

January 2019

ConsensusPKI

DATA DRIVEN PUBLIC KEY ECOSYSTEM BACKED BY
BLOCKCHAIN & FAULT TOLERANCE
VOLKAN KAYA

CYBER SECURITY ACADEMY | LEIDEN UNIVERSITY | volkankaya81@gmail.com

Student Number: s1954253

Supervisor: Assoc. Prof. Dr. Ir. J.C.A. van der Lubbe | TU Delft

Thesis Committee: Assoc. Prof. Dr. Ir. J.C.A. van der Lubbe | TU Delft
Assoc. Prof. Dr. Ir. F.A. Kuipers | TU Delft

Abstract

The X.509 standard and PKI ecosystem we use sits in the core of the confidentiality and integrity services on the internet. The X.509 ecosystem as we use has many structural flaws which can be abused by malicious parties to generate and use fake certificates in various man-in-the-middle attacks. In this thesis, we present the ConsensusPKI, a public blockchain supported data-driven PKI that addresses all known structural flaws of the X.509 standard. The certificates in the ConsensusPKI does not carry signatures, but it carries Merkle proofs which can be validated during the protocol handshake. The PBFT based consensus algorithms serve to detect non-honest or faulty CAs without the need to interrupt the end user communication. The ConsensusPKI has no trust and requires verification of the public keys each time it is interpreted. In ConsensusPKI, CAs conduct completely different role without explicit trust. A CA in the ConsensusPKI is an entity that never lies. All processes and the data model of the ConsensusPKI are built on this simple principle. The certificate issuance process of the ConsensusPKI is entirely transparent and rigorous verification process that guarantees that the requestors of the certificate issuance to have full control over the subject. To the best of our knowledge, the thesis introduces the following four new structures to the literature: a) the use of the Merkle proofs instead of signatures in the certificate files, b) the use of Practical Byzantines Fault Tolerance algorithm during interpretation of the certificates, c) the use of leading root blockchain to manage the data retention for blockchains, d) the use of consensus algorithms verifying identities to issue certificates.

Acknowledgment

After working for more than 15 years in an IT organization, building experience in multiple disciplines of IT, deciding to go back to “school” was one of the best decisions I made in my professional life. The master program was tough, especially combining it with a full-time job and having a family with two small kids, but very rewarding and challenging, providing me with new insights and inspiration to learn and develop.

The transdisciplinary approach of the Cyber Security Academy resulted in this thesis in which I combined insights, material, and information from all courses, but especially the following three; the Cyber Risk Management Course, the Course Scientific foundations for Cyber Security Research and the Cryptography Classes in the technical track. The Risk Management course opened my eyes to understanding the risk and impact involved with public key infrastructure. The Cyber Security Research course helped me to develop a better understanding of cybersecurity issues. Finally, the Cryptography Classes were the door that I had to go through to get to this point; being a well-rounded specialist in one of the core foundations of the Cyberspace.

I would like to especially express my appreciation for the expertise, help, and support I received from my thesis coach, Assoc. Prof. Dr. Ir. J.C.A. van der Lubbe of TU Delft. Furthermore, I want to thank my second reader for his part in the finalization of this master thesis, Assoc. Prof. Dr. Ir. F.A. Kuipers of TU Delft. A special thanks go to my long-term colleague, Dheeraj Katarya for getting me excited on the trust issues concerning the public key infrastructure, it being the turning point for me to focus my attention to this topic for my master thesis. A final thanks go to my lovely wife, Evren, and my two children, Sezen, and Ediz for being patient when needed and being supportive when called for.

Abstract	1
Acknowledgment	2
Glossary	6
Chapter 1) Introduction	8
1.1) Secure communication on the Internet, PKI and X.509 scheme	10
1.2) Structural flaws on the trust model of the X.509 scheme	12
1.3) Cryptographic functions and Signature Schemes	15
1.4) Emergence of Blockchain technology as a decentralized trust	15
1.5) Research Question	15
1.6) Contribution	16
1.7) Methodology & Reporting outline	17
Chapter 2) Prior research and proposed solutions	18
2.1) Proposed solutions without blockchain support	18
2.1.1) Log Servers	20
2.2) Proposed solutions with blockchain support	20
2.3) Comparison of Alternative Solutions	21
Chapter 3) Background	23
3.1) Cryptographic functions	23
3.1.1) Hash functions	23
3.1.2) Cryptographic Signatures	23
3.1.3) Cryptographic Puzzles & Zero-Knowledge Proofs	24
3.2) Domain Name System (DNS)	25
3.2.1) TXT records	27
3.2.2) DomainKeys Identified Mail (DKIM) Protocol	28
3.3) PKI ecosystem	29
3.3.1) Management Protocols	30
3.3.2) Certificate Structure and Extensions	31
3.3.3) Certificate Revocation Lists and Online Certificate Status Protocol	32
3.3.4) X.509 certification path validation	33
3.4) Blockchain Technology	34
3.4.1) Memory Pool & Timestamping	36
3.4.2) Merkle Trees	37
3.4.3) Distributed Network	37
3.4.4) Oracle	38
3.4.5) Practical Byzantine Fault Tolerance (PBFT)	38
3.5) Automatic certificate management environment (ACME)	38

Chapter 4) ConsensusPKI – A Blockchain based public key ecosystem	40
4.1) Ecosystem focused solution & general characteristics	40
4.1.1) Verifiable evidence instead of signatures	41
4.2) A General view of the ConsensusPKI	41
4.3) Blockchains, Merkle Trees & Certificate Structures	44
4.3.1) Root Blockchain structure	46
4.3.2) Certificates Blockchain	47
4.3.3) Evidences Blockchain	48
4.3.4) Certificates Merkle Tree	49
4.3.5) Evidences Merkle Tree	49
4.3.6) Root Blockchain structure for Blockchain Nodes	50
4.3.7) Certificates Blockchain for Blockchain Nodes	50
4.3.8) Evidence blockchain and Merkle tree for Blockchain nodes	50
4.3.9) Certificate structure & fields	51
4.4) Certificate issuance: Zero-knowledge identity proof backed by Blockchain & Consensus	51
4.4.1) Certificate issuance for CAs	57
4.5) Anchoring of Public keys of CAs on End-user systems	58
4.5.1) Distribution of Root and Certificate blockchains for CAs to End-user systems	58
4.6) Indication and interpretation on the client side: The public key query protocol	59
4.7) The honesty check and revocation of public keys for CAs and regular services	60
4.7.1) Non-honest CA validation	61
4.7.2) Revocation of Regular Public Keys	61
Chapter 5) Validation	62
5.1) Implementation Aspects	62
5.1.1) Considerations on CA Datastore on end-user systems	62
5.1.2) Considerations on the CA Certificate file structure	65
5.1.3) Considerations on Datastore on CA infrastructure for Regular Certificates	65
5.1.4) Considerations on Regular Certificate file structure	67
5.1.5) Location and data of a Certificate in the Certificate Merkle tree	68
5.1.6) ConsensusPKI protocols in TCP stack	69
5.1.7) An Example Execution Flow of PKQuery	70
5.1.8) Block Header Calculation and Consistency Check on Blockchain Blocks	71
5.1.8.1) Block level validation	71
5.1.8.1) Cross-validation among the blocks	73
5.1.9) Example PoW algorithm to calculate a specific blockheader	73

5.2) Considerations on the Blockchain Network	74
5.2.1) Initialization of the Blockchain Network	74
5.2.2) CA infrastructure	74
5.2.2.1) Storage of collected evidence	76
5.2.3) Bandwidth requirements	76
5.3) Prototype for PKQuery protocol of the ConsensusPKI	77
5.3.1) Valid ConsensusPKI certificate generator	80
5.3.2) The CA Server Application for the PKQuery protocol	83
5.3.3) PKQuery Client Application	84
5.3.4) PKQuery simulation results	86
5.4) Formal validation of PKQuery protocol using Tamarin prover	89
Chapter 6) Evaluation	96
6.1) Reflections on the ConsensusPKI	96
6.2) The impact of the ConsensusPKI on the Cyberspace	98
6.3) The shortcomings of the ConsensusPKI	99
Chapter 7) Conclusion	100
Chapter 8) Further study	101
List of Figures	102
Bibliography	104
Appendix A: An example output for the simulation runs	108
Appendix B: Data model overview for the CA Certificates	111
Appendix C: Data model overview for the Regular Certificates	112

Glossary

Term	Explanation
<i>Automatic Certificate Management Environment (ACME)</i>	Peace of the software that manages automatic issuance of X.509 certificates
<i>Automatic Public Key Management Environment (APKME)</i>	An extended and updated version of ACME for the ConsensusPKI ecosystem
<i>Block header</i>	Hash fingerprint of one block of a blockchain
<i>Byzantine Generals' Problem (BGP)</i>	A consensus problem formulated by L. Lamport et al. in 1982. It is used in fault tolerance and consensus mechanisms
<i>Certificate</i>	A digital document that binds a subject to a public key
<i>Certificate authority (CA)</i>	An entity that can issue a certificate after identifying and verifying the subject
<i>Certificate Revocation Lists (CRL)</i>	The public list that keeps records of the revoked certificate (compromised or lost)
<i>Certificate Signing Request (CSR)</i>	A document must be sent to a certificate authority to initiate a certificate issuance process
<i>Certificate Transparency (CT)</i>	A public database system that keeps records of the issued certificates
<i>Certificate Verification Request (CVR)</i>	A document must be sent to a certificate authority to initiate a certificate issuance process in the ConsensusPKI ecosystem
<i>Certification path</i>	A valid trust chain in X.509 PKI ecosystem
<i>Common Alternative Name (CAN)</i>	The alternative name(s) of the subject that holds the certificate
<i>Common Name (CN)</i>	The name of the subject that holds the certificate
<i>Consensus algorithm</i>	A decision algorithm based on Byzantine Generals' Problem
<i>Cyberspace</i>	The interconnected information network consists of computers, physical automation systems, and humans. See section 6.2 on page 98
<i>Digital Signature Schemes (DSS)</i>	A cryptographic protocol based on asymmetric encryption that helps to verify and check the authenticity of messages
<i>Distinguished Name (DN)</i>	See Common Name
<i>Domain Name System (DNS)</i>	Public database service that maps an IP address to subject domain names
<i>DNS Query</i>	An algorithm helps the client to retrieve the IP address of a domain name
<i>Domain Key</i>	Self-declared public key on the DNS system using the DNS records of a domain
<i>Domain Validation (DV)</i>	A protocol that CAs must execute to validate the control of the certificate issuance requestor on the subject
<i>Extended Validation (EV)</i>	An additional validation protocol that CAs execute on top of domain validation to issue a special certificate interpreted differently (with a green address bar) by the browsers.
<i>Fingerprint</i>	A hash value of a data
<i>FIPS</i>	Federal Information Processing Standards. A US standard

