2 Get the subset of associated IP in the class 192.168.x.y;
Step 3 Generate a subset of M/2 random numbers in the set [1,50,000];
Step 4 Get a subset of associated IP in the class 192.169.x.y.

Given the set of IP addresses, we calculated the absolute sparsity index (AbsoluteSparsityIndex).

**Time and TimeCostValue fields**

Given that, we defined the time cost for the target peer as the sum of the costs required to execute the following steps:

- Step 1: Direct connection to the RLp (the target peer TP knows its IP address).
- Step 2: The RLp selects the previously generated subset of IP addresses (RandIPSet) and asks them to execute the Validate procedure passing the actual parameters (certificate, domain name, IP address), executing the first part of the CONS procedure.
- Step 3: Each selected peer (blockchain node) runs the Validate procedure.
- Step 4: RLp receives M answers and sends the result to the target peer and to the whole blockchain (second part of the CONS procedure). We added functionality to the standard consensus algorithm to speed up the certificate check in our context and particularly, to avoid blockchain sharing time.

The RLp first sends the result to the TP (to update its local blockchain) and then shares the same result in the blockchain (following standard blockchain protocol). The TP receives the same result twice, but this is not a problem.

- Step 5: the TP reads the validation of the target certificate from its copy of the blockchain.

Using a packet sniffer (Wireshark) positioned in the same subnet of each target peer, we measured, on the network, the time from Step 1 to Step 5 for each trial. In particular, we developed a program to read the output file of Wireshark and to update database records opportunely.

To be more realistic and consider the fact that all of our clients are identical in their hardware and network configuration, we added a random component to each trial:

TimeLen = Time length of a trial(in seconds) = time measured by the sniffer + NormalDistr(1,1).

NormalDistr(1,1) means a number selected randomly from a normal distribution with parameters ($\mu = 0$, $\sigma = 1$). With this decision, we considered the effects of clients with heterogeneous CPUs, other PC components, and network traffic.

At this point, we adopted a cost function called CF(TimeLen). Considering that the final user cannot wait a lot of time for a response, CF has to be an increasing, always positive function:

$$CF = (TimeLen - \tau)\hat{}3 + \tau\hat{}3$$

For TimeLen less than $\tau$, CF is concave. For TimeLen greater than $\tau$, CF is convex. Then, $\tau$ represents a time beyond which waiting becomes quickly hard to accept. In our experiments, we used $\tau = 3$.

The last field in the table is the RelativeSparsityIndex:

RelativeSparsityIndex = AbsoluteSparsityIndex/CF

**Results**

We submitted the following query to select the best M value (the sample size):
Select M, avg(RelativeSparsityIndex), avg(Time)
From ExperimentTrials
Group by M.
Table A1 shows the output (main results) of the query.

**Table A1.** Summary of the most significant tests, where M represents the sample size (number of the selected peers), the second column represents the randomness index of the sample and the third column represents the average amount of time spent by the blockchain to obtain an answer.

| *M* | avg(RelativeSparsityIndex) | Avg(Time) |
|---|---|---|
| ***20*** | ***30,434.78*** | ***5.66*** |
| ***21*** | ***31,533.55*** | ***5.65*** |
| ***22*** | ***29,675.02*** | ***5.92*** |
| ***23*** | ***30,298.61*** | ***5.93*** |
| ***24*** | ***31,341.9*** | ***5.94*** |
| ***25*** | ***32,194.41*** | ***5.99*** |
| ***26*** | ***31,380.52*** | ***6.01*** |
| ***27*** | ***33,372.6*** | ***6.1*** |
| ***28*** | ***39,412.3*** | ***5.98*** |
| ***29*** | ***38,401.43*** | ***5.8*** |
| ***30*** | ***39,532.11*** | ***6.04*** |
| ***31*** | ***36,978.64*** | ***6.16*** |
| ***32*** | ***34,779.43*** | ***6.12*** |

**Table A1.** *Cont.*

| *M* | avg(RelativeSparsityIndex) | Avg(Time) |
|---|---|---|
| ***33*** | ***33,610.2*** | ***6.13*** |
| ***34*** | ***30,001.11*** | ***6.81*** |
| ***35*** | ***26,344.21*** | ***7.14*** |
| ***36*** | ***12,989.01*** | ***8.4*** |
| ***37*** | ***12,291.99*** | ***8.72*** |
| ***38*** | ***14,700.12*** | ***8.33*** |
| ***39*** | ***7288.54*** | ***9.68*** |
| ***40*** | ***7277.10*** | ***9.43*** |

Using these data, Figure 2 was obtained.

