A Blockchain-based Certificate Revocation Management and Status Verification System

Yves Christian Elloh Adja*, Badis Hammi†, Ahmed Serhrouchni*, Sherali Zeadally‡
*Telecom Paristech, France
ello.adja, ahmed.serhrouchni@telecom-paristech.fr
†EPITA Engineering School, France
badis.hammi@epita.fr
‡University of Kentucky, USA
szeadally@uky.edu

***Abstract*—Revocation management is one of the main tasks of the Public Key Infrastructure (PKI). It is also critical to the security of any PKI. As a result of the increase in the number and sizes of networks as well as the adoption of novel paradigms such as the Internet of Things and their usage of the web, current revocation mechanisms are vulnerable to single point of failures as the network loads increase. To address this challenge, we take advantage of blockchains' power and resiliency in order to propose an efficient decentralized certificates' revocation management and status verification system. We use the extension field of the X509 certificate's structure to introduce a field that describes to which distribution point the certificate will belong to if revoked. Each distribution point is represented by a Bloom filter filled with revoked certificates. Bloom filters and revocation information are stored in a public blockchain. We developed a real implementation of our proposed mechanism in Python and the Namecoin blockchain. Then, we conducted an extensive evaluation of our scheme using performance metrics such as execution time and data consumption to demonstrate that it can meet the needed requirements with high efficiency and low cost. Moreover, we compare the performance of our approach with two of the most well-known/used revocation techniques which are Online Certificate Status Protocol (OCSP) and Certificate Revocation List (CRL). The results obtained show that our proposed approach outperforms these current schemes.**

***Index Terms*—Authentication, Blockchain, Bloom filter, Certificate, Revocation, Decentralization, PKI, Security, X509**

I. Introduction

Since the emergence of the Internet in the world, it has been changing the way people live, interact, and conduct businesses. Today the Word Wide Web has become an integral part of our daily tasks and environment and is involved in huge amounts of data transfers every day. Most of the exchanged data is sensitive and should be protected for users' privacy. In addition, numerous communications and requested services require authentication to be accepted or accessed. To handle these security requirements, multiple mechanisms were proposed, and the most common solution is the Public Key Infrastructure (PKI). Indeed, the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols, coupled with a Public Key Infrastructure provides authentication via certificate chains and private communication via encryption.

A public key infrastructure (PKI) is a set of authorities, policies, and procedures needed to manage public-key mechanisms. It is a set of authorities and protocols that binds public keys with respective identities of entities. The binding is established through a process of registration and issuance of certificates. Thus, a PKI creates, manages, distributes, uses, stores, and revokes these defined certificates [1].

The ability to revoke previously-issued certificates is critical to the security of any PKI, that is, to invalidate a certificate before it expires [2] due to many reasons such as the certificate's private key compromise or the fraudulent behavior of the certificate's owner. Certificate revocation is essential to authentication/authorization mechanisms that use certificates and their absence in the authentication process/cycle can lead to disastrous consequences [3]. For example, when the Heartbleed SSL/TLS vulnerability was announced [4], more than 80,000 SSL certificates were revoked in the week following the publication [5]. The Heartbleed bug made it possible for remote attackers to steal private keys from vulnerable servers. Most web server access logs are unlikely to show any evidence of such a compromise. Even if the certificate is replaced, the secure site could still be vulnerable if the vulnerable certificate has not been revoked. Indeed, a compromised certificate will remain usable by an attacker until its natural expiry date, which could be years away. A correctly positioned attacker, with knowledge of the old certificate's private key and the ability to intercept a victim's Internet traffic, can use the old certificate to impersonate the target site (e.g. using phishing techniques). Another example is described by [6] where a security incident related to certificate revocation checking made headlines. It was discovered that a legitimate website was hosting a malicious Java application that installed malware on the computers of people who visited the site. It was found out that the certificate of the website used in the attack had indeed been revoked and the infected victims did not verify the revocation status of the certificate.

There exist numerous revocation methods and systems such as Certificate Revocation Lists (CRL), Online Certificate Status Protocol (OCSP), Certificate Revocation Tree (CRT), and so on. Nonetheless, each of the existing techniques suffers from some different issues such as scalability support, financial and additional computational costs, users' privacy exposure, and so on [7].

Main research contributions of this work

We believe, like many researchers [8][9][10][11][12], that blockchains represent a very promising technology for the development of decentralized and resilient security solutions. We summarize the main contributions of this work as follows:

- Relying on the power and resiliency advantages of blockchains, we propose an efficient decentralized certificates' revocation management and status verification system.
- We use the extension field of the X509 certificate's structure to introduce a field that describes to which distribution point the certificate will belong to if revoked. Each distribution point is represented by a Bloom filter filled with revoked certificates. Bloom filters and revocation information are stored in a public blockchain.
- Using bloom filters we drastically minimize the time needed to obtain the revocation information compared to the existing works.
- We propose an approach fully compatible with the current web standards. The approach does not require any modification to be implemented in a web context.
- We present an implementation of our approach based on the public blockchain *Namecoin*.
- We conducted an extensive evaluation of our proposed revocation system and the results (using performance metrics such as time and data consumption) obtained demonstrate that it can meet the needed security and performance requirements.
- We compare the performance of our approach with two of the most well-known revocation techniques which are OCSP and CRL and the results show that our proposed approach outperforms these current schemes.

The remaining of the manuscript is organized as follows: Section II describes the existing works and techniques regarding certificate revocation. Then, Section III describes our revocation and status verification approach. Afterwards, Section IV describes our implementation and presents a performance evaluation of our proposed approach as well as the results obtained. Finally, Section V concludes the paper and points out future research perspectives.

II. Related work

There exist numerous schemes and proposals that manage and improve certificates revocation mechanism. In this section, we describe some of the most well-known ones, that are deployed by actual systems and standards.

A. Main certificate revocation mechanisms

Certificate Revocation List A Certificate Revocation List (CRL) [13][14] contains the list of revoked certificates, dated and signed by a Certification Authority (CA) and is periodically published. To check the validity of a certificate, the verifier must send a request to the publication server hosting the corresponding CRL, with argument the identifier of the CA in charge of the certificate; it then receives the last CRL generated by the CA; it must then check the CRL signature and its validity, and then searches for the certificate in the CRL.

The advantage of the CRL is its simplicity, its wealth of information and its low risk. However, the size of the CRL is its major disadvantage, because the bandwidth required for the update and verification is very high, which greatly limits its extensibility.

To ensure its freshness, the CRL contains the date of the next update (of the CRL). As a result, users who require fresh revocation information will want to retrieve the new CRL, all at the same time. This may cause an implosion of CRL requests which can add additional load on the server distributing the CRL, making it a possible single point of failure.

There are other variants, which represent extensions and enhancements to the CRL method. Next, we describe some of the most well-known ones.

Delta-CRL scheme [13] represents a list that contains all the non-expired certificates that have been revoked since the last CRL was published. Delta-CRL was designed as a solution to address the scalability issue of downloading CRLs. Indeed, the client does not have to download the entire CRL each time. However, the revocation information can only be used after the association of the Delta-CRL with the main CRL. The Delta-CRL represents only a fraction of the CRL. Thus, if the CRL size increases, the time and computation complexity of the revocation verification increases also.

CRL Distribution Points (CRL-DP) [13] represents another proposal to address scalability. Its main idea is the fragmentation of the CRL into small fragments. Each fragment is associated with a CRL distribution point which can be located on different hosts or on different directories of the same host. Each certificate has a pointer to the CRL-DP to which it will belong if revoked. Hence, there is no need to either search through the distribution points or have a prior knowledge of revocation information location [15]. The Delta-CRLs can also be used in the CRL-DP.

However, this solution has one main drawback: the non-uniform growth of the fragments. More specifically, when a certificate is established, it is decided in which fragment it will belong if revoked. Thus, it suffers from an unbalanced load on the distribution points because (1) some distribution points have more revocations than others; and (2) some CRL segments are requested more than the others [15]. Moreover, when CRL-DP technique is used, the CRL segmentation is permanently fixed for the lifetime of the certificates involved. Thus, the CA is required to define a static fragmentation before issuing the certificates which, represents an additional difficulty.

Dynamic CRL Distribution Points [16][17] also called *Enhanced CRL Distribution Point* [18] provides a solution to the CRL Distribution Points non-uniform growth issue. In this technique, two CRL extensions are defined: (1) *CRL Scope Field* which allows scope statements[1] to be defined and associated with CRL distribution points; and (2) *Status*

[1]a *scope* statement represents a range of certificates covered by a CRL partition.

Referral Fields which can be used to modify the partitioning for CRL Distribution Points by using the scope statements to refer to a CRL Distribution Point [15]. If an end entity wants to verify the validity of a certificate it can obtain a CRL which contains a *Status Referral Extension* when the certificate's enhanced CRL distribution point is referenced. This extension may include a scope that covers the verified certificate and a pointer to a new location for the CRL for the certificate in question [15].

Another alternative to CRL scheme is the **Certificate Revocation Status Directory (CRS Directory)** [19][20]. In a system that adopts the CRS technique, the certificate structure is extended with two additional fields of 100 bits each [15]. Every day, the CA sends signed statements to the CRS Directory about the status of single issued certificates. There are signed statements for every non-expired certificate.

When a user inquires about a certificate revocation status, the CRS Directory replies with information which the user can use to verify the requested status. The CRS approach decreases the communication load between the server and end entities which makes it achieve an overall performance gain compared to CRL approach. However, it considerably increases the communication load between the server and the CA.

Certificate Revocation Tree (CRT) [21] is commonly a *Merkle hash tree* which represents all certificate revocation information of a given PKI domain where a set of statements about certificate serial numbers are provided in the leaves. More specifically the leaves are ordered by certificate serial number in a logical order where two adjacent certificates $Cert_i$ and $Cert_j$ are revoked certificates, but no certificate number between them is revoked. Thus, it provides the information whether a certificate is revoked or not. Indeed, the path from the root to the appropriate leaf represents the proof about the state of a requested certificate.

The main advantage of this approach is that we do not need the entire CRL in order to provide a certificate verification. Nonetheless, its main weakness is its update since any change in the set of revoked certificates may result in the recomputation of the entire CRT which results in a continuous workload [22].