	to securely process data
<i>HTTP Public Key Pinning (HPKP)</i>	A client-side experimental protocol that binds certificates to a subject domain
<i>HTTPS</i>	Hypertext Transfer Protocol Secure. A secure communication protocol for the world wide web
<i>Intermediary Certificate Authority (Intermediary CA)</i>	A certificate authority which is explicitly trusted by a root certificate authority, which is also implicitly trusted by all end-user systems
<i>Merkle hash trees</i>	A cryptographic data structure that helps to prove the existence of a data in a dataset without revealing the data itself
<i>Memory pool</i>	The temporary data store of a CA in the ConsensusPKI that holds records of verified or non-verified CVRs
<i>Merkle root hash</i>	The final fingerprint of the data in a Merkle hash tree
<i>Merkle tree</i>	See Merkle hash tree
<i>Man in the Middle (MITM) attack</i>	An attack where the adversary sits between two communicating systems. The adversary can intercept, create, drop and manipulate the messages between the parties
<i>NIST</i>	National Institute of Standards and Technology. An agency of a US government that defines standards for US government and non-government organizations
<i>Online Certificate Status Protocol (OCSP)</i>	A protocol to check the revocation status of a certificate in the X.509 PKI ecosystem
<i>Path Validation</i>	A mechanism to verify a certificate signed by a trusted root or intermediary CA
<i>Practical Byzantine Fault Tolerance (PBFT)</i>	An algorithm to detect faulty messages that can also be used to reach a consensus
<i>Proof of Work (PoW)</i>	A consensus algorithm based on brute force. In ConsensusPKI the PoW is to find a particular hash value
<i>Public key</i>	The public part of an asymmetric encryption key pair. The public key can encrypt messages or verify signatures in asymmetric encryption
<i>Public key infrastructure (PKI)</i>	An ecosystem to verify the identity of the public key of a subject
<i>Secure Socket Layer (SSL)</i>	A deprecated protocol suit to secure communication confidentiality and integrity in computer networks
<i>Signature</i>	See digital signature schemes
<i>Time to Live (TTL)</i>	A mechanism to define lifespan of a message or data
<i>Top-Level Domain (TLD)</i>	A TLD is a domain name that sits at the top of the Domain Name System hierarchy of the Internet, such as .com or .nl
<i>Transport Layer Security (TLS)</i>	TLS is the successor of the SSL. A protocol suite to secure communication confidentiality and integrity in computer networks
<i>Trust store</i>	The root certificate store at the end user systems
<i>X.509 scheme</i>	The standards of X.509 as a whole
<i>Zero Knowledge Proof (ZKP)</i>	An interactive proof system that helps a subject to prove that he/she possesses a valuable something without revealing the valuable itself

Chapter 1) Introduction

The authenticity of the public key against active adversaries has always been an issue for the public key cryptography. The X.509 scheme defines the standards for the public key certificates and gives procedures to process the certification paths [1]. A certificate acts as an identity, and the certificate authorities (CA) are the verifiers of those identities. Once a CA completes the verification of identity, the CA signs the certificate using its private key. A signed certificate then can be used by the service provider to identify the public key used by the service. A crucial component of the ecosystem is the trust store on the end user systems. The trust store holds the public keys of the certificate authorities that the end user system trusts. It is the task of the end user system to process the certification path of a received certificate to check whether an earlier trusted certificate authority signs the received certificate. Additionally, The X.509 scheme allows the creation of a trust chain, where a root certificate authority can trust an intermediary. Furthermore, a trusted intermediary can also act as a certificate authority. In the modern internet era, the role of the certificate authorities is crucial since the trust model relies on the correct execution of the certificate issuance process. A chain of trust created by the certificate authorities is the backbone of the confidentiality and integrity services used by the applications in the cyberspace. However, the trust chain between the operating system (OS), browser vendors and CA's is travail, and it is as strong as the weakest link in the chain. In the last decennia, attackers successfully gained control of many CA infrastructures and successfully created fake certificates for a large number of websites to be used in man in the middle (MITM) attacks. As an example, the DigiNotar hack demonstrated the weakness of the system where hackers successfully created many certificates to sniff internet traffic for services such as Gmail [2], while the identity of the legitimate Dutch government websites was doubtful for an extended period. The compromised DigiNotar root certificate stayed in the trust store for a long time due to the high impact of the trust revocation on many Dutch services [3].

The trust chain ecosystem gives power to the vendors of operating systems or browsers to decide to whom to trust by default. Once someone purchases a computer or phone, the trusted root certificates are pre-installed on the device. However, those root certificates can be updated with a regular update or installed later by another trusted software, making the trust chain challenging to manage. In practice, no end-user checks the certificates of the websites whether the certificate is a forged one or not. The end users rely on the automatic checks that are executed by the procedures defined by the X.509 scheme. Trustturk, Comodo, and Diginotar incidents demonstrated that the X.509 scheme has minimal capability detecting compromised or lousy behaving certificate authorities [4].

The created trust chain in the current PKI ecosystem works under two assumptions:

- It is assumed that no CA in any level of the trust chain, would issue a certificate for a service without adequately identifying the service owner.
- It is assumed that a CA will timely detect and acknowledge when its infrastructure is compromised.

As an industry standard, it is up to a CA to execute related management tasks properly without a single error. The magnitude of an incident on a CA is very high due to the following structural flaws:

- The chain structure in the trust relationship between the CAs
- Lack of a technical boundary to limit CA to sign a certificate request without a request of a service owner.
- Lack of technical limitation to prevent a CA to sign a certificate for any domain

The structural flows exploited in recent incidents. Trustturk assigned an intermediary without adequately identifying the request. Further, Diginotar could not timely detect and act on a hacking incident that the attackers successfully could create a forged certificate for many services across the globe [4]. The security community proposed several improvements to solve the structural flaws caused by the assumptions mentioned above [5].

HTTP Public Key Pinning (HPKP) is a security standard adds new HTTP headers that can be used to declare expected pins (Hashes) of a certificate that a server can use [6]. It relies on the user's browser cache to hold the pin of a website for some time (max-age (TTL) duration) and detect later if there is an inconsistency. HPKP suffers two issues: first of all, when there is no pin for a website in the cache of the browser of the client, HPKP does not solve active adversary attacks, since the attacker can manipulate the headers that the server is sending to the client. Second, due to the max-age parameter acting as a TTL duration, if a website administrator does not update the certificate of its service with care, returning clients might receive false warnings from their browsers.

Certificate Transparency (CT) is an experimental standard that creates append only audit logs for HTTPS certificates on third-party custodian servers [7]. CT uses Markle hash trees for consistency proofs and audit proofs. Using Markle hash trees makes it possible to add all certificates to the audit logs consistently while giving the ability to check whether or not a certificate is added to the hash tree. Both mentioned functionally are convenient tools to maintain the integrity of the log server. On the other hand, CT log server is an append-only log server which is prone to scalability and sizing issues. Another issue is that the CT does not have an automatic detection mechanism of vogue certificates. It is the task of the service owners to check the log servers to see whether a certificate is issued for their services without their request.

DNS Certification Authority Authorization (DNS CAA) [8] is an alternative to HPKP, where CA's are obligated to check wheatear or not a domain holds CAA DNA records. Using CAA records, service owner identifies which CA can issue a certificate for their domain. Since CAA relies on a single, non-mandatory DNS check, it only reduces the likelihood of creation of a rogue certificate. Furthermore, the DNS system itself is not safe proof, and MITM attacks are possible to mislead the CA infrastructure during certificate issuance.

In practice, the CT and the DNS CAA are used together to improve the certainty of public keys of a service. However, the implementations are far from satisfactory as there are no mechanisms to prevent a rogue certificate to be issued and to detect timely when a rogue certificate is being used to mitigate MITM attacks against services on cyberspace.

1.1) Secure communication on the Internet, PKI and X.509 scheme

The Internet is a non-trusted environment. When a standard TCP message travels from point A to point B, it passes many locations where the communication can be attacked. The attacks on the internet can be grouped under a) passive attacks and b) active attacks [9]. In the passive attack scenarios, the attacker tries to gain information about the message in the transmit by sniffing or eavesdropping. In the active attack scenarios, the attacker does not only try to gain information about the message but tries to update the message without the communicating parties recognize it. The passive attacks endanger the confidentiality of communication, while the active attacks target the confidentiality and the integrity of a communication. The confidentiality and integrity are the two pillars of secure communication. The processes of a standard, the X.509 scheme the Public Key Infrastructure (PKI), helps end users to achieve a secure communication on the internet. The PKI uses cryptographic functions to ensure secure communication on the internet. In the center of PKI sit the certificates. A certificate is a document containing the public key of a system being certified. A certificate holds a number of attributes about the system, and it requires to be digitally signed by a CA to be trusted. The attributes in a certificate hold information to identify different aspects of the service that holds the private key of the certificate. The most important attributes of a certificate are the following:

- A subject, it is typically the domain name which is used to access the service
- The validity duration of the certificate (valid from and valid to fields)
- Uniform Resource Locator (URL) where the revocation information for the certificate would be located
- An identifier of the issuing CA to validate the relationship to the certificate

The X.509 PKI is built on a trust model that lies on the following five pillars [5]:

Pillar 1) Certification

Pillar 2) Anchoring trust

Pillar 3) Transitivity of trust

Pillar 4) Maintenance of trust

Pillar 5) Indication and interpretation of trust.

It is implicitly expected that all parties that are responsible for one or more pillar would accurately execute their duties for excellent functioning PKI.

There are two types of the certification process that a CA can offer. First one is the domain validation (DV). DV certificate would be issued once the requestor can demonstrate that he/she can control the domain name. The second validation process called extended validation (EV). The EV certification process executes not only the domain validation but additionally; it identifies the real entity controls the domain name. DV and EV certificates are cryptographically the same, but EV certificates affect the browser behavior by showing the full name of the entity in the browser address bar for users easier to indicate and interpret that they communicate with the correct counterparty.

Anchoring trust is executed either by the software vendors of browsers and operating systems or the end users themselves. It is a very crucial process that defines trusted root certificates. The joint community group called CA/Browser forum brings the parties together that collectively defines guidelines, however different software vendors can require additional measures from CAs to satisfy to be anchored as trusted or to maintain EV indication [10]. Anchoring trust by the end users is executed either by a software such as antivirus software or network administrators in the corporate network. Although, there is no technical boundary or separation whether the trust is anchored by the end user or by the software vendor; a typical application of the trust anchored by an OS or browser vendor is the public key identification of the websites. The root certificates that is anchored by the end users are used in general for antivirus scanning, or the public key identification of the intranet deployed web application within a corporate network.

Transitivity of trust is executed by the CAs to trust another institution to act as a CA [5]. Trusted CA certificate becomes intermediary CA certificate and can be constrained by the “CA” Boolean parameter and “pathlen” constraint fields in the certificate file. Both parameters are input variables of the X.509 path validation mechanism [1], and it defines whether or not the intermediary CA trusted through transitivity can trust another CA to act as an independent CA. The intermediate CAs are invisible for the end users until the end user visits a website that identifies itself with a certificate signed by an intermediary CA.

Maintenance of trust is executed by the issuing CA of a certificate when the concerning certificate is a domain certificate or an intermediary certificate [1]. The software vendors also execute maintenance of trust if the concerning trust is anchored [4]. Software vendors can revoke an anchored trust by publishing a software update, and the trust will only be revoked if the end user deploys the update. CAs have the two options to execute revocation tasks. The first option is the certificate revocation lists (CRL) which must be accessible through a URL included in the certificate. The second option is the online certificate status protocol (OCSP). Both of the options need to be invoked by the certificate owner whom the certificate is lost or compromised. Once the certificate owner requests the certificate to be revoked, the CA must put the revocation information into the CRL or OCSP server, depending on the information included in the certificate.