## References


1. Cooper, D. Internet X.509 public key infrastructure certificate and certificate revocation list (crl) profile. *RFC* **2008**, *5280*, 1–151.
2. Prins, J.; Cybercrime, B.U. Diginotar certificate authority breach operation black tulip. *Fox-IT Interim Rep.* **2011**. [CrossRef]
3. Hasan, H.R.; Salah, K. Proof of Delivery of Digital Assets using Blockchain and Smart Contracts. *IEEE Access* **2018**, *6*, 65439–65448. [CrossRef]
4. Pongnumkul, S.; Siripanpornchana, C.; Thajchayapong, S. Performance Analysis of Private Blockchain Platforms in Varying Workloads. In Proceedings of the 2017 26th International Conference on Computer Communication and Networks (ICCCN), Vancouver, BC, Canada, 31 July–3 August 2017. [CrossRef]
5. Garay, J.A.; Kiayas, A.; Leonardos, N. Bootstrapping the Blockchain with Applications to Consensus and Fast PKI setup. In *Public-Key Cryptography—PKC 2018*; Springer: Berlin/Heidelberg, Germany, 2018; Volume 10770.
6. Constantin, L. rustware Admits Issuing Man-in-the-Middle Digital Certificate; Mozilla Debates Punishment. February 2012. Available online: https://www.computerworld.com/article/2501291/trustwave-admits-issuing-man-in-the-middle-digital-certificate--mozilla-debates-punishment.html (accessed on 10 May 2019).
7. Langley, A. Enhancing Digital Certificate Security. January 2013. Available online: http://security.blogspot.com/2013/01/enhancing-digital-certificate-security.html (accessed on 10 May 2019).
8. Baitha, A.; Vinod, S. Session Hijacking and Prevention Technique. *Int. J. Eng. Technol.* **2018**, *7*, 193. [CrossRef]
9. Certificate Transparency. Available online: https://www.certificate-transparency.org/ (accessed on 10 May 2019).
10. Matsumoto, S.; Reischuk, R.M. Ikp: Turning a pki around with blockchains. *IACR Cryptol. ePrint Arch.* **2016**, *2016*, 1018.
11. Wazan, A.S.; Laborde, R.; Barrère, F.; Benzekri, A.; Chadwick, D.W. PKI Interoperability: Still an Issue? A Solution in the X.509 Realm. In Proceedings of the IFIP World Conference on Information Security Education, Bento Gonçalves, Brazil, 27–31 July 2009. [CrossRef]
12. Androulaki, E.; Barger, A.; Bortnikov, V.; Cachin, C.; Christidis, K.; De Caro, A.; Enyeart, D.; Ferris, C.; Laventman, G.; Manevich, Y.; et al. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. In Proceedings of the EuroSys '18, Thirteenth EuroSys Conference, Porto, Portugal, 23–26 April 2018.
13. Riabi, I.; Ayed, H.K.; Saidane, L.A. A survey on Blockchain based access control for Internet of Things. In Proceedings of the 2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC), Tangier, Morocco, 24–28 June 2019; pp. 502–507.
14. Yakubov, A.; Shbair, W.; Wallbom, A.; Sanda, D. A Blockchain based PKI Management Framework. In Proceedings of the First IEEE/IFIP International Workshop on Managing and Managed by Blockchain (Man2Block) Colocated with IEEE/IFIP NOMS 2018, Tapei, Tawain, 23–27 April 2018.
15. Baldi, M.; Chiaraluce, F.; Frontoni, E.; Gottardi, G.; Sciarroni, D.; Spalazzi, L. Certificate Validation through Public Ledgers and Blockchains. In Proceedings of the First Italian Conference on Cybersecurity (ITASEC17), Venice, Italy, 17–20 January 2017.

16. Al-Bassam, M. *SCPKI:* A Smart Contract-based PKI and Identity System. In Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts, Abu Dhabi, UAE, 2–6 April 2017.
17. Roozbehani, M.; Povilionis, A.; Schunck, C.H.; Talamo, M. On the Fragility of Network Security Verification in Rare-Observation Regimes. In Proceedings of the IFAC 2017 World Congress, Toulouse, France, 9–14 July 2017.
18. Wagner, J. Why Performance Matters. Available online: https://developers.google.com/web/fundamentals/performance/why-performance-matters (accessed on 12 February 2020).
19. Gilad, Y.; Hemo, R.; Micali, S.; Vlachos, G.; Zeldovich, N. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, 28 October 2017.
20. Van Der Aalst, W.; Adriansyah, A.; De Medeiros, A.K.A.; Arcieri, F.; Baier, T.; Blickle, T.; Bose, J.C.; Van Den Brand, P.; Brandtjen, R.; Buijs, J.; et al. Process Mining Manifesto. In Proceedings of the Business Process Management Workshops, Clermont-Ferrand, France, 30 August–2 September 2011.
21. Ongaro, D.; Ousterhout, J. The Raft Consensus Algorithm. Available online: https://raft.github.io/ (accessed on 10 May 2019).