The **Online Certificate Status Protocol (OCSP)** [23] is an online revocation system which relies on a request/response mechanism. The revocation information is available on a server called the OCSP responder which receives it directly from the CA. The OCSP mechanism is designed to check and request exclusively the revocation status. The mechanism relies on a third party. Indeed, the OCSP responses are not signed by the CA. Hence, the revocation server must be trusted by the CA. If an end entity wants to verify the status of one or more certificates, it sends an OCSP request to the OCSP responder. The latter checks the revocation status information of the certificate(s) and replies with an OCSP response. The latter must be signed by the responder server.

The OCSP approach addresses the problem of low timeliness as well as the problem of revocation information update. However, its suffers from some drawbacks, mainly, (1) since the approach is centralized, the OCSP server represents a single point of failure [15]; (2) the OCSP responder verifies the revocation status of a certificate without checking the validity of its serial number and if it belongs to the CA. Thus, a malicious user can flood the server with verification requests for certificates that do not belong to the CA, making the server work intensively which can cause its denial of service; (3), it was proven that OCSP lookups are costly, especially in time [24][25], which increases the client side latency; (4) OCSP is an on-line scheme which makes it ineffective for offline systems. (5) OCSP may provide real-time responses to revocation queries, however it is unclear whether the responses actually contain updated revocation information. Some OCSP responders may rely on cached CRLs on their backend [25]. Finally, (6) the OCSP approach introduces a privacy risk. Indeed, the OCSP responders know which certificates are being verified by end users and they can therefore track the sites a user is visiting [25].

To resolve the OCSP approach's drawbacks, the **OCSP stapling** [26] approach was proposed. In this solution, the web server itself requests OCSP validation which it passes as a response to inquiring clients. Stapling removes the latency involved with OCSP validation because the client does not need an additional round trip to communicate with the OCSP responder to check the certificate's validity [25]. It also addresses the privacy issue since the OCSP responder does not have access to knowledge about a web site's visitors. Nonetheless, it does not resolve the problem of scalability due to the single point of failure.

The **Simple Certificate Validation Protocol (SCVP)** [27] is a more general request/response scheme than OCSP because it handles the entire certificate verification process rather than the sole verification of the the revocation status [15]. However, since it relies on a centralized server, it suffers from almost all the drawbacks described for OCSP.

B. Comparison of certificate revocation mechanisms

Table I presents a comparison of the different revocation schemes. The comparison is based on five metrics:

1) *Scalability*: describes how the approach behaves if the number of users or/and the revocation rate increases. We define three levels: *Low* level states that the approach is (or will soon) unable to face the current systems requirements; *Medium* level states that the approach can meet current requirements, but cannot overcome the system's evolution especially considering IoT requirements [28]; *High* level states that the approach is able to meet current and future requirements.
2) *Connectivity*: describes the connectivity status (online/offline) that the relying party must adopt in order to ensure the reliability.
3) *List type*: describes the type of the list used such as blacklist, whitelist or a combination of both.
4) *Real-time Service*: defines the capacity of the approach to give real-time information to the end-user.
5) *Additional cost*: indicates if the described approach is more costly in computational cycles than the CRL approach relying on the related works and their results. We

Approach	Scalability	Connectivity	List type	Additional cost	Real-time service	Privacy exposure
CRL	Low	offline, online	Blacklist	/	No	No
Delta-CRL	Low	offline, online	Blacklist	No	No	No
CRL distribution points	Medium	offline, online	Blacklist	No	No	No
Dynamic CRL distribution Points	Hight	offline, online	Blacklist	Yes	No	No
CRS	Low	offline, online	Blacklist, Whitelist	Yes	No	Yes
CRT	Low	offline, online	Blacklist, Whitelist	Yes	No	Yes
OCSP	Medium	Online	Blacklist	Yes	Yes	Yes
OCSP Stapling	Medium	offline	Blacklist	Yes	Yes	No
SCVD	Low	offline, online	Blacklist	Yes	Yes	Yes

Table I: Summary of revocation solutions for X509 certificates

chose to define CRL approach for comparison because it represents the most well-known used technique.

6) *Privacy exposure:* indicates if the approach uses a responder that can know which certificates are being verified by end users and it can therefore track the sites a user is visiting [25].

From Table I, we note that the majority of existing techniques incur additional costs and they do not provide real time information. Moreover, except for the Dynamic CRL Distribution Points approach [16], all the existing approaches cannot meet the future scalability requirement. Additionally, most of the techniques work in both offline and online mode. From this comparison we can conclude that none of these techniques can meet all the required needs for a resilient revocation mechanism.

Thus, it is necessary to design a decentralized revocation mechanism that avoids the single point of failure and supports networks scalability. Moreover, it must provide real time revocation information. Also, it must respect users privacy and resolve the problem of users' privacy exposure

C. Blockchain based revocation proposals

The distributed, event-recording and non-reproducibility features of the blockchain technology make it a desirable technology for PKI design and deployment [8][9]. Indeed, blockchain features help in meeting the major challenges of traditional PKI infrastructures: (1) since the blockchain-based PKI solutions are distributed; they have no centralized point of failure. (2) The trust is built based on the majority vote of the miners; Hence, there is no single trusted third-party and it does not require prior trustworthiness in the system. And (3), The blockchain technology has several open-source implementations which helps in building cost-effective solutions [8]. Next, we describe several blockchain-based PKI approaches. We focus on their revocation management.

Certcoin [29], is a completely decentralized PKI that leverages the consistency offered by Namecoin blockchain [30][31] to provide a strong identity retention guarantee. Certcoin uses five functions: registration, update, lookup, verification and revocation. During the registration, a user generates its own private and public keys locally. Then it submits a transaction of the public key and its signature to the blockchain. The blockchain network verifies the transaction signature and checks if this ownership was not registered before in the system. If the verification is successful, the (ID, public key) tuple is added to the blockchain; otherwise, it is dropped. Certcoin defines a PKI scheme that addresses some of the problems discussed earlier. However, it still suffers from numerous shortcomings such as the high costs in mining and public keys lookups and verifications [8]. Moreover, there is no real verification of the ID linkability to a registered public key. Finally, since we are interested in revocation techniques in this work, in Certcoin, an owner of an identity ID can revoke its public key simply by posting a transaction to the blockchain. Thus, the revocation process is completely handled by the owner himself/herself which can lead to different shortcomings such as: (1) It can be a difficult task for a user to handle the revocation by himself/herself because it needs some knowledge about how to proceed. Moreover, a user could not know about if his/her keys have been compromised. (2) A malicious user will not revoke his/her keys because he/she is acting maliciously. (3) To verify a certificate's status, the mechanism must first ensure that the certificate is not revoked by verifying the revoked certificates that are published in the blockchain, which implies browsing the blockchain. However, it is well known that searching in the blockchain can be very costly in terms of time.

Axon *et al.* proposed a Privacy-Aware Blockchain-Based PKI (PB-PKI) [32], an adaptation of Certcoin to make it privacy-aware. PB-PKI, does not publicly link identity with a public key. Rather, it provides unlinkable short-term key updates and user-controlled disclosure wherein a user's identity and previously used public keys can be disclosed either by the user himself/herself, or through consensus of a network majority. However, the revocation mechanism is still the same as in Certcoin. Hence, it inherits all its shortcomings.

Leiding *et al.* proposed Authcoin [33]. Authcoin is similar to Certcoin. However, it combines a challenge response-based validation and authentication process for domains, certificates, email accounts and public keys with the advantages of a blockchain-based storage system. As a result, Authcoin does not suffer from the weaknesses of existing solutions and it is much more resilient to sybil attacks. But Authcoin uses the same revocation mechanism as Certcoin and therefore also inherits the same shortcomings.

AL-Bassam *et al.* [34] proposed SCPKI: A Smart Contract-based PKI and Identity System as an alternative PKI system based on a decentralized and transparent design using a web-of-trust model and a smart contract on the Ethereum blockchain. The smart contract of SCPKI centers around the entity, which publishes a set of attributes, signatures, and revocations on the blockchain for its identity. Each entity is

Approach	Issue(s) addressed	Implementable on current web architecture?	Time cost	Blockchain used	Delegation of the revocation task to the user
Certcoin [29]	-Centralization of trust -CA misbehavior	No	High	Namecoin	Yes
PB-PKI [32]	User privacy	No	High	Namecoin	Yes
Authcoin [33]	-Centralization of trust -sybil attacks	No	Hight	Namecoin	Yes
SCPKI [34]	Centralization of trust	No	Low	Ethereum	Yes
Yakubov *et al.* [7]	-Centralization of trust -CA misbehavior	Yes	High	Any blockchain with smart contract	No
Pemcor [7]	-Centralization of trust -CA misbehavior	No	High	Any blockchain	No
IKP [35]	-CA misbehavior	No	High	Any blockchain	Yes
Blockstack ID [36]	-Centralization of trust -CA misbehavior	No	Hight	Namecoin	Yes
Cecoin [37]	-Centralization of trust -CA misbehavior - MITM attack	No	High	Any blockchain	Yes
Proposed approach	-Certificate revocation management -Certificate status verification	Yes	Low	Namecoin	No

Table II: Summary of blockchain-based revocation solutions

represented by an Ethereum address. Publishing an attribute to an entity's identity binds the identity to the attribute. Finally, a dedicated function allows entities to revoke their own signatures and the keys revocation status can be checked directly in the blockchain. We note that the SCPKI revocation technique suffers from the same shortcomings as Certcoin because the revocation task is provided by the owner himself/herself.

Yakubov *et al.* [7] proposed a blockchain-based PKI framework to manage X.509 certificates. They extend the standard X.509 certificate to be compatible with blockchain-based PKI approach, thanks to X.509 extension fields that they used to embed blockchain meta data. The main idea of the proposed framework is that each CA has a dedicated smart contract that executes the CA functions as in a traditional PKI. The smart contract acts on two lists: a white list for the created certificates and a black list for the revoked ones. Thus, when revoked, the hash of the certificate is added to the blacklist. Even if the revocation method is efficient, it forces browsing the blacklist to verify if a given certificate is revoked or not for each certificate's status verification which can be very costly knowing that it requires searching a hash in the blockchain block by block.