The indication and interpretation process is the most crucial process and happens on the end user system to validate the trust path and to interpret the trust for the end user. This process decides whether the connection to the counterparty is secure. In the X.509 scheme, path validation mechanism defines the algorithm and sets the rules for the end user system vendors to apply [1]. During this process, the end user interacts with its client software. Client software such as browsers gives cues to the end user for the end user to decide whether the connection is secure or not. The most important cue is the browser warning stating that “the connection is not secure.” This warning is given in the following cases:

- When the certificate received from the server cannot be validated through the path validation
- When the validity of certificate received from the server does not fall in the system time of the end user
- When HPKP cache on end-user system hold records other than pinned keys on DNS registry of the service (HPKP is not a mandatory deployment) [6]

The other cues given to end users varies per vendor to help the end user to consider the connection to accept as secure. A lock item, green address bar, entity name on address bar are examples of those cues given to the end users by the browsers.

1.2) Structural flaws on the trust model of the X.509 scheme

Massive deployment of the internet brought many opportunities and removed boundaries for our daily lives. The end-users are unaware of where the traffic passes through when they visit a web page. The number of hops that a TCP package would pass depends on the physical distance of the communicating parties. Every hop that a package passes is a place that can be attacked. The complexity of the massive connection network makes the internet a hostile environment. End users, in general, do not know who their counterparty is. The PKI is the essential protection measure that the communicating parties have. The PKI is built on a trust chain that supposed to work in a hostile environment to bring security to open communication by applying cryptographical measures. Unfortunately, even perfectly functioning PKI has structural flaws that bring the confidentiality and integrity of secure communication in danger [5]. On every one of the five pillars that PKI lies on there exists structural flaws.

A typical certification process starts with the creation of a certificate signing request (CSR) by the service owner that can demonstrate that he/she controls the subject domain name. When a CA processes a DV certification request, CA sends an email to one of the predefined email addresses such as ssladmin@domain or admin@domain or administrative contact email in the DNS records that domain owner supposed to control [5]. In the email message CA place a link to a specific URL bound to the CSR where the receiver of the email message must click and execute a specific task that would demonstrate the requester controls the domain. This email challenge-response mechanism suffers the following issues:

- Flaw 1: Email messaging is not a secure protocol, and email messages can be intercepted by attackers [11]
- Flaw 2: DNS also suffers many attacks that CA can be tricked to send an email to an email address that has no relation to the requesting domain [11]

Flaw 3: The X.509 standards allow issuance of multiple certificates, by using different key pairs, at the same time for a specific domain name identified in the common name (CN) field in the certificate files. The flexibility of having multiple valid certificates for the same subject at the same time makes it also possible to issue fake valid certificates without knowledge of the subject. Detection of such fake certificates is challenging since no one knows the existence of the fake certificate.

Trust anchoring and trust transitivity are very flexible processes that stimulate easy executions of many transactions. The flexible structure is created by the following lea ways that the X.509 scheme supports [5]:

- Flaw 4: Any trusted CA is allowed issue a certificate for any domain name, as no technical boundary limits CAs to operate across the globe
- Flaw 5: An anchored CA can issue intermediary certificates that transfer the trust to another party
- Flaw 6: Intermediary CA will be unknown until a client receives a certificate issued by the intermediary CA

- Flaw 7: CA does not need to transparently execute the management processes around certificate issuance, both for end-user certificates and CA certificates
- Flaw 8: Trust revocation from a certificate in the trust chain invalidates all certificates under the revoked certificate including the certificates that are not compromised
- Flaw 9: A software update or manual intervention is required to execute the trust revocation from an anchored root certificate
- Flaw 10: Anchoring trust can be executed by end-user systems [12]

The flexibility created by X.509 scheme assumes all parties will be equally committed to the correct functioning of the ecosystems. However, in reality, the ecosystem exploited many times by attackers. The flexibility created by the X.509 functions as multiple structural flaws creating uncertainty about the public keys used by the services in the cyberspace. Additionally, there is no outbound mechanism for end users to verify whether the certificate received by their end is the certificate that the counterparty system uses.

The Certificate transparency (CT) project, which is now supported by many CAs, aims to monitor the CAs using log servers. CT adds the log server maintainers into the PKI ecosystem to monitor the trusted CAs. However, the log server maintainers are another trusted party with limited detection only capabilities.

Figure 1 indicates the relationship between the structural flaws earlier indicated as Flaw 1,..., Flaw 10 and the X.509 pillars introduced in the previous section on page 10.

	<i>Pillar 1</i>	<i>Pillar 2</i>	<i>Pillar 3</i>	<i>Pillar 4</i>	<i>Pillar 5</i>
<i>Flaw 1</i>	X				
<i>Flaw 2</i>	X				
<i>Flaw 3</i>	X	X	X	X	X
<i>Flaw 4</i>	X	X	X	X	X
<i>Flaw 5</i>	X	X	X	X	X
<i>Flaw 6</i>			X	X	X
<i>Flaw 7</i>	X	X	X	X	
<i>Flaw 8</i>		X	X	X	
<i>Flaw 9</i>		X	X	X	X
<i>Flaw 10</i>		X		X	X

Figure 1: Relationship between the structural flaws and the pillars of the X.509 ecosystem

Figure 2 summarizes the structural flaws of the X.509 scheme as we use it on the internet.

Process	Problems on initial phase	Problems on maintenance
<i>Certification</i>	<ul style="list-style-type: none"> • Identification of a subject is a single step and weak. Demonstration of control over the subject (Domain) relies on a non-secure infrastructure • It is possible to issue multiple certificates for the same subject by multiple CAs • Certificate issuance is not transparent. CAs does not need to report issued certificates • CAs does not need to report how they identified the subject. No transparency over the identification process 	<ul style="list-style-type: none"> • Vogue certificate issuance is possible. Lack of automatic process to detect rogue certificates • CT is not mandatory and has limitations • CRL and OCSP has limitations • Revocation of intermediary certificates cause invalidation of all certificates under the intermediary certificate due to trust chain
<i>Anchoring Trust</i>	<ul style="list-style-type: none"> • There is no transparency over root certificates, and how they are placed in the certificate store of end-user systems • There is no boundary for the activities of the root CAs • It is possible to add root certificate manually or with a software update or by administrators 	<ul style="list-style-type: none"> • Revocation of root certificates are not possible with CRL or OCSP • Impact of certificate revocation is impossible to measure
<i>Transitivity of Trust</i>	<ul style="list-style-type: none"> • The trust to a CA can be transferred to other parties without limitation • There is no transparency on transferred trust from a CA. The intermediary CAs are invisible until the end user system receives a certificate issued by the intermediary CA 	<ul style="list-style-type: none"> • Impact of certificate revocation is impossible to measure
<i>Maintenance of Trust</i>	<ul style="list-style-type: none"> • CRLs are not reflected immediately and requires the end-user system to update its records • Certificate revocation is not automatic and requires acknowledgment by the subject (certificate owner) 	
<i>Indication and Interpretation</i>	<ul style="list-style-type: none"> • There is no distinction between internet, intranet, and local applications. The end user systems always behave the same, and It is possible to accept and use invalid certificates for internet domains 	<ul style="list-style-type: none"> • Lack of distinction influences user behavior to accept invalid certificates on the internet, creating overall insecure user behavior

Figure 2: Summary of structural flaws of X.509 ecosystem

1.3) Cryptographic functions and Signature Schemes

The X.509 scheme uses cryptographic primitives as building blocks within the processes of the PKI ecosystem. A typical X.509 certificate contains a fingerprint of the signature of the issuing CA. A digital signature in PKI ecosystem is usually a large integer value. For the verification process, the verifying party needs to know which algorithm is used when the signature is created. Due to efficiency reasons, a digital certificate contains only the fingerprint of the signature. That is why the verifying party needs to know which function is used when the fingerprint is generated. Signature algorithm field in a certificate file defines the signature algorithm to be used in the verification process, such as RSA. Signature hash function field in a certificate file contains the information of the hash function to be used in the verification process to calculate the fingerprint of the signature [1]. The signature is only useful to the client who receives the certificate if he/she has a valid trust chain in the trust store of his/her device that verifies the signature [13].

1.4) Emergence of Blockchain technology as a decentralized trust

In 2008 the bitcoin was introduced as a peer to peer electronic cash system [14]. The paper described an advantageous structure to maintain the integrity of the system. The structure later named blockchain. A blockchain is an append-only public ledger, organized in blocks maintained in a chain form that are linked to each other using hash functions. In the digital financial world, clearing houses are trusted third parties that keep track of the money moving around the world. Since the electronic money is kept on electronic mediums, a core task of a clearing house is to keep the balances of each account to avoid double spending. A blockchain network replaces the function of a clearinghouse on electronic money by using a blockchain formed public ledger and a consensus mechanism among the nodes of a blockchain network. The blockchain is replicated among all of the nodes, and each transaction is broadcasted to all nodes and recorded in the blockchain. In blockchain, the transactions are confirmed by consecutive blocks, and if two blocks are added at approximately the same time, a consensus algorithm decides which block to add to the chain. There are various consensus algorithms such as Proof of Work (PoW) or the practical Byzantine fault tolerance (PBFT) algorithm [15] to the Byzantine Generals' Problem to solve the conflict [16].

1.5) Research Question

Section 1.2) Structural flaws on the trust model of the X.509 scheme explains the issues around PKI, and experimental deployment of certificate log servers to solve some of the structural flaws of the PKI ecosystem. The blockchain technology, however, replaces the single trust with an ecosystem that works without trusted parties. Instead of adding new trusted parties with different tasks into the ecosystem, this research aims to take a different path to remove the trusted third parties, by replacing the tasks of CAs, and by creating a tamper-proof environment backed by blockchain as a decentralized trust, so that the structural flaws of the current PKI ecosystem would be remedied. Hence, the research must answer the following primary research question at the end of this research:

Are blockchains and consensus algorithms suitable as a building block to solve the structural flaws of the X.509 scheme and what would such a public key ecosystem look like?

The problem requires following subproblems to be answered individually:

1. How to construct a scheme to solve the structural flaws of anchoring, transitivity, and maintenance of trust processes of the X.509 scheme using blockchain technology?
2. How to construct a scheme to solve the structural flaws in the certification processes of the X.509 scheme?
3. How to construct a scheme to solve the structural flaws in the indication and interpretation process of the certification path validation algorithm of the X.509 scheme?

1.5.1) Requirements

The requirements can be grouped into two categories. The first category is the requirements due to the structural flows of X.509, and the second category is the requirements due to the current usage standards such as TLS handshake duration. Requirements in the second category are based on the following expectations of the end users:

- Initial connection to a service (a website) should be as quick as today's standards
- The certificate store size on end-user systems should not be exponentially big compared to today's standards

Requirements due to the structural flaws of X.509:

- **Requirement 1:** Transparency on CA processes; all processes of all CA nodes must be transparent to the public, and can be verified using cryptographic functions
- **Requirement 2:** Transparency on the Public keys of the services in the cyberspace; all public keys verified by the certification process must be transparent, publicly available, and can be verified using cryptographic functions
- **Requirement 3:** Temper proof; all information in the public blockchain database should be tamper-proof, an attempt to temper the information must be detected automatically
- **Requirement 4:** Trust revocation from a CA should not have cascading effects and should not cause invalidation of other public keys in the blockchain database
- **Requirement 5:** Temper proof certification process; control over the subject domain names must be demonstrated using cryptographic functions guaranteeing the certainty of the identification
- **Requirement 6:** Higher public key certainty; the certainty of the public key of a service should be more than the current X.509 PKI ecosystem on the internet

Requirements due to current usage standards:

- **Requirement 7:** Scalable; the ecosystem should not suffer from ever growing blockchain size, and the processes should not overload the operations of the CAs
- **Requirement 8:** High performing; the new ecosystem should not create extra latency to services such as HTTPS (SSL or TLS) handshake

1.6) Contribution

To the best of our knowledge, the ConsensusPKI is the first attempt that integrates blockchains and consensus mechanisms to solve all known structural flaws of the X.509 scheme. To achieve the goal, we propose to use a combination of blockchains, consensus algorithms, and cryptographic primitives as a full functioning PKI ecosystem.