Yakubov's proposal [7] is very similar to Corella's *et al.* proposal that is called Pomcor [38][39]. In Pomcor, a CA issues a certificate to a subject as usual, except that it does not sign it. Instead, it stores a cryptographic hash of the certificate in a blockchain store that it controls, dedicated to storing hashes of issued certificates. If the certificate is compromised, the CA revokes it by storing its hash in another blockchain store that it controls, dedicated to storing hashes of revoked certificates. Thus, when a subject presents the certificate to a verifier, the latter acts exactly as in [7]. Hence, this approach suffers from the same shortcomings as [7].

Matsumoto *et al.* proposed Instant Karma PKI (IKP) [35], an automated platform for defining and reporting CA misbehavior that incentivizes CAs to correctly issue certificates and detectors to quickly report unauthorized certificates. IKP allows domains to specify policies that define CA misbehaviors, and CAs that sell insurance against misbehaviors. IKP is very costly and is a complex implementation [8] because it requires modeling the CA behavior. Furthermore, the proposed solution solves mainly the problem of misbehaving CA, which does happen very rarely.

Ali *et al.* proposed Blockstack ID [36], a Namecoin based PKI implementation. Blockstack ID modifies Namecoin by adding another name-value pair dedicated for the public keys. Thus, the public key is the value and the name is the identity of the owner. Blockstack implementation binds the user identity to an elliptic curve public key. Blockstack is one of the most popular blockchain-based PKI implementations [8]. However, it suffers from numerous shortcomings such as how the system handles public key updates, lookups, and revocations have not been considered. Also, the identity retention problem has not been addressed.

Qin *et al.* proposed Cecoin [37], a distributed blockchain-based PKI. In Cecoin, the task to distribute and manage certificates is accomplished by miners who separate the power from CAs. Moreover, it provides services of multi-certificate and identity assignment. Finally, it uses a structure of Merkle Patricia tree to implement a distributed Certificate Library which stores all valid certificates. However, Cecoin uses the same revocation mechanism as Certcoin, which makes it suffer from the same issues. Comparison of blockchain based revocation proposals

Table II summarizes the different approaches we have discussed above. It is worth noting that the existing approaches suffer from three main issues: (1) the majority are not applicable to the current X509 PKI standards and require a whole new protocol, which makes their adoption more complex and costly. It is therefore necessary to propose a revocation mechanism that is compatible with current web standards and can be directly adopted and implemented. (2) the blockchain

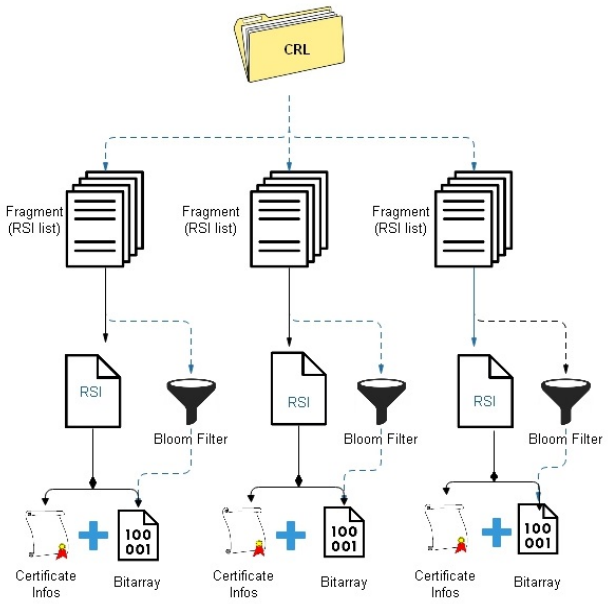

Figure 1: New Revocation information structure: The revocation records that are usually stored in a CRL will be stored in bloom filters. Each filter represents a distribution point.

based solutions are very costly in time because of blockchain browsing. Consequently, it is necessary that the proposed revocation scheme optimizes the search in the blockchain and avoids blind browsing.(3) In most approaches, the revocation task is delegated to the certificate's owner who can find the task too complex or the user can misbehave without revoking his/her own certificate. Consequently, we need to design a revocation system that spares the users from such a task and provides the legitimate revoker (which is the CA) with the necessary means to ensure such a task.

III. Proposed approach

The main goal of our approach is the proposal of a new certificate revocation and status verification scheme. Our approach relies on a public blockchain to store and disseminate the revoked certificates information. More specifically, in order to support scalability, our proposal uses the same principles as CRL distribution points. We use the extension field of the X509 certificate's structure to introduce a field that describes which distribution point the certificate will belong to if revoked. Each distribution point is presented by a Bloom filter filled with revoked certificates (See Figure 1). Bloom filters and revocation information are stored in a blockchain. Each time the CA revokes a certificate, it re-computes the corresponding Bloom filter and provides a new transaction to store it in the blockchain.

A. Background

Our approach relies mainly on (1) a blockchain and (2) Bloom filters. In this section we provide a quick summary of these concepts.

1) Blockchain: A blockchain is defined as a distributed database (ledger) that maintains a permanent and tamper-proof record of transactional data. A blockchain is completely decentralized by relying on a peer-to-peer network. More precisely, each node of the network maintains a copy of the ledger to prevent a single point of failure. All copies are updated and validated simultaneously [40].

Blockchain technology was created to solve the double spending problem in crypto-currency [41]. However, currently, numerous works [40][42][43][44] explore blockchain applications in multiple use cases and use them as a secure way to create and manage a distributed database and maintain records for digital transactions of all types.

The blockchain ledger is composed of multiple blocks, each block is composed of two parts. The first contains the transactions or facts (that the database must store), which can be of any type such as monetary transactions, health data, system logs, traffic information, and so on. The second is called the header and contains information (e.g., timestamp, hash of its transactions, and others) about its block, as well as the hash of the previous block. Thus, the set of the existing blocks forms a chain of linked and ordered blocks. The longer the chain, the harder it is to falsify. Indeed, if a malicious user wants to modify or swap a transaction on a block, (1) it must modify all the following blocks, since they are linked with their hashes. (2) Then, it must change the version of the block chain that each participating node stores [40].

2) Bloom filter: A Bloom filter is a space-efficient probabilistic data structure that supports set membership queries [45]. A Bloom filter is used to represent a set $S = \{s_1, s_2, ..., s_n\}$ of n elements through an array bit of m bits and using k independent hash functions $\{h_1, ..., h_k\}$ [46][47]. Bloom filters are useful for many different tasks that involve lists and sets. The basic operations involve adding elements to the set as well as querying for element membership in the probabilistic set representation. The structure offers a compact probabilistic way to represent a set that can result in false positives but not false negatives [45].

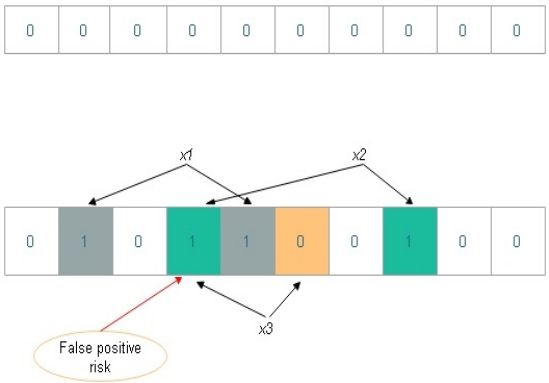

Figure 2: Overview of a Bloom filter

For a better understanding of Bloom filters, we present the following application example illustrated by Figure 2. In this

example:

- The set $S = \{x_1, x_2\}$ is composed of two elements ($n = 2$).
- The Bloom filter is a bitstring of 10 bits ($m = 10$).
- We use two hash functions $\{h_1, h_2\}$ ($k = 2$).

A hash function maps each item of the set S to a random number uniform over the range $\{1, ..., m\}$. Initially all the bits in the filter are set to zero. Afterwards, when adding an element x, the values of $h_1(x)$ and $h_2(x)$ (modulo 10) are calculated for the element, and the corresponding bit positions are set to one. After adding x_1 and x_2, the Bloom filter has positions 1, 3, 4 and 7 set to 1. To query the membership of an element x the same hash functions are applied on the queried element. Then, the bit positions corresponding to the results obtained are examined. If the two bits corresponding to the results of the hash functions after applying them to the element are set to one, that element is assumed to be present. Thus, if the membership of x_1 or x_2 is queried, the response will always be positive. However, in the case of x_3 membership query (Figure 2), the position 5 is set to zero and therefore x_3 is guaranteed not to be present in the Bloom filter.

Due to the limited space of the filter, the application of hash functions on a non-existing element can provide results that correspond to bit positions that are already set, leading to a false positive. Hence, querying a membership can result in false positives (claiming an element to be part of the set when it was not inserted), but not false negatives (reporting an inserted element to be absent from the set when it is not absent). The probability of having false positives is equal to the probability of having all positions of the Bloom filter set to one, for the k hash functions, as described by Equation 1.

$$P_{fp} = (1 - e^{\frac{-kn}{m}})^k \quad (1)$$

B. System's architecture

Our approach includes four entities such as depicted by Figure 3:

- **Certificate Authority (CA):** the CA is the entity that revokes certificates. In our approach, for each certificate revocation, the CA sends a transaction to the blockchain to add the new record. In other words, each revocation record added in the CRL implies a new transaction signed by the CA to add it in the blockchain in order to share the information.
- **Blockchain (BC):** the Blockchain represents the distributed ledger that stores the revocation information, called Revocation Status Information (RSI). More precisely, since we use an approach similar to distribution points, the global CA's CRL is summarized as a set of RSIs. Each RSI is mainly represented by a Bloom filter. Each filter is considered as a distribution point. Reading and downloading blockchain's records (blocks/transactions) do not need additional permissions and do not involve any costs. Thus, RSIs are visible to any user.
- **Server:** represents the web server, part of the TLS connection. In order to be authenticated, the server uses a certificate issued by the CA.