1.7) Methodology & Reporting outline

The research aims to create a new artifact to solve the structural flaws of the current X.509 scheme. The new artifact needs to meet the requirements listed in the previous sections. As illustrated in Figure 3, the design process will roughly follow the design cycles of the design science for information systems research as described by Hevner et al. [17]. In the previous sections, the current X.509 standards and its structural flaws, relevance to confidentiality and integrity of communication in information systems are shortly explained, and the requirements to create a new artifact that aims not suffer any of the explained structural flaws are introduced. In chapter 2, we will evaluate the alternative solutions that were proposed by the relevant scientific community in order to solve the structural flaws of the X.509 ecosystem. In chapter 3, we will evaluate the existing knowledge base and identify relevant information system structures that would be used as building blocks of the new PKI ecosystem.

The ConsensusPKI ecosystem will be introduced in chapter 4. Introduction of the new ecosystem will be followed by a detailed explanation of the underlying components of the new ecosystem. Validation of the new model will take place in chapter 5. The validation process will take the following aspects into account a) mathematical correctness, b) implementation aspects from a software engineering perspective, c) practical results of a developed prototype. In chapter 6, the results of the research will be evaluated. In Chapter 7, we revisit the research question and discuss whether the output of the research satisfies the requirements. Finally, in chapter 8, we will provide recommendations for future work.

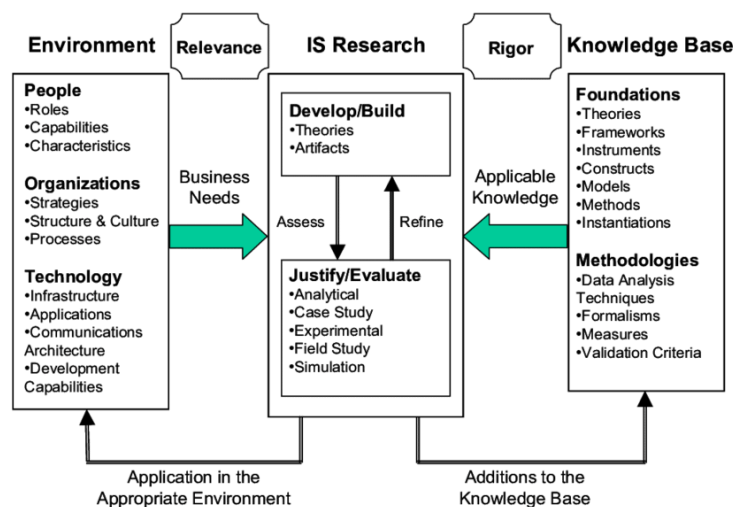


Figure 3: Information system research framework by Hevner et al.

Chapter 2) Prior research and proposed solutions

In last the decennia, the scientific community tried to address the structural flaws of the X.509 ecosystem. Most scientific efforts focus on some of the structural flaws, by extending some functions of the X.509 ecosystem. The emergence of the bitcoin in 2008 as a peer to peer electronic cash system [14] changed this approach, and the scientific community started discovering the possibilities of the core element of the bitcoin, the blockchain technology.

There is an uncommunicated consensus among all of the scientific efforts to use some form of a public log server to solve the structural flaws of the X.509. In the following sections, we will evaluate the scientific efforts under two categories: a) solutions without blockchain support and b) solutions with blockchain support.

2.1) Proposed solutions without blockchain support

Many alternative proposals are addressing the structural flaws of the trust model of the current X.509 PKI ecosystem. The PKI ecosystem has three main stakeholders. The first stakeholder group is the clients, who rely on the trust that is anchored in their devices. The second group is the service owners, thus the domain names, whose public keys needed to be trusted by a certificate authority that has a trust anchor in a device of a client. The third group consists of the certificate authorities. We will analyze the alternative solutions first by the stakeholder model [18], and later by the technology that the solution relies on.

Client focused solutions: Alternative solutions in this category try to solve the flaws of the PKI by improving the indication and interpretation phase of trust on the client side by giving end users a choice or by observing their trust history so that the end users become actively responsible for choosing the trusted party. The policy engine [19] and the SSL Observatory [20] are examples of the client-focused solutions. However, both solutions struggle to reach a big audience because a significant number of people lack the knowledge and tools to decide whom to trust. On the other hand, the dataset of the SSL observatory brought many issues to the attention of the scientific community.

Domain focused solutions: Alternative solutions in this category focus on the identity of the domain and try to protect the identity assuming a CA would be compromised. As discussed in Chapter 1) the HPKP [6] and the CT [7] are examples of this approach. The solutions in this category require domain owners to monitor and control their public keys actively. However, proposed solutions in this category have structural flaws. First, the CT requires site owners to monitor the issued certificates for their domain name manually. Secondly, the HPKP does not protect the visitors on the first visit to a domain.

Certificate authority focused solutions: The solutions in this category put the CA in the center of the (proposed) solution. Certificate Revocation Lists (CRL) and Online Certificate Status Protocol (OCSP) are already in use under current PKI infrastructure. The main problem with both solutions is that they both rely on an adequately functioning CA infrastructure, even when they are compromised. Additionally, both solutions do not solve the cascading effect because of the revoked trust path that occurs when a CA gets compromised.

From a technology point of view, all alternative solutions, that we will discuss, try to increase the transparency by using public log servers. The use of a public log server increases the transparency of the certification process of CAs, by adding all issued certificates to a

publicly available append-only log server. Sovereign Keys (SK) [21] is proposed before CT and is similar to CT. SK proposes to have a sovereign key pair by a domain to authorize a certificate issuance. The main purpose of SK is to detect a forged certificate, rather than to prevent the issuance of it. SK requires clients to trust in an append-only log server. This solution is far from usable since it somehow pushes the client to intervene with the automatic X.509 path processing during the indication and interpretation phase of a certificate. In practice, SK shifts the trust from CAs to the append-only log server. SK lacks cryptographic integrity checks against tampering of log servers. SK also does not provide verifiable proofs whether a certificate is included in the log server.

Certificate Transparency (CT) [7] is proposed by Google and aims to detect vague public key certificates issued by a compromised or inappropriate behaving CA. CT improves the transparency by adding the certificates during issuance to append only log server that is organized in the form of a Markle tree. As discussed in Chapter 1) the Markle trees are then used for consistency proofs and audit proofs. However, CT is not obligatory, and does not cover the full lifecycle of the certificates, lacks automatic detection, and requires site owners to manually check whether there exists a forged certificate for their domain name [5]. Furthermore, like SK, CT shifts the trust again to another party.

Another alternative is the Accountable Key Infrastructure (AKI) [22]. AKI also uses public logs to increase the transparency of a CA. Different than CT, AKI uses a data structure based on lexicographic (~alphabetical) ordering to solve the key revocations that CT misses. Furthermore, AKI uses public log validators and maintainers for CAs to check and monitor the other CAs. By adding validators and maintainers into the ecosystem, the AKI reduces the requirement to trust another party fully. AKI prevents the usage of fake certificates, which is an improvement compared to the detection mechanism in CT. However, AKI again shifts the trust towards newly introduced third parties; the public log maintainers and validators.

Certificate issuance and revocation transparency (CIRT) [23] is a proposal to manage the lifecycle of certificates, including certificate revocation. It somehow combines the CT and the AKI and solves the issues that the CT and the AKI suffer. First, CIRT adds revocation of the certificates in the public log servers, which the CT misses. Secondly, it shifts the monitoring and detection of fake certificates to end user browsers, instead of validators and maintainer in the AKI. Like the CT, the CIRT also only detects the attacks that use fake certificates, but it does not prevent the issuance of fake certificates.

CONIKS [24] is another improvement proposed to increase the transparency of the keys that end users receive on their end. The main aim of the CONIKS is to increase the end users interpretation and detect deviations of the keys they receive from a service. Querying CONIKS based certificate log servers is done while preserving end users privacy. Unfortunately, CONIKS is only an improvement to the CT and is limited to the detection of forged certificates.

The Attack Resilient Public-Key Infrastructure (ARPKI) [18] is an improved version of the AKI. The main difference between the ARPKI and the AKI is that in the ARPKI, a domain owner picks n number of CAs or public log maintainers to participate to certificate issuance. Instead of connecting each of the parties (CA or log maintainer) separately like in the AKI, the ARPKI requires domain owners to contact only one CA to issue a certificate. During the lifecycle of a certificate, the CAs and log maintainers monitor the activity of the other parties. As a result, the ARPKI prevents forged certificate issuance even when the $n - 1$ party

involved is compromised together. As in the AKI, the ARPKI shifts the trusts again to other third parties; the independent log server operators.

The Distributed Transparent Key Infrastructure (DTKI) [25] is another solution based on public log servers and integrates the SK, the CT, and the AKI to manage the certificate lifecycle. In the DTKI certificate issuance requires a domain key pair to be involved in the certificate lifecycle. Usage of public log servers is similar to the CT, and as in the AKI, domain owners need only one CA during certificate issuance. The logs are maintained in Merkle tree for consistency proofs and audit proofs. The DTKI does not require another party to check the integrity of log servers since the task is given to the clients. The main issue in the DTKI is that the synchronization of log servers has delays and are not guaranteed.

2.1.1) Log Servers

As discussed in the previous section, the scientific community relies on log servers to detect or prevent intentional or unintentional misbehavior of a CA. It is apparent that the community does not have consensus on the following essential aspects of implementing public log servers into the PKI:

- Who will operate log servers? (independent or part of CA infrastructure)
- How to guarantee the integrity, and reliability of log servers?
- How to guarantee the synchronization between the log servers?
- How to make sure the interpretation of the logs is adequately implemented on the end user side?

The questions above bring us to the blockchain technology, where data integrity and synchronization is handed over to an untrusted distributed network of computers. In the next section, we will discuss the proposed alternative solutions that use the blockchain network as a replacement for trust.

2.2) Proposed solutions with blockchain support

The emergence of blockchain as a centralized trust became a popular topic in many fields of research including the research for the PKI. Following, we will evaluate the alternative solutions backed by blockchain. The first known proposal for a blockchain backed solution is Certcoin [26]. In Certcoin, two public key pairs are used for certification, for public key query, and revocation of a public key for a given domain name. There are several issues with Certcoin: First of all, in contrast to the current PKI model, there is no identity verification. Secondly, Certcoin does not provide a solution when both keys are lost or compromised. Losing both keys causes a certificate owner not to be able to identify itself and thereby a certificate cannot be revoked or renewed. Further, because Certcoin uses the structure of bitcoin, it is very inefficient. To compensate the inefficiency, Certcoin uses Merkle hash tree accumulators, leading to an ever-growing data structure.