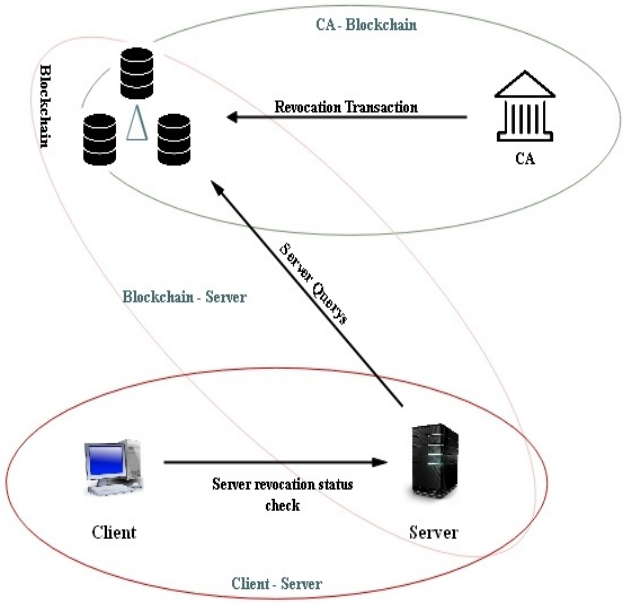

Figure 3: Overview of the system's architecture: the client receives the RSI from the web server during the TLS handshake. The sever downloads the RSI to which it belongs from the blockchain. The CA stores the different RSIs into the blockchain.

- **Client:** represents the entity that initiates the TLS connection. Before fully opening a TLS session with the server, the client must authenticate it using its certificate. This authentication step includes the revocation status verification. Without the revocation status verification, no session will be established.

The four entities described above operate so as to build three different sub-systems which work independently as described in Section III-C (depicted in Figure 3). The sub-systems are:

1) CA – Blockchain.
2) Blockchain – Server.
3) Server – Client.

C. System operation

In this section we present a detailed description of the operations of our system. To do so, we describe the functions of each of the sub-systems.

1) CA - Blockchain: The CA decides on a time interval when it disseminates and updates its revocation information. For example, each hour it informs, through network communications, about all the new revoked certificates. We refer to this time interval as T_u (update time). When the update time arrives, for each revoked certificate the CA builds a data structure called the Revocation Status Information (RSI) and performs a blockchain transaction to disseminate each RSI. More precisely, the CA calculates the Bloom filter, which contains the old revoked certificates as well as the new revoked one and disseminate it through a blockchain transaction (the RSI corresponds to a CRL distribution point equivalent).

Part 1		Part 2	
Data field	Size (bytes)	Data field	Size (bytes)
Bloom filter	350	Version	2
Hash	32	Crl ID	2
		Issuer	20
		Publication date	4
		Last update	4
		Next update	4
		Certificate ID	20
		Revocation date	4
Reserved		10	
Signature algorithm		2	
Signature		64	

Table III: Revocation Status Information (RSI) structure (case of *Namecoin* blockchain usage)

The RSI contains two main types of information: (1) the new Bloom filter and (2) the identity of the new revoked certificate (certificate added to the RSI). The bloom filter is calculated on all revoked certificates, and recalculated for each new addition of a revoked certificate. Table III depicts the structure of an RSI (implemented on *Namecoin* blockchain). The data structure is divided into two parts mainly to facilitate the creation of another data structure called Lightweight Revocation Status Information (LRSI) through the usage of *Merkle root hash* (described in more detail in Section III-C3). ***Part 1 :*** contains the following fields:

- Bloom filter: contains the Bloom filter which represents the main data of the RSI. More precisely, each new revoked certificate is hashed according to the chosen algorithms. Next, each bit array corresponding to one hash $modulo\ m$ is set to 1 thereby adding the new revoked certificate to the Bloom filter. When a certificate's revocation status is required, the filter is verified to check whether the bit arrays corresponding to the verified certificate are set to 1 or not.
- Hash: contains the hash of the first part. This field is not included when computing the *Merkle root* later. This hash is used during the verification of the integrity and authentication of the LRSI structure.

Part 2 : contains the following fields:

- Version: the version of the protocol in use.
- CRL ID: identifies the CRL distribution point.
- Issuer : the issuer of the CRL.
- Publication date: the publication date of the RSI.
- Next Update: the date of the next RSI update. When verifying the revocation status, the client must ensure that the current date is before this Next Update date to ensure the freshness of the RSI. The next update is set according to T_u. For example, if the current RSI was published in 2019-05-25T19:20+01:00 and $T_u = 60\ minutes$, the next update field will be set at 2019-05-25T20:20+01:00. Nonetheless, if the CA revokes more than one certificate (e.g., the CA revokes n certificates with $n > 1$), then, only the last RSI will have the next update corresponding to the next T_u. However, for the other $n-1$ RSIs, each of them, will have in its Next Update field, the same value as the Publication Date field of another RSI. Figure 4 presents a detailed example of this mechanism.
- Last Update: date of the precedent RSI update. When verifying the revocation status, the client must ensure that the current date is after this Last Update date to ensure the freshness of the RSI (see Figure 4).
- Certificate ID: serial number of the new revoked certificate.
- Revocation Date: revocation date of the new added certificate.

In addition to these two main parts, another common part completes the RSI structure. This part contains:

- Signature Algorithm: describes the signature algorithm used by the CA (e.g., Elliptic Curve Digital Signature Algorithm (ECDSA)).
- Signature: represents the signature applied by the CA on the *Merkel root (hash)* obtained from the two parts by using the chosen signature algorithm (e.g., ECDSA) (Figure 5 describes how the signature is provided).

In addition, there is a Reserved field for future extensions of the approach.

If no certificate is revoked in the T_u period, then the CA resends the last provided RSI after modifying the dates of the related fields as well as the signature.

For the filter implementation, we note, as demonstrated by [45] from Equation 1, that the false positive rate can be optimal for:

$$k = ln2 \cdot \frac{m}{n} \tag{2}$$

Hence, for a chosen false positive threshold, it is possible to determine the optimal filter dimension, as described by Equation 3:

$$m = \frac{n \cdot \ln p}{(\ln 2)^2} \tag{3}$$

2) Blockchain - Server: There are two types of blockchains: public and private. A public blockchain is characterized by the use of an unlimited number of anonymous nodes. Any actor can read, write, and validate transactions in the blockchain. In contrast, a private blockchain makes restrictions on the consensus contributors. Only the chosen trustful actors have the rights to validate transactions.

We have used a public blockchain due to its openness because any node in the world can access the blockchain's data and traffic without necessarily belonging to the network. The blocks and transactions transit clearly without any encryption, and only integrity protection and immutability services are performed on them. Moreover, generally a public blockchain is maintained by a strong community which ensures its reliability.

In our approach, the server must continuously check new transaction provided by its CA. In other words, the server must download each transaction related to its distribution point. Hence it downloads the RSI which includes the new Bloom filter of its distribution point. Moreover, for each downloaded RSI, the server must generate a structure called Lightweight Revocation Status Information (LRSI). The LRSI has the same structure and data fields as the RSI, except that it does not

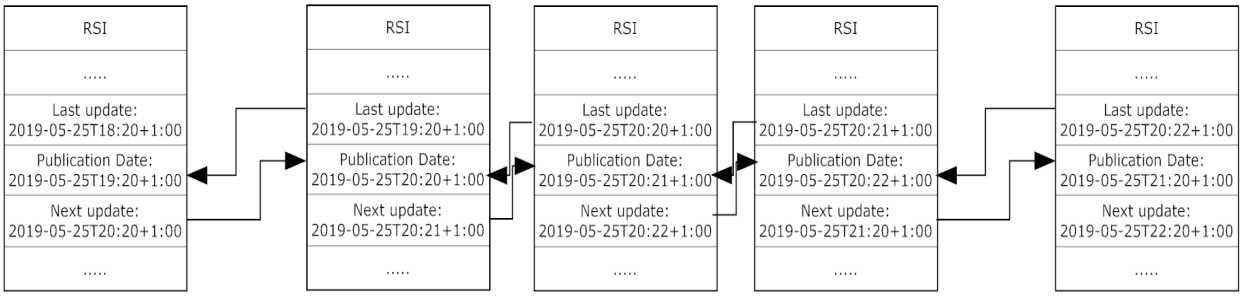

Figure 4: Next update/Last update fields mechanism of the RSI

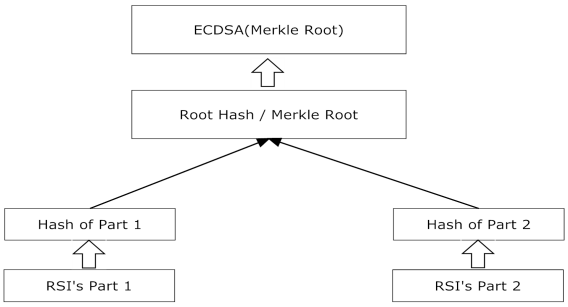

Figure 5: Signature method of the RSI

include the Bloom filter. Indeed, the server just removes the Bloom filter field and keeps all the others. The LRSI structure will be used in case of further investigation from the client side in order to detect false positives (more details of this structure's goal and usage are provided in Section III-C3).

Table IV shows the structure of the LRSI. As explained, it is very similar to the RSI structure except that it does not include the Bloom filter.

Part 1		Part 2	
Data field	Size (bytes)	Data field	Size (bytes)
Hash	32	Version	2
		Crl ID	2
		Issuer	20
		Publication date	4
		Last update	4
		Next update	4
		Certificate ID	20
		Revocation date	4
Reserved		10	
Signature algorithm		2	
Signature		64	

Table IV: Light Revocation Status Information (LRSI) structure (case of *Namecoin* blockchain usage)

3) Client - Server: In order to open a communication session with the server, the client initiates a TLS connection. However, in the handshake phase, the server sends its certificate, as well as the last downloaded RSI (timestamped and signed by the CA) to which it belongs. This way, the client can directly check through this RSI the certificate's revocation status without the need to open another connection with a another remote server such as OCSP. Furthermore, the client does not need to download a large CRL. Figure 6 depicts the modified TLS handshake steps.