Another alternative is Authcoin [27]. Authcoin shares some principles with Certcoin and adds verification of public keys with a single challenge-response scheme; a replica of ACME discussed in section 3.5) Automatic certificate management environment (ACME). In detail, Authcoin relies on a single verification that depends on a single non-secure DNS query. Furthermore, Authcoin suffers the same issue as Certcoin regarding certificate revocation upon losing the key pairs.

The third alternative is the Instant Karma PKI (IKP) [28]. The focus of the IKP is to create mechanisms to detect and punish CA misbehaviors. To achieve this goal, the IKP creates two new roles within its infrastructure called detectors and IKP authority. Creating incentives for detectors to check for misbehaviors, IKP creates a blockchain based public ledger that also keeps the finances of the CAs. Finally, a CA is punished financially within the mechanism of IKP, upon detection of misbehavior. The detectors receive fees from the financial balance of a CA. There are several issues regarding the IKP. First, the IKP focusses on detection rather than prevention, and in the IKP it is possible to create fake certificates. The IKP shifts the trust to detectors and the IKP authority, rather than creating a trust ecosystem. Furthermore, no mechanism guarantees that a detector reports a found misbehavior.

CeCoin is another proposed alternative [29]. CeCoin adds the PKI into the bitcoin structure in a way that the lifecycle of certificates can be managed. Since CeCoin adds all certificates and their operations into a single blockchain, storage requirement on operating nodes will grow exponentially with every certificate added to the network. Additionally, to create incentives for the node operators in CeCoin, the certificates are decoupled from domain names and are coupled with identities, called tokens. All certificate operations then can be managed using the token associated with the identity. As a result of that, in case a token is lost, the certificate operations are no longer possible for the domain names that are coupled to the lost token.

Certchain has recently proposed another alternative [30]. Certchain improves the current PKI infrastructure by improving the log server infrastructure that CT uses. Certchain adds the certification and revocation logs of certificate operations of all CAs in a blockchain distributed among all CAs. Certchain changes the CT log servers into blockchain nodes, called bookkeepers, to guarantee the integrity and availability of logs. Certchain does not try to prevent the issuance of fake certificates, but it improves the detection mechanisms. Detection task is given to the identity owners themselves or the end users.

Distributed X.509 PKI is another proposed alternative [31] and is similar to CeCoin. Opposite to CeCoin, it strictly couples the identities to domain names and the DNS system. Identity verification during certificate issuance is a single verification, and thereby prone to DNS attacks. Furthermore, strict coupling creates new attack possibilities. For example, decoupling an identity from a domain name causes all certificates of the domain to be revoked. Given that the DNS is a non-secure structure, this situation can be abused by the attackers.

2.3) Comparison of Alternative Solutions

In this section, we will assess the alternative solutions against the requirements of the thesis. Figure 4 illustrates the comparison of the alternative solutions against the requirements indicated as R1,..., R8. The ARPKI is the only solution where trust revocation from a single CA does not have a chain effect on all certificates in the trust path. However, in the ARPKI the revocation of trust from multiple CAs and log servers will still impact many certificates, even though they are not compromised. As a verification mechanism, all alternative solution uses a version of the signature scheme on the certificates.

Alternative Solution	R1	R2	R3	R4	R5	R6	R7	R8
<i>Policy Engine</i>	No	No	No	No	No	Yes	No	No
<i>SSL Observatory</i>	No	No	No	No	No	Yes	No	No
<i>HPKP</i>	No	No	No	No	No	Yes	Yes	Yes
<i>CT</i>	Partial	Partial	Yes	No	No	Yes	No	No
<i>CRL</i>	No	No	No	No	No	Yes	Yes	Yes
<i>OCSP</i>	No	No	No	No	No	Yes	Yes	Yes
<i>SK</i>	No	No	No	No	No	Yes	No	No
<i>AKI</i>	Partial	Partial	Yes	No	Partial	Yes	Yes	Yes
<i>CIRT</i>	Partial	Partial	Yes	No	No	Yes	No	No
<i>CONIKS</i>	No	No	No	No	No	Yes	No	Yes
<i>ARPKI</i>	Yes	No	Yes	Partial	No	Yes	No	Yes
<i>DTKI</i>	Yes	Yes	No	No	Partial	Yes	No	Yes
<i>Certcoin</i>	No	Yes	Yes	No	No	No	No	No
<i>Authcoin</i>	Yes	Yes	Yes	No	No	Yes	No	No
<i>IKP</i>	Yes	Yes	Yes	No	No	Partial	No	No
<i>CeCoin</i>	Yes	Yes	Yes	No	No	Yes	No	No
<i>Certchain</i>	Yes	Yes	Yes	No	No	Yes	No	No
<i>Distributed X.509 PKI</i>	Yes	Yes	Yes	No	No	Yes	No	No

Figure 4: Comparison of alternative solutions against the requirements of the thesis

Chapter 3) Background

This chapter gives a theoretical foundation for the topics that were shortly introduced in the introduction chapter. The information in this chapter will be used in the rest of the research. Topics are categorized based on their functions and include cryptographic functions, the domain name system (DNS), the current PKI ecosystem, and the blockchain technology.

3.1) Cryptographic functions

3.1.1) Hash functions

Hash functions are essential cryptographic primitive, and they are widely used in cryptographic protocols including X.509. Hash functions compute a fingerprint of a message as a fixed-length string. The output of a hash function, the fingerprint, is a unique representation of a message. A fundamental security feature of a hash function is that hash functions do not have a key and they are one-way functions [13]. Hash functions in PKI ecosystem mainly used for efficiency. For example, a certificate file does not need to contain a full signature of the issuing CA, since the fingerprint of the signature will represent the signature. For 2048 bits RSA signature, the SHA256 fingerprint will save roughly (depending on the protocol fields and encoding) 1792 bits traffic for every HTTPS handshake that a web server executes with a client.

Three properties measure the security of a hash function; a secure hash function needs to satisfy all the properties [13]. First of all, the preimage resistance, the one-way-needs. It is expected that from a hash output value, it is computationally not possible to retrieve the input value of the function. Rainbow attacks are not considered as a weakness, since hash functions do not have a key, and same input value will always give the same output value to a hash function. The second important property of a hash function is second preimage resistance, weak collision resistance. The third important property of a hash function is strong collision resistance. Collision resistance in general means that a hash function does not generate the same output value from two different input values. Difference between weak collision resistance and strong collision resistance is in the weak collision, the attacker knows the input and the output value of a hash function and tries to find a different input value that would give same output value that he/she knows. The strong collision resistance is harder to achieve since the attacker tries to find two input values that would give the same output.

There are many theoretically or practically broken hash functions [32]. The hash functions that we use today could be broken in the future. The new PKI ecosystem needs to take into account to support multiple hash functions in a parametric way so that users can decide depending on their requirements which hash function to use.

3.1.2) Cryptographic Signatures

In the real world, a standard document can be signed to certify a consent or an agreement written on the document. By signing, the signer accepts the responsibility for the content of the document. The signature is always physically a part of the document, without a signature, the document is not accepted as binding. Although forgery is possible in the real world, it is often difficult to replicate a signature of someone else. In the digital world, signing a document is a lot more complicated than in the real world, since it is very convenient to cut and paste a signature, in a digital medium. Cryptographic signatures

provide solutions to the problem [13]. A cryptographic signature scheme requires to have the following properties to mitigate forgery attacks:

- Integrity: the content of the signed document cannot change, after it is signed
- Nonrepudiation: The signer of the document cannot deny that he/she signed the document
- Authenticity: the signature on the document can be verified that it belongs to signing the party

A Digital Signature Scheme has two components, a signing algorithm that enables a user to sign a message with his/her private key, and a verification algorithm that enables anyone who has the public key of the signer to verify that the signature is authentic. However, the above two components do not guarantee the authenticity of a signature. The authenticity of a signature can be guaranteed with a trusted third party, where both signer and verifier trusts. The current X.509 ecosystem is built on cryptographic signature schemes, and it is widely used all over the globe. There are many digital signature schemes (DSS) meets the conditions mentioned above. The digital signature algorithm (DSA) from NIST FIPS 186 [33] is an example of such a signature scheme. In 2010, it was discovered that repeating the secret number in the algorithm that supposed to be used once in a series of message signing, results in a private key compromise in DSA [13]. A variant with deterministic (EC)DSA proposed in RFC 6979 to solve the issue [34]. The incident demonstrated that a signing scheme that we use today could be broken in the future. The new PKI ecosystem needs to take into account to support multiple signing schemes in a parametric way so that users can decide depending on their requirements which signing scheme to use. There are many variants of signature schemes such as threshold signature schemes, blind signature schemes, multi-signature schemes. While explaining the new ecosystem, we will revisit signature schemes since it is one of the core pillars that the new ecosystem relies on.

3.1.3) Cryptographic Puzzles & Zero-Knowledge Proofs

Cryptographic puzzles are widely used algorithms in various applications in the cyberspace. The variations are enormous, a protection mechanism against denial of service attacks is a good example of a client puzzles [35]. A daily life example is a challenge-response mechanism that many online banking systems use during login stage or transaction approve stage of their clients. The idea behind a challenge-response puzzle is that the responder is someone capable of executing a task because he/she possesses a secret that is only known to an authorized responder. By solving the puzzle, the responder identifies itself as the controller of the secret that can solve the puzzle. Another example is the proof of work puzzle in the bitcoin network. Bitcoin miners need to find a special hash that starts with a number of 0 “Zero” characters that the bitcoin network at the computation moment requires for a block to be added to bitcoin blockchain [14]. By finding the particular hash output, the miner demonstrates that he/she worked for the verification of the transactions in the block, and the miner is rewarded with a transaction fee. An interesting property of this approach is only the first miner that finds the hash is rewarded since a transaction can be added to blockchain only once.

Cryptographic puzzles can be used as a single puzzle or in a chained form so that the final result can be correct only if the puzzles are solved in the correct order [36]. If the puzzles in the chain are not solved in the correct order, the output will be wrong.

Zero-knowledge proofs (ZKP) [37] schemes is another approach proving an identity without revealing it. An example application is solving a puzzle without disclosing the solution. A ZKP scheme can be built in many ways; for example, an interactive ZKP requires interaction between two parties, usually a client and a server, for the client to prove that he/she is capable of something since it possesses a secret without revealing the secret. The tricky part is whether the server will accept a single proof or will require the client to demonstrate that he/she is capable of solving different puzzles multiple times. Consecutively solving different puzzles multiple times is essential since the probability of someone solving a puzzle is %50 (correct or wrong). If the challenge is repeated only once, the probability will be 75% that the claimer (client) is capable of solving if both of the time he/she solves the puzzle. Repeating the challenge ten times, given that the answer is correct each time, the probability of the client is capable of solving the puzzle is more than %99.9. Following is the formula to calculate the probability of identity after a proof test is executed n times:

$$\text{Probability of correct identity} = 1 - 0.5^n$$

Figure 5 illustrates the probability distribution of proof of identity versus a number of tests. If a test is executed only once, the chance that the solver solves the puzzle by chance is fifty percent. When the test is repeated twenty times, the probability of someone to solve all twenty puzzles consecutively by luck reduces to one in a million. Challenging many times with different puzzles is a convenient approach, and we will use this property and introduce a different application of an interactive ZKP system with one client and multiple servers creating a consensus to verifying the identity of a client. The idea behind this new system is the n number of systems without knowing each other will come into consensus, because they **independently** challenged the client, and the client successfully solved all challenged puzzles consecutively.

Number of tests	Probability
1	1-(½)
5	~1-(3/100)
10	~1-(98/100000)
15	~1-(3/100000)
20	~1-(1/1000000)
30	~1-(1/1000000000)

Figure 5: Probability distribution of a ZKP system

3.2) Domain Name System (DNS)

A modern computer communication network works using internet protocol (IP) addresses to exchange information between hosts. However, IP addresses being numbers are difficult for humans to remember, and using only IP addresses would limit the number of applications that can run simultaneously on a server. In the early days of the Arpanet, each host needed to hold a copy of HOSTS.TXT file that mapped the name and IP address of every host in the ARPANET. The mapping structure was from name to address called as a forward mapping, and from address to name, also called as a reverse mapping. The maintenance and the distribution of the HOSTS.TXT file became an issue with the rapid growth of the ARPANET network [38]. DNS was proposed as an answer to the problems around the name resolution using the HOSTS.TXT file and later became one of the core standards of the internet [39]. Similar to HOSTS.TXT file DNS is also designed to answer forward and reverse lookup of addresses.

DNS works as a client-server model where a client can query the DNS server to resolve the IP address of a host and vice versa. A host in a TCP/IP network can have multiple domain names and IP address at the same time. The domain names to IP address mappings are kept in the DNS database, and domain names can be used in all applications or commands instead of using the IP address. The DNS database is a distributed database and replicated to all DNS servers worldwide. The DNS database contains resource records (RR), and for every domain name, there can be one or multiple resource records [39].

The DNS protocol is an application-layer protocol sits on top of the transport layer that carries out the packet transfer of packages. DNS protocol uses both UDP and TCP. DNS queries are executed in a connectionless manner and through the UDP port 53. Database replication, also called zone transfer, between primary and other DNS servers uses TCP port 53 next to UDP port 53. The DNS queries or zone transfers are carried on a non-secure channel that creates many attack scenarios for malicious users actively or passively can abuse.

DNS system is a hierarchical system, and it is also replicated how a domain name is structured. A domain name is a hierarchical string that consists of several strings separated by dots. The name is processed from left to right, and the outmost right string after the dot is called top-level domain (TLD). The “com” or “nl” can be given as an example to a TLD. In the DNS hierarchy, the top domain is called the root domain represented by a dot (.) on the very right of the domain name string, and almost all applications leave out the root dot. Although, technically there is no difference, there are two kinds of TLD. First one is generic TLD such as “com.” The second one is country code TLD or gTLD such as “nl” [39]. The structure of a domain name hierarchy can be the following string: “subdomain.domain.tld.”. Although hypothetically, there is no hierarchical constraint in the subdomains, most users use only one or two hierarchical subdomains. If we take “www.google.com.” as an example, the com is the TLD, google is the domain, and www would be the subdomain. A domain name string, TLD, and subdomain strings can be a maximum of 63 characters due to the limitation of the DNS standard [40].

After the “.” the root domain, the registration of the domain names for a specific TLD are delegated to other organizations. As an example, in the Netherlands SIDN is the authority that registers the domain names for “nl” TLD. When a domain name is registered, the registration authority keeps the records of the authorized DNS servers, called Name Servers (NS), for the registered domain. The RR information in the name servers of the domain is the authoritative records and is replicated to all DNS servers in the world from the NS of a domain [40]. Replication of the records is done through a pull structure. Instead of NS of a domain pushing the authoritative records to every DNS server, a non-authoritative DNS server pulls the RR records when one of the clients of the non-authoritative DNS server requests it. When a non-authoritative DNS server pulls the data from another DNS server, the pulling server acts as a client and stores the data in its local database. Every RR information has a time to live (TTL) value, and a DNS server will keep a record in its cache (unless it is queried again) until the TTL duration expires.

There exist many RR types that are kept in a DNS database. Below are the essential RR types that are widely used [39]:

- NS record type: A NS record indicates the IP addresses of the authoritative name server(s) of a domain. For redundancy reasons, it is better to use multiple geographically dispersed NS
- SOA record type: Start of Authority (SOA) record points the cutover point where the domain is delegated from its parent domain. For example, if the domain domain.nl is delegated to name servers X; the name server X must include an SOA record for the domain.nl in its authoritative DNS records. Every domain needs to have an SOA record for a DNS to function correctly for the domain name
- A record type: A record type keeps the IPv4 address of the host of the domain. when DNS address is queried, the A record in the DNS database ensures the translation of a hostname (domain name) to an IPv4 address
- MX record type: An MX record ensures the handling of incoming e-mail for the domain. An MX record refers to a full hostname of a mail server (full hostname must point to an A record). An MX record holds an extra valued called priority to determine the receiver of an email when multiple MX records are present within the DNS zone of a domain. In case of failure on the incoming mail server with top priority, the email will be received by servers with lesser priorities
- TXT record type: A TXT record is a very versatile record type and is used for many applications. Next chapters will give brief information about TXT records and how they are used

3.2.1) TXT records

A DNS TXT is a free format record can be up to 65535 (0xFFFF) bytes long [41]. Capability to store such size of data allows users to use the TXT records in many applications where domain owners need to prove identity, or share a public key, or delegate authority to a particular service to act on behalf of the domain name. DNS based service discovery standardized RFC 6763 the usage of the TXT records, and it gives detailed information regarding various scenarios. As an example, we will investigate the tudelft.nl domain name and check how the domain owner uses TXT records.

Figure 6 illustrates the response of a DNS query executed on 10 September 2018. The DNS query requested the TXT records of the tudelft.nl domain name from ns1.tudelft.nl DNS server, which is one the authoritative NS of the tudelft.nl domain:

```
DNS server handling your query: ns1.tudelft.NL
DNS server's address: 130.161.180.1#53

tudelft.NL      text = "v=spf1 ip4:131.180.0.0/16 ip4:130.161.0.0/16 ip4:145.94.0.0/16
ip4:18.195.63.225 ip4:35.156.240.218 ip4:94.143.211.204 include:servers.mcsv.net
include:_spf.mailcampaigns.nl include:spf-a.tudelft.nl ~all"
tudelft.NL      text = "google-site-
verification=vIPyHrqp8W5EwG3sBPAoZXrMFzyYmVJeDIDshyLCrzk"
tudelft.NL      text = "MS=ms22016283"
tudelft.NL      text = "MS=6A1047C526547CB468CBC883DC902BD16F7ACF4F"
```

Figure 6: DNS query response for TXR records of tudelft.nl

There are four TXT records for the tudelft.nl domain. The beginning of the TXT records usually used as an indicator of the service that the record is meant for. The first record is for the sender policy framework (SPF). The services of Google and Microsoft use the other three records for a challenge-response verification. When a domain owner would like to use a

service using the servers of a service provider such as Google, the service provider requests from the domain owner to demonstrate that he/she controls the domain name, by executing the following [42]:

- Step 1: The service provider generates a random key, and asks domain owner to create a TXT record using a special syntax that includes the key
- Step 2: The domain owner adds the TXT record using his/her domain name providers DNS service
- Step 3: The service provider checks the DNS records of the domain whether there exists a TXT record that the service provider requested from the domain owner
- Step 4: if the TXT record exists and the content of the record is correct than the service provider considers that the party that requested to use a service on behalf of the domain owner is the domain owner itself. Otherwise, the request is rejected

In the next section, we will analyze an example usage of TXT records in a security service to build an understanding of the relevance to the subject of the research. New PKI ecosystem will use some ideas that will be explained in the next chapter as a building block of the new ecosystem.

3.2.2) DomainKeys Identified Mail (DKIM) Protocol

DomainKeys Identified Mail (DKIM) is an email authentication protocol that uses cryptographic measures to authenticate email messages [43]. DKIM allows the email recipient to verify the email sender, whether he/she is an authorized sender for a domain. DKIM gives domain owners flexibility to include the elements of an email message to be part of the signing process as their requirements. The domain owner can decide to have the full email message in the authentication process, or only some fields of the email header can be included in the authentication. The elements that the domain owner choose to include in their DKIM signing process cannot be unchanged while a message is in transit. Changing the fields or adding header information (if the message passes through several SMTP servers) during transmission will break the DKIM signature and the authentication will fail.

DKIM requires domain owner to add a special TXT DNS record with a specific name called [selector]._domainkey.domain.tld. The selector field is the name that the domain owner assigns to the public key. The content of the record holds the v=DKIM1 mark and the RSA public key of the domain to be used in the DKIM protocol. DKIM support currently only RSA as a signing algorithm, and SHA1 and SHA256 as hash algorithms. Finally, the domain owner must configure his/her email server to sign the outgoing email messages as follows:

- Step 1: Domain owner sets up the email server. The configuration includes the private key of the key pair (the public key is declared in the DNS TXT field), the selector, and the fields that he/she would like to be part of the signing such as from-address
- Step 2: Before email is transported to the destination server, the email server of the sender takes the hash of the fields configured in step 1 and signs it with the private key and attaches the signature as a header to the email messages together with the required fields for destination email server to verify the email message.

Signing step is a quite complicated step due to SMTP transport standards. The fields need to be processed before it is hashed and signed. Following is a brief explanation of the variables (separated by “;”) to be included in a DKIM header [43]:

- v = the version of the signature specification and is always equal to 1
- a = signature algorithm, supported algorithms are RSA-SHA256 and RSA-SHA1. Using RSA-SHA1 is not advised due to collusion in SHA1
- s = the selector to locate the public key in the DNS TXT records of the domain
- d = the (sub)domain that the DKIM should look for to locate the public key. It is usually the sender (sub)domain name
- h = list of the headers that are part of the signing process. The order of the headers listed here is the order of the DKIM signing processes. Therefore DKIM verification must execute the verification in the same order
- b = the Base64 encoded signature of the hash of the headers listed in the h tag
- bh = the hash value of the message body

Figure 7 illustrates an example DKIM header. The important aspect of the example is the usage of the variables in a signature body. The DKIM-signature data does not only hold the signature, but it also holds pre-agreed records for the receiver to verify the signature.

```
DKIM-Signature: v=1; a=rsa-sha256; s=SaperatorKey; d=domain.tld;
h=From:To:Subject:Date:Message-ID; bh=6jvzju4KoYwCJpdUP0ZiUg7GrBc=;
b=a+UCPboLbLFcrcNI5UiO8FlxvA6TiapxUE8wEeQeINEaKtg91WvBHd44GMtyETY9BjyG4FZAR
MV9Cy4AluHCqkNA322Yr1NFmsbqXWt9Z0eSnyT5+uGXRJXtL0C0xHZxIKiKANIX+mg1UxcUam
7zLzerTZRp5gYCM3YZbog6bvg=
```

Figure 7: Example DKIM header included in an email

Validating a DKIM signature

The destination email server checks the headers of the received email message whether there exists a DKIM signature. If yes, validating the signature requires the destination email server to query the DNS records for a TXT record under the specified selector and (sub)domain name in the header. For above example place of the TXT record in the DNS database is SaperatorKey._domainkey.domain.tld. Upon retrieval of the public key, the destination email server first generates the hash using the fields listed in the h variable. The destination email server uses the values in the content of the email to find the message the calculate the hash. For the above example, the destination email server must parse the email message and retrieve the from, to, subject, date, and Message-ID fields to find the message to calculate the SHA256 value. Further, the destination email server validates the signature using the found hash, the retrieved signature, and the key queried from the DNS TXT record.