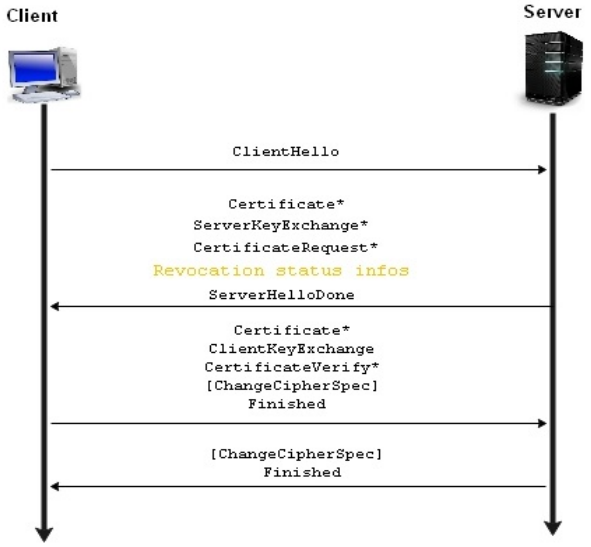

Figure 6: Modified TLS Client-Server handshake

We assume that each client provides basic protocol primitives to handle the modified TLS extension used and to treat the RSI and LRSI structures. Such an API is depicted in Algorithm 1. Algorithm 2 presents the details of the proposed approach.

When the client receives the RSI (see Algorithm 2), it verifies if the RSI corresponds to the certificate's distribution point. In other words, it verifies that the certificate will be added to this RSI if revoked. Next, it extracts the Bloom filter. Then, it applies the chosen hash functions on the server's certificate. Finally, it checks the bit positions in the filter which correspond to the hashes $modulo\ m$. Two results are possible:

1) The Bloom filter gives a negative answer (at least one bit array is set to 0). That is, the certificate is not included in the filter. As described earlier, a Bloom filter cannot indicate the presence of some data when it is not included (no false negatives). Hence, in the case of a negative response, the TLS session is established

Algorithm 1: Basic operations of the client

Function `InitiateConnection` (Node *client*, Node *server*) : DataArray `// Initiates a TLS connection and allows the reception of the server's parameters e.g., Certificate, RSI, etc.`
Function `getRSI` (DataArray *serverResponse*) : DataArray `// Extracts the RSI from the server's response`
Function `getLRSI` (Node *client*, Node *server*) : List of DataArray `// Allows the reception of the LRSI list`
Function `getCertificate` (DataArray *serverResponse*) : Certificate `// Extracts the server's certificate from the server's response`
Function `VerifyDistPoint` (String *crlID*, Certificate *certificate*) : Boolean `// Verifies if the received RSI corresponds to the distribution point of the server's certificate if revoked.`
Function `getFilter` (DataArray *rsi*) : BitArray `// Extracts the Bloom filter from the RSI structure`
Function `requestMembership` (Certificate *certificate*, BitArray *rsi*) : Boolean `// Requests the membership of the certificate in the Bloom filter by applying the hash functions needed`
Function `SortLrsiList` (List of DataArray *listRsi*): List of DataArray `// Sorts the different received LRSI according to the "Publication Date" field`
Function `VerifyLrsiList` (List of DataArray *listRsi*) : Boolean `// Verifies if an LRSI is missing relying on the field "Last Update" (or Next Update) as a pointer from one LRSI to another.`
Function `Error` (String *errorMessage*) : Void `// returns and error message`
Function `AbortConnection` (Node *client*, Node *server*) : Void `// Aborts the TLS connection`
Function `ContinueTlsConnection` (Node *client*, Node *server*) : Void `// The inquired certificate is not revoked and the TLS connection can continue its steps`

because we have the assurance that the certificate has not been revoked.

2) The filter provides a positive reply. In this case, since the filter can generate false positives, further checks must be made, as described below.

When the filter provides a positive reply, the client must execute further operations to ensure if the reply stands for a false positive or not. More specifically, the client asks the server to transmit all the revocation information generated by the CA to it. Thus, the server sends all the Light Revocation Status Information (LRSI) it generated. As described previously, the LRSI is a lightweight structure which is exactly the same as the RSI except that it does not include the Bloom filter field (Table IV depicts the structure of the LRSI). Indeed, the Bloom filter represents more than 70% of the RSI total size. Using LRSI instead of RSI achieves huge savings in terms of bandwidth and transmission time. When the client receives the LRSIs, it first verifies that the server has sent all the LRSI and did not omit one. This verification is made by checking the next update (or last update) field by following a pointer from one LRSI to another. Thereafter, the client verifies their integrity and authentication by verifying their signatures. Indeed, the client must ensure that the LRSIs it received contains the same exact data (generated by the CA) as the original RSIs and no fields have been modified. To achieve such verification, the client computes a *Merkle hash* of the LRSI by considering the following as leaves for the *Merkle tree*: (1) the hash field of the first part provided by the CA and (2) computing the hash of the second part. Then, it compares this hash result with the signature of the LRSI after its decryption using the CA's public key. Figure 7 shows the signature verification process. Indeed, the client must ensure that the revocation information is generated by the CA. However, the LRSI is generated by the server. Since the LRSI owns all the data fields generated by the CA except the Bloom filter, the hash field provided on the first part of the RSI (including the Bloom filter) is also provided by the CA. This hash is used as a leaf of the *Merkle tree*. If any of the LRSI's fields is modified, the *Merkle root* obtained will be different from the one computed by the CA, which causes the non verification of the ECDSA signature (that is applied on the *Merkle root*).

Finally, the client inspects, one by one, all the Certificate ID fields of the LRSIs. If the Certificate ID of the server is found, it means that the certificate was revoked and in this case the TLS connection aborts with fatal error. Otherwise, the TLS connection continues (because it was a false positive). For optimization concerns, we consider that the Certificate ID inspection is provided after the signature verification of each LRSI (for each LRSI, we verify its signature, then the *CRL ID* correspondance before verifying the next LRSI).

IV. Implementation and Evaluation

A. Implementation

To implement our approach, we used *Namecoin* blockchain [30][31]. *Namecoin* is a fork of *Bitcoin* which aims to provide a decentralized DNS. Indeed, it implements the top level domain .bit, which is independent of the Internet Corporation for Assigned Names and Numbers (ICANN)[2]. It uses the Proof of Work (PoW) consensus system. Table V describes the main features of *Namecoin*.

To implement our approach, any public blockchain can be used because they all allow data storage (e.g., Bitcoin[3], Litecoin[4], dash[5] or any other). Our proposal does not require the use of a blockchain that implements smart contracts. However, such blockchains (e.g., Ethereum[6], cosmos[7], tezos[8], metahash[9] or any other) can be used. Each blockchain has its own advantages and drawbacks to our approach. However, the majority of the contract-less blockchains offer a limited space to store information by transaction. Regarding the contract based blockchains, they need the development of contracts in order to read and write into the blockchain which adds

[2]https://www.icann.org
[3]https://bitcoin.org/
[4]https://litecoin.org
[5]https://www.dash.org
[6]https://ethereum.org/
[7]https://www.crypto-sous.fr/cosmos-network/
[8]https://tezos.com
[9]https://metahash.org

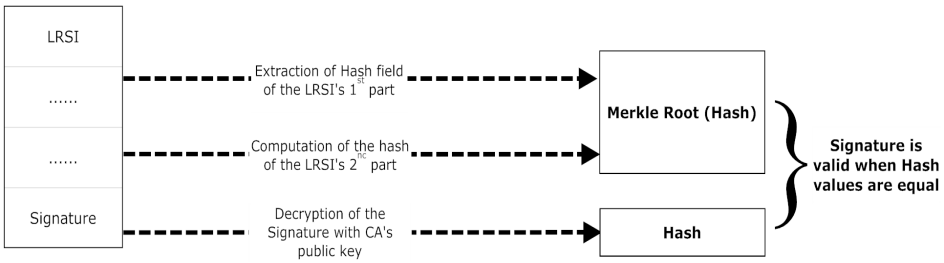

Figure 7: Signature verification of the LRSI structure

Algorithm 2: Proposed revocation status verification

```
rsi, lrsi : DataArray
lrsiList : List of DataArray
serverResponse : DataArray
bloomFilter : BitArray
client, server : Node // The client and server
    involved in the TLS connection
certificate : Certificate
begin
    serverResponse ← InitiateConnection (client,
      server);
    certificate ←getCertificate (serverResponse);
    rsi ← getRSI (serverResponse);
    if (VerifyDistPoint (rsi.crlID, certificate))
      then
        bloomFilter ← getFilter (rsi);
        if (requestMembership (certificate,
          bloomFilter)) then
            lrsiList ← getLRSI (client, server);
            lrsiList ← SortLrsiList (lrsiList);
            if (VerifyLrsiList (lrsiList)) then
                for each lrsi in lrsiList do
                    if certificate.ID == lrsi.CertificateID
                      then
                        Error ("Revoked certificate");
                        AbortConnection (client,
                          server);
                        Break;
                ContinueTlsConnection (client,
                  server);
            else
                Error ("Missing LRSI");
                AbortConnection (client, server);
                Break;
        else
            ContinueTlsConnection (client, server);
    else
        Error ("CRL ID mismatch");
        AbortConnection (client, server);
end
```

Data field	Feature
Type	Public blockchain
Feature	Fork of Bitcoin
Average transaction fee	0.00028 $
Block time	12 minutes 19 seconds
Transaction average /h	23
Blockchain dimension	6.29 GB

Table V: *Namecoin* features (02/12/2020)

unnecessary complexity to our approach. Moreover, contracts incur additional latency during their execution. Thus, we opted for *Namecoin* because of the following three reasons:

1) It allows data storage in the form of key/value pair, which is a suitable solution for our approach. Users are able to store keys along with their values which are 520 bytes in size.
2) The daily volume of transactions is relatively small which facilitates the data search in the blockchain.
3) Transactions fees are very low (the average transaction fee is about $0.00028 USD (accessed on 02/12/2020))[10]

As described earlier, in order to support scalability, we use an approach similar to distribution points. Each distribution point is represented by a distinct RSI. Considering the RSI structure described earlier, the fields (without considering the Bloom filter field) need 170 bytes of storage. Since in *Namecoin* values are limited to 520 bytes, our Bloom filter will have a size of 350 bytes (2800 bits).

In our implementation, the Bloom filter must be able to represent a maximum number of revoked certificates as well as being able to keep a reasonable probability of false positives. Thus, in order to define the optimal number of hash functions (k) we need to consider the number of revoked certificates (n) to be represented by one Bloom filter as well. We computed the false positive probability of the filter responses as described by Equation 1 by varying k and n by considering the filter size $m = 2800$. Figure 8 depicts the results obtained. It is worth noting that when $n = 100$ or $n = 250$, the false positive rate is always considerably low. However, the number of the revoked certificates we need to represent is also low. Considering these n values incurs the use of numerous Bloom filters which can be detrimental for the issuance/revocation ecosystem. Additionally, when $n = 1000$, the false positive probability is always over 26%, which represents a high probability, especially that, in this case, the system will check the certificate revocation through LRSI mechanism so often, leading to additional time and increased computation costs. Consequently, we argue that by considering the parameters $k = 3$ with $n = 500$ or $n = 750$ we can achieve the best compromise. Indeed, when $n = 500$, $P_{fp} = 7\%$ and when $n = 750$, $P_{fp} = 18\%$. Hence, one of these values can be chosen according to the use case.

[10]https://bitinfocharts.com/namecoin/

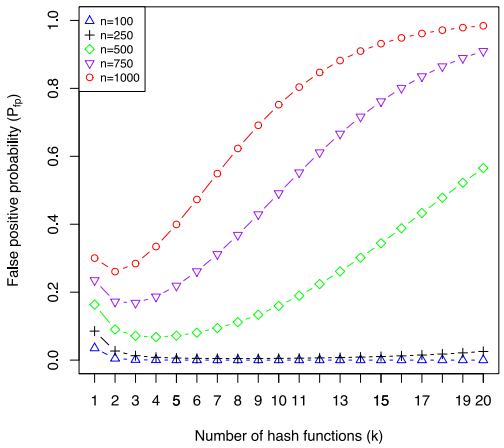

Figure 8: Evolution of false positive rate according to the number of hash functions (k) and the number of revoked certificates to consider (n)

B. Evaluation framework

To evaluate our system, we use a real implementation of our approach. To achieve this goal, we developed using *Python* language prototypes for the Server, the Client and the CA by relying on the *openSSL 1.0.2g* library [48]. We considered $k = 3$ where we used *Python mmh3* library which includes a set of fast and robust hash functions[11]. Regarding the blockchain implementation, we used *Multichain*[12] to simulate the *Namecoin* blockchain. *Multichain* is an open source blockchain platform which helps in the building and deployment of blockchain applications. It is fully configurable according to the user's needs and therefore it can be setup to reproduce the same functions as any other blockchain. We used this feature to simulate a *Namecoin* blockchain. Each entity (Client, Server and CA) was implemented on a different machine. Each machine was connected to the Internet through the provision with a different public IP address. Each of the Server and the CA hosts a copy of the blockchain. Table VI describes the technical features of the different machines used in our testbed.

Node type	CPU architecture	CPU operation mode	CPU max speed	RAM	Operation System
Server	core i7-3770, X86_64	64 bits	3.4 GHz	8 GB	Ubuntu 16.04
CA	Core i5-5300u, x86_64	64 bits	2.3GHz	8 GB	Kali Linux 4.19
Client	Core i5-5300u, x86_64	64 bits	2.3GHz	8 GB	Kali Linux 4.19

Table VI: Technical features of the testbed's machines

[11]https://pypi.org/project/mmh3/
[12]https://www.multichain.com

In this evaluation we study the time needed to obtain the revocation information as well as the quantity of data transmitted to fulfill this goal. These two parameters are among the most important ones for a revocation mechanism. Indeed, (1) the time needed to obtain a revocation information is responsible for the latency of connections which has an impact on the quality of service provided. (2) the amount of data needed must be the most optimal to minimize congestions on the network. For both indicators (time and data quantity) there are two possible scenarios that we analyzed separately:

1) The Bloom filter provides a negative membership response which means that the certificate is not revoked.
2) The Bloom filter provides a positive membership response. Knowing that the filter can generate false positives, further checks must be done using LRSI as described by Section III-C3.

In current PKI systems, the number of valid certificates is by far greater than the number of revoked certificates, which makes the probabilities of occurrence of the last two scenarios (positive or negative response of the filter) very different. Table VII depicts some statistics retrieved from [2] about the most well-known and largest CAs. According to the statistics presented, in average 5.3% of the certificates issued by a CA are revoked. This result includes the consideration of *GoDaddy* CA (26%) entry. However, this value is clearly an outlier regarding the set of the data depicted in Table VII. Indeed, by removing this entry, the average value is equal to 2.72% which represents a low percentage. In order to be fair in this evaluation, for the remainder of this paper, we consider that the average revocation ratio of certificates to be 5.3%.

CA	Unique CRLs	Certificates Total	Certificates Revoked	Avg. CRL size (KB)	Rate %
GoDaddy	322	1,050,014	277,500	1,184	26
RapidSSL	5	626,774	2,153	34,5	0.3
Comodo	30	447,506	7,169	517.6	1.6
PositiveSSL	3	415,075	8,177	441.3	1.97
GeoTrust	27	335,380	3,081	12,9	0.91
VeriSign	37	311,788	15,438	205.2	4.95
Thawte	32	278,563	4,446	25.4	1.59
GlobalSign	26	247,819	24,242	2,050.0	9.78
StartCom	17	236,776	1,752	240.5	0.73
Total rate					5.3

Table VII: Statistics on CRLs and Certificates of the most well-known CAs

Let us denote ep to be the event describing the occurrence of a revocation status request where the Bloom filter provides a positive answer and en to be the event describing the occurrence of a revocation status request where the Bloom filter provides a negative answer. We define P_n the probability that the event en occurs which leads to the first scenario where the client obtains a very fast response informing it that the certificate is not revoked. We define P_p to be the probability that the event ep occurs, which leads to the second scenario wherein additional verifications using LRSI structures are provided to decide about the validity of the corresponding certificate. We can calculate, with a certain accuracy, the occurrence probabilities (P_n and P_p) of these possible scenarios.

Symbol	Description	Symbol	Description
S	Set of n elements	n	Number of elements in the set S
m	Size of the bloom filter in bits	k	Number of hash functions used
P_{fp}	The probability of having false positives	T_u	Update interval of the revocation information
ep	The event describing the occurrence of a revocation status request where the Bloom filter provides a positive answer	en	The event describing the occurrence of a revocation status request where the Bloom filter provides a negative answer
P_n	The probability that the event en occurs	P_p	The probability that the event ep occurs
P_v	The probability that a requested certificate is valid	P_F	the false positive rate corresponding to the total number of requests
P_r	The probability that a requested certificate is revoked		

Table VIII: Notations table

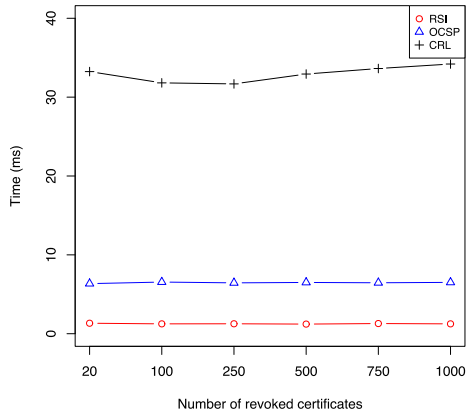

Figure 9: Time needed to provide a response on the revocation status of a non-revoked certificate

Let us consider an equiprobable scenario where all the CA's certificates have the same probability of being requested. Let us denote P_r to be the probability that a requested certificate is revoked, which is on average 5.3% ($P_r = 0.053$) of the revocation status requests cases. Consequently, the probability (denoted P_v) that a requested certificate is valid (not revoked) is calculated by Equation 4:

$$\begin{aligned} P_v = \overline{P_r} &= 1 - P_r \\ &= 1 - 0.053 = 0.947 \end{aligned} \tag{4}$$

The probability that a filter provides a false positive response (providing a positive response while the certificate is still not revoked) is $P_{fp} = 0.07$ when $n = 500$ or $P_{fp} = 0.18$ when $n = 750$ (according to Equation 1). Nonetheless, these P_{fp} rates correspond to the sole revocation status requests that should provide a negative response. In other words, the false positive rate is 7% of the set of 94% of the revocation status requests that are provided on non-revoked certificates ($P_v = 0.947$). Hence, according to Equation 5, the false positive rate corresponding to the total number of requests (denoted by P_F) is equal to $P_F = 0.947 * 0.07 = 0.066$ when $n = 500$ and $P_F = 0.947 * 0.18 = 0.17$ when $n = 750$.

$$P_F = P_v \times P_{fp} \tag{5}$$

Finally, the probability that the Bloom filter provides a positive response, which forces additional verifications using LRSI structures, as described by Equation 6 is equal to $P_p = 0.119$ when $n = 500$ and $P_p = 0.223$ when $n = 750$.

$$P_p = P_r + P_F \tag{6}$$

As a result, the probability that the Bloom filter provides a negative response, as described by Equation 7 is equal to $P_n = 0.881$ when $n = 500$ and $P_n = 0.777$ when $n = 750$.

$$P_n = \overline{P_p} = 1 - P_p \tag{7}$$

To compare the performance of our proposed approach with existing revocation management systems, we re-executed the same experiments used on our approach by considering a revocation system relying on: (1) OCSP and then on (2) a CRL. The implementations were achieved by using the *OpenSSL 1.0.2g* Library. For each of the experiments, we considered the number of revoked certificates based on the value of n. In other words, if we executed 100 experiments where $n = 500$, we also executed 100 experiments by considering a revocation system that relies on OCSP where 500 certificates are revoked, as well as 100 experiments by considering a revocation system that relies on a CRL that contains 500 entries. We chose to compare our approach with OCSP and CRL because they represent the main methods used for certificates revocation and status verification.

C. Numerical results and discussion

As described above, we have evaluated our approach according to the two possible scenarios: (1) the filter membership request provides a negative response and (2) the filter membership request provides a positive response.

1) Time consumption:

Negative response scenario

Figure 9 shows the results obtained with our implementation regarding the time needed to provide a response on the revocation status of a non-revoked certificate. More precisely, the time measured includes all the connection steps, namely (1) the client request for a connection, (2) the server reply and the RSI download time and (3) the RSI treatment duration by the client side. Each result presented represents the average computed on the results obtained from 100 experiments. During these experiments we requested the status of certificates that would not trigger a false positive causing the filter to provide a negative response. The standard deviations calculated on the results of each of the 100 experiments are very low ($< 0.08ms$).