There is one major flaw of DKIM. DKIM relies on a single verification of an unverified public key that is declared on a non-secure channel, a DNS TXT record. Although it is quite complicated to execute, it is theoretically possible to spoof a single destination email server using a DNS poisoning attack. The ConsensusPKI will use domain keys structure of the DKIM protocol in the certification process.

3.3) PKI ecosystem

The current public key infrastructure (PKI) is the essential security measure that is widely used in the cyberspace. The X.509 standard [1] describes the rules, and procedures to create, manage, use and revoke digital certificates bound to an identity to be used in public-key encryption systems. The primary purpose of PKI is to verify the identity of a public key, and it relies on trusted third parties. A trusted third party helps the communicating parties

to verify the identity of the counterparty they exchange information. Current PKI ecosystem has following components to execute the required tasks needed to verify digital identities:

- Certificate Policy (CP) defines the security specification, format and the hierarchical structure of the PKI ecosystem. CP also defines the standards for the management of keys, including securely storing and handling of the public keys, such as issuing and revocation of certificates
- Root Certificate Authority (CA) is an entity that its public key is anchored at the end user systems. The root CAs are the most powerful entities of the ecosystem, as they can transfer the trust to other parties, the intermediate CAs
- Intermediate CAs are the entities that are usually used to verify identity to issue a certificate. Certificate issuance is the process to sign the public key of the requestor using the private key of the signing CA. A root CA must trust the public key of the signing private key of the CA for path building on the end-user client to work. Depending on the transferred trust, an intermediary CA can also transfer trust to the third level, to another entity
- Certificate Database is the name of the database that a CA stores
- Certificate store holds the trusted root certificates in the end user system
- Revocation Services are the servers within the infrastructure of a CA that sends the Certificate Revocation Lists (CRLs) or Online Certificate Status Protocol (OCSP) responses to the requestors. OCSP is used as an alternative to CRLs. The CRLs and OCSP work only for intermediary CA or service certificates. Revoking trust from a root certificate can be executed only by the end user, by removing the certificate from the certificate store, or by a software update, that removes the certificate from the certificate store of the device
- Digital Certificate is a “digital identity” of an entity (for example a web server) that holds the signed Public key, subject (the common name CN), and related information about the entity so that the end user systems can find a validation chain path to trust the certificate
- Certificate Chain is a hierarchal trust consist of many certificates and starts from a root certificate to several branches of intermediary certificates and ends by a service corticate. A service certificate that sits at the end of the hierarchy must satisfy all of the following conditions to be trusted:
 - The entity in the prior certificate must sign the next certificate building the chain
 - None of the certificates in the chain is expired
 - None of the certificates in the chain is revoked
 - All certificates in the chain satisfy the policies specified by prior the certificates

The X.509 PKI is highly distributed ecosystem with many geographically separated CAs that needs to be trusted. The following sections will give more information about the current PKI ecosystem, which will be a baseline for this thesis since ConsensusPKI is an alternative to the current PKI.

3.3.1) Management Protocols

The current X.509 defines many management protocols to set up and maintain its infrastructure [1]. We will shortly visit those management protocols as the thesis will

propose new PKI infrastructure replacing some of those management protocols. Following are included in the standard:

- The registration process is the initial process before certification and can be compared to registering to an e-commerce website before purchasing goods. At the end of registration, the user can send certificate signing requests to the CA that he/she is registered
- Initialization protocol refers to setting up of a client system. It is the trust anchoring process where clients (the vendors of the end user system) stores the trusted certificates into their trust store
- Certification protocol is certificate issuance protocol, where a CA, after required verification for the identity in the certification request file, signs the public key in the CSR file and creates a certificate file that includes relevant information to be used in certification path validation
- Key pair recovery refers to the recovery of an issued certificate after it is lost. This optional protocol requires the CAs to keep a copy of CSR or issued a certificate in a certificate database for its customers so that the certificates can be recovered
- Key pair update refers to issuing a new certificate using new key pair for an identity. As a best practice, it is advised to update the key pairs regularly, recently as an industry standard, the CA and browser vendors agreed to limit the maximum validity of a certificate by 825 days [44]
- Revocation request refers to the process of informing the issuer CA of a certificate by the owner of a certificate. Usually, it is executed by an individual, after identifying him/herself using the information given during the registration process. After receiving a revocation request, a CA must include the certificate in the CRLs or OCSP responders as revoked
- Cross-certification is a protocol that enables CAs to sign the public keys of each other. Other than standard hierarchical trust, the cross-certification is not hierarchical and establishes a horizontal relationship between CAs in both directions

The management protocols of X.509 PKI creates a flexible environment for CAs to operate. Following are the main responsibility of a CA [5]:

- Protecting the private keys of the CAs
- Verifying the Identity of the entities in a CSR
- Executing revocation requests

The ecosystem allows issuance of multiple certificates at the same time to a single identity (a CN). The thesis will propose a change in this approach and will include a structure to constrain the issuance of multiple certificates to a single CN.

3.3.2) Certificate Structure and Extensions

Certificates are the identity cards on the internet. The basic structure and fields of the certificates are defined by RFC 5280 [1]. There are many fields (extensions) that a certificate can carry [45]. However, we will shortly visit them and explain the essential parts of X.509 certificates, which will guide our design process for the new PKI ecosystem. Later, the thesis will propose new fields because of the changes in the ecosystem. Following are the primary fields of the certificates in the current ecosystem:

- **Version** field specifies the version of the certificate format (1, 2, or 3)

- **Serial number** field is the unique ID of a certificate assigned by the issuer. An issuer can use a serial number only once
- **Algorithm ID** specifies the signing algorithm such as sha256RSA indicates the combined use of SHA256 and RSA
- **Issuer** field specifies to name the issuer CA. The name is a DN comprised by CN, OU, C, N fields
- **Valid from** and **Valid to** fields specify the validity period for a certificate.
- **The subject** field is the name of the certificate owner. Like issuer, the name is a DN comprised CN, OU, C, N, L fields
- **Public Key** specifies the public key and its algorithm

Starting with certificate version 3 it is possible to add extension fields to the certificates, called shortly extension. Following are some critical extensions:

- **Subject Alternative Name** specifies additional unique CN fields that identify the subject (Such as additional DNS addresses)
- **Key usage** identifies the purpose of a certificate. Digital Signature, Key Encipherment, Certificate Signing, Off-line CRL Signing, CRL Signing are some possible values of key usage
- **Basic constraints** field identifies the subject type and set a limit to trust path. As an example, a CA certificate with basic constraint equal to **Subject Type=CA** and **Path Length Constraint=0** cannot transfer trust to another CA.

3.3.3) Certificate Revocation Lists and Online Certificate Status Protocol

Even though certificates have a validity period, it might be that a private key of a certificate is compromised. In that case, the trust to the compromised certificate must be revoked so that an attacker can not use the compromised certificate. The X.509 model is a hierarchical trust model which also gives the possibility to create a horizontal trust relationship between CAs. The cross-certification is a convenient tool that creates multiple trust chains for a single certificate [45]. In case of a trust revocation from a CA certificate, the horizontal trust relationship reduces the impact of the revocation for other certificates that have an indirect relationship with the revoked certificate. This complex trust relationship structure handles the trust revocation using two protocols. The first one is the certificate revocation lists (CRLs). A CRL is a list that holds the records of the certificates that are valid (not yet expired) but are revoked. A revocation status of a certificate can only be checked if the certificate holds a record that points a CRL distribution URL. During certificate validation, the client software that executes the validation must check the CRL distribution URL and check whether the received certificate from the server is revoked. Further, it is not required to keep a revoked certificate in the CRL after it is also expired. An alternative to CRL, the Online Certificate Status Protocol OCSP can be deployed [46]. The OCSP, in essence, a straightforward protocol. Instead of CRL distribution URL, a certificate can hold an OCSP responder URL where the status of the certificate can be checked. In both of the protocol's CAs gains a power to profile the usage of the websites that the CA issued a certificate since clients must connect to the URL of a CA for certificate validation. Furthermore, in the initial OCSP protocol, the response messages received from a CA can be attacked since it had no cryptographic protection. The issue is later resolved with OCSP stapling, where the response messages are signed and timestamped by the CA. However, privacy concerns remain unaddressed.

3.3.4) X.509 certification path validation

X.509 certificate path validation is the most critical process of the PKI infrastructure, as it creates an indication for the end-user system to interpret the received certificate whether to trust it [47]. The certificate validation is executed in three steps, and requires following inputs to validate a certificate:

- The certificate to be validated (the target certificate)
- The intermediate certificates (if there are any)
- The certificate store, such as OS or browser certificate store, that the root and intermediate certificates are kept

Following steps are executed consecutively to verify a target certificate, if any of it fails, the validation fails.

Step 1: Chain Construction and Signature Validation

A certificate cannot be trusted as long as the signature on the certificate is not validated. That is why the first step of the validation process is to locate the signing certificate in the certificate store. Once the signing (intermediary) certificate is found, the validation process checks the signature on the target certificate using the public key in the signing intermediary certificate. Searching the correct intermediary certificate in the certificate store is done based on the **issuer** field on the target certificate. The issuer field on the target certificate must match with the **subject** field of the signing intermediary certificate. If the signing certificate of the target certificate is not in the root of the certificate store, the chain construction continues until a signing certificate is found that is in the root of the certificate store. If any of the signature checks fail or no root certificate is found, the validation fails.

Step 2: Validity, Policy and Key Usage checks

After successfully constructing the trust chain, the validity, policy, and key usage checks are executed using the information in the certificates in the chain. Each certificate is checked separately. The validity is checked using the date time of the end user system and issue and valid until fields in the certificates. The date time of the end user system must be between the issue and valid until days of the certificate. Further, all certificates in the chain are checked whether they satisfy the ruleset defined in each certificate in the chain. Following are the most important checks that need to be executed for all certificates in the constructed chain:

- BasicConstraints certificate extension defines whether an intermediary certificate can sign another intermediary certificate by defining the limit to the depth of a certificate chain. All certificates above the target certificate must be within the set limit by any of the CA certificates in the constructed chain
- KeyUsage certificate extensions limit the purpose of a certificate. For example, a server authentication certificate is not allowed to sign and issue a certificate (act as a CA). The root and all intermediary CA certificates, except the CA certificate that signed the target certificate, in the constructed chain must have KeyUsage extension that allows signing of a CA certificate
- Subject certificate extension (the CN field, or Cn fields) defines for which service the target certificate can be used. The validation process checks whether the target certificate is used for the service in the CN or Cn fields

If any of the above checks fail, the validation of the target certificate fails.

Step 3: Revocation Check

After successfully executing the checks in step one and step two, the validation much checks whether any of the certificates in the constructed chain is revoked. There are two possible protocols to execute the revocation check, the Certificate Revocation Lists(CRLs), and the Online Certificate Status Protocol (OCSP). A certificate may contain extensions that point to a URL of a CRL of the signing CA or a URL that points to an OCSP responder of the signing CA. Interestingly, the CRL or OCSP is not mandatory, and a certificate in the constructed chain may not be checked for the revocation status. Additionally, the CRL is updates based on the nextUpdate extension in the CRLs. It is possible that a certificate remains trusted even though it is placed in the CRL of a CA since the validating client would download the update of the CRL on nextUpdate set by the existing CRL. The OCSP designed as a remedy to the time gap issues of CRLs. The OCSP check is straightforward. When a certificate (CA or target) contains an extension to check the revocation status of a certificate on an OCSP server, the validation process sends OCSP check request using the serial number, the hash of the issuer name, and the hash of the subject name in the certificate to the OCSP URL defined in the target certificate. The OCSP server gives one of the following responses good, revoked, or unknown. The validation fails if any of the certificates in the constructed chain is in the CRLs or for any of the certificates the responsible OCSP server responded as revoked.