We note that the time needed by our approach is very stable with 1.2 milliseconds (ms) on average and does not vary with

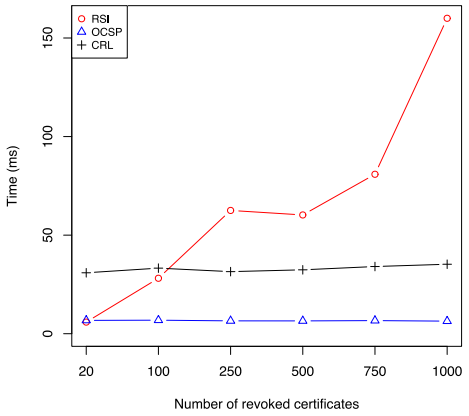

Figure 10: Time needed to provide a response on the revocation status of a revoked certificate

the number of the revoked certificated covered by the filter. Indeed, since the filter provides a negative response, no further verifications are needed. We recall that this performance occurs in most cases. Indeed, it occurs in more than 88% of the revocation status request cases when $n = 500$ and occurs in more than 77% of the cases when $n = 750$.

Figure 9 also depicts the performances of OCPS and a CRL-based systems. Both of these systems provide a stable time needed for a response: OCSP needs a time between $[6.35ms, 6.55ms]$ with a standard deviation between $[0.960ms, 1.34ms]$. The CRL-based system needs a time between $[31.68ms, 34.2ms]$ with a standard deviation between $[2.81ms, 4.65ms]$. Thereupon, based on these results, we conclude that our approach significantly outperforms the traditional systems in most cases.

Positive response scenario

Figure 10 shows the results obtained with our approach, OCSP and CRL-based implementations for the time needed to provide a response on the revocation status where the filter provides a positive response. OCSP and CRL-based systems kept their last stable results. Nonetheless, the time needed by our approach increases with n. In fact, each time the filter provides a positive response, additional verifications are provided as described in Section III-C3. Therefore, when the number of revoked certificates considered by the filter increases, the time needed for additional verifications increases. For example, when $n = 500$ the time needed to provide a response is on average equal to $60.24ms$. We recall that this use case scenario occurs only in less than 12% of the cases.

2) Quantity of transmitted data:

Negative response scenario

In this section we measure the data amount needed to exchange in order to ensure the revocation verification task. Since OCSP relies on a request/response mechanism, the amount of data exchanged is constant and is limited to request and response packets' sizes which makes them negligible. Hence, in this section we only compare the performance of our approach with the CRL-based approaches.

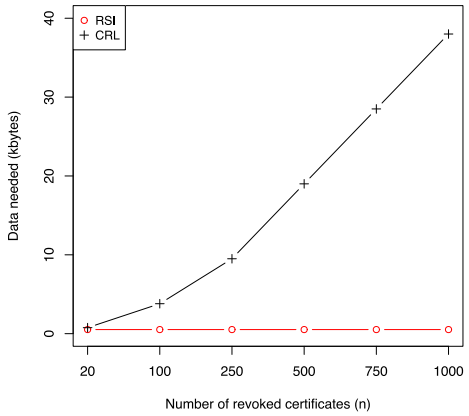

Figure 11: Amount of data needed to exchange to provide a response on the revocation status of a non-revoked certificate

Figure 11 depicts the amount of data needed by our approach and by CRL based approaches in order to provide a response on the revocation status where the filter provides a negative response. As with OCSP, in this case, our approach relies on a simple request with one response that contains the RSI structure (520 bytes). This performance does not change when the number of certificates (n) considered by the filter increases because the latter will provide a negative response. Nevertheless, CRL based approaches exchange more data when the number of the revoked certificates considered by the CRL increases. In fact, CRLs contain one entry for each certificate that is revoked. Thus, the size of the CRL (in bytes) is expected to correlate with the number of entries. According to [2], on average, each entry is 38 bytes.

Thus, our approach largely outperforms the CRL based approaches by exchanging a low amount of data to ensure the revocation status verification. Furthermore, in this case, the amount of data needed by our approach is very similar to the amount needed by OCSP [49], regardless of the value of n. We recall that this case occurs in most cases such as in more than 88% of the revocation status requests cases when $n = 500$ and also occurs in more than 77% of the cases when $n = 750$.

Positive response scenario

Figure 12 depicts the amount of data needed by our approach and by CRL-based approaches in order to provide a response on the revocation status when the filter provides a positive response. It is worth noting that our approach achieves the worst performance. Indeed, it must first download the RSI structure. When the filter extracted from the latter provides a positive response, the system must ensure whether it is a false positive or not. Hence, additional verification using LRSI structures is needed. This phase requires the downloading of all the LRSI structures. There are as many LRSIs as revoked certificates and each LRSI is 170 bytes. For example, when

$n = 500$, our approach needs 80.52 Kbytes to provide a response. However, we recall that this use case scenario happens only in less than 12% of the cases. Moreover, systems that have sufficient computational ressources may apply cashing techniques in order to not download the set of LRSIs each time as well as to optimize the verification process to decrease the time and data needed to provide a response.

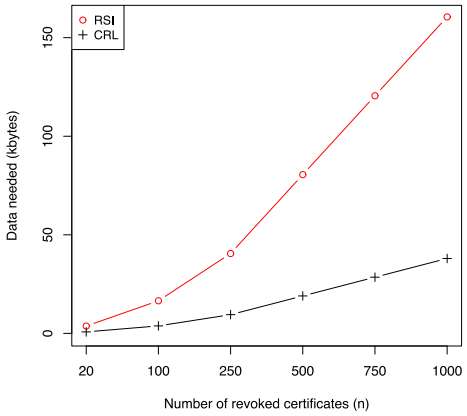

Figure 12: Amount of data needed to exchange to provide a response on the revocation status of a revoked certificate

3) Evaluation of security requirements : In this section we show how our approach satisfies the security and performance requirements discussed earlier in Section II-B and Section II-C.

Scalability: our system relies on a public blockchain, which, in turn, relies on a peer-to-peer network. It is well-known that peer-to-peer networks are one of the best scalable solutions [50]. Moreover, the client does not need to download the entire CRL. Instead it downloads the lightweight data structure directly from the connection's server which saves the time of the connection with the CA.

Connectivity: a mechanism such as OCSP cannot work without a stable Internet connection and continuous access to the CA. This is an important issue because most of the current browsers open only a security warning dialog to the user, or even worse, they do not alert the user when they cannot reach the OCSP responder and establish the connection with the server. Our approach overcomes this vulnerability because of its ability to work in an offline mode. Indeed, the servers host a copy of the blockchain that contains the revocation information. Thus, they can access this data without a connection to the CA.

Privacy: as described above, the OCSP responders know which certificates are being verified by end users and they can therefore track the sites a user is visiting [25], which represents a privacy breach that can be exploited. In our approach there is no communication with a third party responder. Each server sends to the client the RSI that may contain its certificate if it has been revoked. Then the verification is provided by the client side. Thus, there is no information exposure.

Authentication/RSI message substitution: since all RSI/LRSI are signed, if an attacker alters or substitutes a structure, it will be quickly detected at the client level because the signature will not correspond to the CA's public key.

Availability/DDoS protection: the highly decentralized architecture of blockchains makes them robust against DoS/DDoS attacks. Indeed, services are duplicated and distributed over different network nodes. That is to say, even if an attacker manages to block a node, it cannot block all the other nodes.

Revocation information falsification: In addition to the CA's signature, the RSI structures and thus the revocation information are protected through the blockchain architecture design. Indeed, if a malicious user wants to modify or swap a transaction on a block, first, it must modify all the following blocks because they are linked with their hashes. Then, second, it must change the version of the blockchain that each participating node stores which is almost impossible.

4) Cost evaluation: Our approach relies on a public blockchain which incurs costs that depend on the cryptocurrency used by the blockchain system. However, we argue that each security service provided incurs a cost which as long as it remains below the cost of potential damages, the security solution is worth implementing. Moreover, existing approaches such as OCSP and CRL are also costly.

One of the reasons we have used *Namecoin* is because of its low cost. Based on the low revocation rate, we can conclude that the certificate's revocation activity is also low. Thus, our approach requires a small number of transactions, at least one transaction each T_u. For example, if we consider (1) $T_u = 60\ minutes$, and (2) there are two certificate revocations per hour: based on the current average transaction fee[13] which is 0.00028 $, our system would need on average 0.40 $ per month. Furthermore, a blockchain is a fully autonomous system, which offers the advantage of removing the infrastructure maintenance cost compared to other approaches. With these low costs, we believe that our approach is significantly less costly than the other current systems, especially OCSP which needs the availability and maintenance of an infrastructure on a continuous basis.

We are aware that cryptocurrencies may vary a lot. Nonetheless, according to studies such as [51] and [52], the evolution of the cryptocurrencies rates will get more stable over time.

V. Conclusion

The certificates revocation management remains to be a recurring issue that continues to grow especially with the current evolution and openness of networks and their continuous adoption of new paradigms such as Internet of Things, cloud computing and so on. The current existing revocation management techniques suffer from some drawbacks such as (1) the centralization which leads to single point of failures, (2) financial and additional computational costs, (3) users' privacy exposure and many others. Thus, considering the evolution of networks and the openness/connection of the

[13]In this example we consider the current *Namecoin* average transaction fee applied on the date of 2^{nd} of December 2020

different use-cases, these drawbacks will lead to the incapacity of the existing approaches in ensuring a correct revocation management.

Blockchain based approaches resolve some of these issues. However, the proposed solutions are not compatible with the current X509 standards and their implementations require a whole new architecture of the web to be deployed.

In this context, we proposed a novel revocation management and status verification system that meets all the requirements and yields strong performance results. Our approach relies on a blockchain which makes it very resilient and completely decentralized in order to meet the evolution of networks as well as their scalability. Moreover, it is fully compatible with the current web standards and does not require any modification to be implemented, which to the best of our knowledge, such an approach has not been proposed yet. In addition, previously proposed blockchain based revocation mechanisms suffer from high time delays due to the time costs incurred by the blockchain browsing to find the needed transaction. We propose a mechanism that relies on bloom filters which drastically optimize the time needed to provide the revocation information. More precisely, our proposal uses the same principles as CRL distribution points. Each distribution point is presented by a Bloom filter filled with revoked certificates. Then, Bloom filters and revocation information are shared and disseminated using a public blockchain.

We implemented and evaluated our approach on a real testbed. This evaluation shows clearly the ability of our revocation system to meet the needed security and performances requirements as well as its capacity to outperform the existing approaches (OCSP and CRL-based systems).

For future works, we plan to focus on the worst case scenario of our solution, that is, when the filter provides a positive response. Indeed, we work to provide an alternative solution that avoids the downloading of all the LRSIs from the server.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable comments which helped us improve the content, organization, and presentation of this paper.

REFERENCES

[1] Jean Philippe Monteuuis, Badis Hammi, Eduardo Salles, Houda Labiod, Remi Blancher, Erwan Abalea, and Brigitte Lonc. Securing pki requests for c-its systems. In *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–8. IEEE, 2017.

[2] Yabing Liu, Will Tome, Liang Zhang, David Choffnes, Dave Levin, Bruce Maggs, Alan Mislove, Aaron Schulman, and Christo Wilson. An end-to-end measurement of certificate revocation in the web's pki. In *Proceedings of the 2015 Internet Measurement Conference*, pages 183–196. ACM, 2015.

[3] Liang Zhang, David Choffnes, Dave Levin, Tudor Dumitraş, Alan Mislove, Aaron Schulman, and Christo Wilson. Analysis of ssl certificate reissues and revocations in the wake of heartbleed. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 489–502. Association for Computing Machinery, 2014.

[4] Zakir Durumeric, Frank Li, James Kasten, Johanna Amann, Jethro Beekman, Mathias Payer, Nicolas Weaver, David Adrian, Vern Paxson, Michael Bailey, et al. The matter of heartbleed. In *Proceedings of the 2014 conference on internet measurement conference*, pages 475–488. Association for Computing Machinery, 2014.

[5] Paul Mutton. Certificate revocation: Why browsers remain affected by Heartbleed. Technical report, Netcraft, April 2014.

[6] Wayne Thayer. The Importance of Revocation Checking. Technical report, CA Security Council, March 2013.

[7] Alexander Yakubov, Wazen Shbair, Anders Wallbom, David Sanda, et al. A blockchain-based pki management framework. In *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*, 2018.

[8] Tara Salman, Maede Zolanvari, Aiman Erbad, Raj Jain, and Mohammed Samaka. Security services using blockchains: A state of the art survey. *IEEE Communications Surveys & Tutorials*, 21(1):858–880, 2018.

[9] Xiaoyang Zhu and Youakim Badr. Identity management systems for the internet of things: a survey towards blockchain solutions. *Sensors*, 18(12):4215, 2018.

[10] Konstantinos Christidis and Michael Devetsikiotis. Blockchains and smart contracts for the internet of things. *IEEE Access*, 4:2292–2303, 2016.

[11] Ana Reyna, Cristian Martín, Jaime Chen, Enrique Soler, and Manuel Díaz. On blockchain and its integration with iot. challenges and opportunities. *Future Generation Computer Systems*, 88:173–190, 2018.

[12] Achraf Fayad, Badis Hammi, and Rida Khatoun. An adaptive authentication and authorization scheme for IoT's gateways: a blockchain based approach. In *2018 Third International Conference on Security of Smart Cities, Industrial Control System and Communications (SSIC)*, pages 1–7. IEEE, 2018.

[13] David Cooper, Stefan Santesson, S Farrell, Sharon Boeyen, Rusell Housley, and W Polk. RFC 5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. *Internet Engineering Task Force (IETF), May*, 2008.

[14] P Yee AKAYLA. RFC 6818: Updates to the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. *Internet Engineering Task Force (IETF), January*, 2013.

[15] Andre Arnes, Mike Just, Svein J Knapskog, Steve Lloyd, and Henk Meijer. Selecting revocation solutions for PKI. In *Fifth Nordic Workshop on Secure IT Systems (NORDSEC 2000)*, pages 1–16. Citeseer, 2000.

[16] ISO/TEC ITU. Final proposed draft amandment on certificate extensions. pages 1–12, April 1999.

[17] X.509 ITU-T RECOMMENDATION. Information technology–open systems interconnection–the directory: Public-key and attribute certificate frameworks. *SERIES X: DATA NETWORKS, OPEN SYSTEM COMMUNICATIONS AND SECURITY*, pages 1–254, 2016.

[18] P Hallam-Baker and W Ford. Internet X.509 Public Key Infrastructure ENHANCED CRL DISTRIBUTION OPTIONS. *PKIX Working Group, Internet Draft, Internet Engineering Task Force (IETF), August*, 1998.

[19] Silvio Micali. Efficient certificate revocation, February 26 2008. US Patent 7,337,315.

[20] Moni Naor and Kobbi Nissim. Certificate revocation and certificate update. *IEEE Journal on selected areas in communications*, 18(4):561–570, 2000.

[21] Paul C Kocher. On certificate revocation and validation. In *International Conference on Financial Cryptography*, pages 172–177. Springer, 1998.

[22] Adam Slagell, Rafael Bonilla, and William Yurcik. A survey of PKI components and scalability issues. In *2006 IEEE International Performance Computing and Communications Conference*, page 64. IEEE, 2006.

[23] Stefan Santesson, Michael Myers, Rich Ankney, Ambarish Malpani, Slava Galperin, and Carlisle Adams. RFC 6960: X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. *Internet Engineering Task Force (IETF), June*, 2013.

[24] Emily Stark, Lin-Shung Huang, Dinesh Israni, Collin Jackson, and Dan Boneh. The case for prefetching and prevalidating tls server certificates. In *Network and Distributed System Security Symposium (NDSS)*, volume 12, 2012.

[25] Emin Topalovic, Brennan Saeta, Lin-Shung Huang, Collin Jackson, and Dan Boneh. Towards short-lived certificates. *Web 2.0 Security and Privacy*, 2012.

[26] Yngve Pettersen. RFC 6961: The Transport Layer Security (TLS) Multiple Certificate Status Request Extension. *Internet Engineering Task Force (IETF), June*, 2013.

[27] Trevor Freeman, Russell Housley, Ambarish Malpani, David Cooper, and William Polk. RFC 5055: Server-Based Certificate Validation Protocol (SCVP). *Internet Engineering Task Force (IETF), December*, 2007.

[28] Badis Hammi, R Khatoun, Sherali Zeadally, Achraf Fayad, and Lyes Khoukhi. Internet of Things (IoT) Technologies for Smart Cities. *IET Networks*, 2017.

[29] Conner Fromknecht, Dragos Velicanu, and Sophia Yakoubov. A decentralized public key infrastructure with identity retention. *IACR Cryptol. ePrint Arch.*, 2014:803, 2014.

[30] Kraft Daniel, Castellucci Ryan, Rand Jeremy, Roberts Brandon, Bisch Joseph, Colosimo Andrew, Conrad Peter, Bodiwala Ahmed, and Dam Lola. Namecoin. https://namecoin.org/, 2020.

[31] Harry A Kalodner, Miles Carlsten, Paul Ellenbogen, Joseph Bonneau, and Arvind Narayanan. An empirical study of namecoin and lessons for decentralized namespace design. In *WEIS*. Citeseer, 2015.

[32] LM Axon and Michael Goldsmith. Pb-pki: A privacy-aware blockchain-based pki. 2016.

[33] Benjamin Leiding, Clemens H Cap, Thomas Mundt, and Samaneh Rashidibajgan. Authcoin: validation and authentication in decentralized networks. *arXiv preprint arXiv:1609.04955*, 2016.

[34] Mustafa Al-Bassam. Scpki: a smart contract-based pki and identity system. In *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*, pages 35–40, 2017.

[35] 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, 2017.

[36] Muneeb Ali, Jude 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} 16)*, pages 181–194, 2016.

[37] Bo Qin, Jikun Huang, Qin Wang, Xizhao Luo, Bin Liang, and Wenchang Shi. Cecoin: A decentralized pki mitigating mitm attacks. *Future Generation Computer Systems*, 107:805–815, 2020.

[38] F Corella. Implementing a pki on a blockchain. *Pomcor Research in Mobile and Web Technology*, 2016.

[39] Karen Lewison and Francisco Corella. Backing rich credentials with a blockchain pki. *Tech. Rep.*, 2016.

[40] Mohamed Tahar Hammi, Badis Hammi, Patrick Bellot, and Ahmed Serhrouchni. Bubbles of Trust: A decentralized blockchain-based authentication system for IoT. *Computers & Security*, 78:126–142, 2018.

[41] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[42] Minhaj Ahmad Khan and Khaled Salah. Iot security: Review, blockchain solutions, and open challenges. *Future Generation Computer Systems*, 82:395–411, 2018.

[43] Arshdeep Bahga and Vijay K Madisetti. Blockchain platform for industrial internet of things. *J. Softw. Eng. Appl*, 9(10):533, 2016.

[44] Seyoung Huh, Sangrae Cho, and Soohyung Kim. Managing iot devices using blockchain platform. In *Advanced Communication Technology (ICACT), 2017 19th International Conference on*, pages 464–467. IEEE, 2017.

[45] Sasu Tarkoma, Christian Esteve Rothenberg, and Eemil Lagerspetz. Theory and practice of bloom filters for distributed systems. *IEEE Communications Surveys & Tutorials*, 14(1):131–155, 2012.

[46] Michael Mitzenmacher. Compressed bloom filters. *IEEE/ACM Transactions on Networking (TON)*, 10(5):604–612, 2002.

[47] Andrei Broder and Michael Mitzenmacher. Network applications of bloom filters: A survey. *Internet mathematics*, 1(4):485–509, 2004.

[48] OpenSSL Cryptography SSL/tLs toolkit: Welcome to OpenSSL. https://www.openssl.org/, 2020.

[49] Michael Myers and Hannes Tschofenig. RFC 4806: Online Certificate Status Protocol (OCSP) Extensions to IKEv2. *Internet Engineering Task Force (IETF), February*, 2007.

[50] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, and Steven Lim. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys & Tutorials*, 7(2):72–93, 2005.

[51] Kenji Saito and Mitsuru Iwamura. How to make a digital currency on a blockchain stable. *arXiv preprint arXiv:1801.06771*, pages 1–15, 2018.

[52] Ousmène Jacques Mandeng. Cryptocurrencies, monetary stability and regulation. Technical report, 2018.