3.4) Blockchain Technology

A blockchain is a data structure, organized in a way that is almost impossible to temper. It uses mathematical and cryptographic functions to create its tamper-proof structure. It all started with a published paper of the bitcoin as a peer to peer online cash system that remedies the double spending problem and requires no trusted third parties [14]. The writer of the paper, pseudo-named Satoshi Nakamoto, combined many well-known cryptographic functions and structures in a chained way that removes a requirement to have a trusted third party in a digital cash system without needing of a clearinghouse in the conventional banking industry. With blockchain, the public **consensus** replaces the trust. Nowadays there are two types of blockchains. The first one is public blockchains. A public blockchain network is an open peer to peer network that anyone is allowed to join. The bitcoin is an example of such a network and has many participant servers called nodes that help to execute bitcoin transactions. An alternative to public blockchain networks, a private blockchain network requires the approval of participating nodes to join the network. Node provisioning acts as an umbrella for parties to at least to know each other before getting in a business relationship. Because of that executing transaction in a private blockchain can be pre-agreed to require fewer resources creating high throughput compared to public blockchains [48].

The core element of the blockchain technology is its data storing structure, a ledger, that a copy is distributed to all participating nodes. Sometimes distributed ledger is used as an acronym to the blockchain. However, a distributed ledger alone does not create public trust. The critical requirement to create a trust in a distributed ledger is to create a consensus among the entire network about the content of the ledger. The consensus mechanism provides a verification mechanism to ensure only valid data to be appended into the ledger. There are many consensus mechanisms currently used by blockchain networks.

However, we will shortly discuss the two widely used consensus mechanisms; Proof of Work (PoW), and Proof of Stake (PoS) mechanisms [48] [49].

The PoW uses a cryptographic puzzle, by solving the puzzle a blockchain node demonstrates (proves) that it worked to add the block, containing the transactions, into the chain. The puzzle for the bitcoin network is simple: a node needs to find a special hash value of the block to add the block into the chain. The only way to find the special hash value is to try different nonce values one by one to hopefully find the special hash. That is why finding the special hash value requires a lot of computer power. Even though a blockchain network requires PoW to add blocks into a chain, the issue of honesty among the nodes remains unanswered. Theoretically, as long as, the majority (in computation power) of participating nodes remains honest, the trust will be sustained. The incentive for participating nodes to stay honest is the reward of PoW. Theoretically, the participating nodes will profit more in the long run if they stay honest for the ecosystem to stay trustworthy. These are all theoretical issues of **public** blockchain network. In a permissioned blockchain network, the ecosystem will be less hostile, since all participating nodes are provisioned before they join the network. Following is the lifecycle of a block before and after it is added to the chain:

- The non-verified data (transaction requests) is bundled together into the memory pool of a blockchain node (all blockchain nodes manages their memory pool)
- Blockchain nodes try to verify the data one-by-one by solving a cryptographic puzzle
- The very first blockchain node that solves the puzzle broadcasts the solution to the network and adds the data to the chain coupled with the solution of the puzzle, the block header
- Depending on the agreements in the network, the node that solved the puzzle is rewarded

For PoW to work the puzzle need to have the following properties:

- The puzzle needs to be a one-way function, a hash function, so that finding the answer would be difficult, but others could easily verify a correct answer
- The only way to solve the puzzle must be trying one by one by brute-forcing so that no node would have an unfair advantage (the only advantage can be more computer power) over other nodes

Moreover, the difficulty of the puzzle should be parametric, in a way to keep the block addition interval to the chain, to be stable. As an example, the bitcoin network targets ten minutes block addition time. Therefore, the difficulty is maintained in a way that on average, the bitcoin network can add one block on every ten minutes [48].

In contrast to PoW, the PoS structure does not require solving the puzzle to add a block. PoS uses a deterministic algorithm (similar to poker game) based on stakes. Before a block is added to the chain, the participating nodes put some stakes to add a block to the chain. The higher the number of stakes, the more chance to put the block in the chain. For PoS to work the nodes must posses stakes (tokens). For example, if one node would stake x tokens (because the nodes must validate x number of data of its own) and another node stakes $2x$ tokens (because the nodes must validate $2x$ number of data of its own), the higher bidder node has more chance to be the node that validates the block. Furthermore, the node that adds a block to the chain receives fees from the data owner (for cryptocurrencies a transaction fee). The most significant advantage of a PoS consensus that it does not

require nodes to have much computational power, thereby it is also very energy efficient [49].

On the other hand, due to its consensus structure, a blockchain can fork. Forking happens when two different nodes find different new blocks with different content retrieved from the memory pool at the same time. When a blockchain forks, in PoW systems, each node must decide about the version of the blockchain that the node will accept as the truth. For the bitcoin network, from the forked chains, the one that gets one additional chain the first usually accepted as the truth. The choice is very logical, if a blockchain node supports two chains at the same, it must share computational power between the chains, and that decrease the change of solving the next puzzle to get a reward. Having consensus on PoS does not require very high computational power. As a result of that, when a blockchain forks on a PoS consensus, the node owners will have their stakes for both of the chains at the same time, as they can continue participating to verify both of the chains. Further, the blockchain node with PoS consensus no longer needs to participate in verification, causing node owner to gain the ability to double-spend their stakes. Furthermore, no incentive or prevention mechanism exists to prevent or to demotivate the node owners not to double-spend their stakes, after such an event. The problem is called “nothing at stake” problem and new versions of PoS consensus tries to prevent this bad behavior, by requiring a security deposit to join the network. Later the deposit can be used for punishment of lousy behavior [49].

In the following sections, we will shortly visit the substructures of a blockchain network to be a baseline for the design choices of the thesis.

3.4.1) Memory Pool & Timestamping

The bitcoin paper identifies the chain structure as a time stamp server. A blockchain keeps events in consecutive order by using the blockID rather than the timestamps. The events in every block are a continuation of the events in previous blocks, and the timestamps of each block are set by the blockchain node that finds the block header, and no mechanism guarantees the accuracy of the timestamps [50]. The accuracy of timestamps could be essential to reconstruct the events in the correct order if they are not kept in a blockchain form. Furthermore, the timestamp is included in the calculation of the block-header, and it is the second variable (together with the nonce) used by the verifying node.

When a blockchain node receives a request to be executed and put in a blockchain, the request is put in a queue, and it stays there until it is put in a block. Following is the process flow of a message received by a node:

- The requester sends the request to a Blockchain node that he/she prefers
- The receiving blockchain node sends a broadcast message to the network for all blockchain nodes to include the request in their memory pool
- Every blockchain node tries to put the request in the next block
- As long as a request is not included in a block, it stays in the memory pool

For a PKI ecosystem, the management of the memory pool and the correct order of evidence are important aspects, since it may change overall experience of end users and network administrators. The new ecosystem will include timestamping of all the events in the network even they are not yet confirmed, for the sake of completeness. Finally, a certificate has a lifespan and validity duration, in the current X.509 certificates the validity is

checked against the time of a client system. In the new ecosystem, this will not be the case since the certificates will not only be validated by using client date time.

3.4.2) Merkle Trees

A Merkle hash tree is a binary data structure constructed using a hash function [51]. A Merkle tree can have many leaves, nodes but only one root. Each leaf represents a data, and a node represents a set of the data, and finally, the root represents all of the data. Figure 8 demonstrates a Merkle tree that represents four data points {D1, D2, D3, D4} where a leaf represents a hash of each data in {L1, L2, L3, L4}. The Leaf L1 and L2 are put together, and the hash of L1 || L2 forms the node N1. The same structure is applied in the right side of the tree where L3, L4, and N2 are found. Finally, N1 and N2 concatenated to be input to the hash function to find the Merkle root R. This simple structure has following beneficial properties that can be used in various applications. First of all, a data point can be a part of the Merkle tree without revealing its content. Secondly, proving the existence of a data point does not require to keep any of the data points or their leaves. If we consider the data point D1 to prove itself that it is in the Merkle tree represented by root R. D1 only needs to store L2 and N2 without knowing the contents. Further, D1 can apply the same logic to find R, and together with the data D1 proving its representation in the Merkle tree. The structure is tamper proof since a single bit change in any of the data will give different root R'. Since it uses hash functions, it is not possible to retrieve other data points. Furthermore, we can generalize the needed data points to be used in a proof as follows:

$$\text{Required data} = (\text{number of levels}) \times \text{hash output size}$$

As an example, a data point in Figure 8 requires keeping two additional hash data L1, and N2 since it has two levels besides the root, to be used in a proof. The thesis will use Merkle trees in various places for proofs and representation of the data kept in the ecosystem.

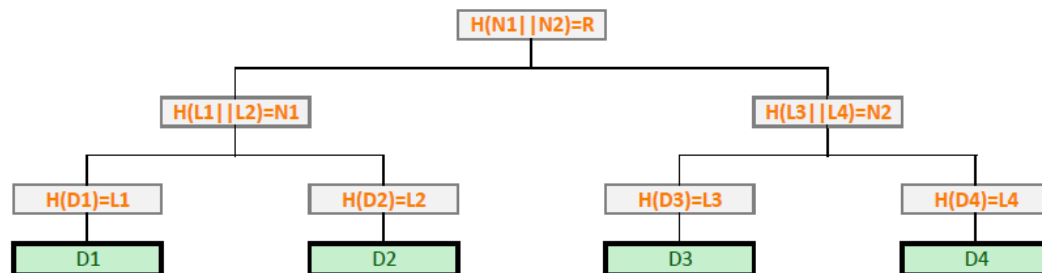


Figure 8: A Merkle Tree

3.4.3) Distributed Network

It would be less valuable to keep a close dataset in a single computer in a blockchain form, supported by Merkle trees for proofs and audit purposes. It makes sense keeping a public data in blockchain form distributed on many computers independently, in a way that the data stored and processed separately by each computer in the network independently, but all nodes finally have the same data. In a blockchain network, the distribution of the data is through broadcast messages, and there is no central authority. Only a distributed network removes the requirement of the trusted third party.

The thesis will propose a structural change for the communication between the certificate authorities forming a blockchain network, where everyone can read the data.

However, the data processing will be executed by the loosely coupled nonhierarchical certificate authorities.

3.4.4) Oracle

An oracle is an agent that feeds blockchains with required external data since blockchains cannot access data outside the network. It should not be confused with a random oracle, which is a mathematical abstraction used in cryptographical proofs.

The thesis will use information sources outside of blockchain network to retrieve several publicly available information such as DNS records. However, an oracle in this thesis refers to a network element within a CA infrastructure, usually deployed on the same host as the blockchain node.

3.4.5) Practical Byzantine Fault Tolerance (PBFT)

Byzantine generals' problem (BGP) posed the problem in 1982 by Lamport et al. [16]. In a distributed computing network, such as a blockchain network, there might be a rogue computing node with different agenda than other honest nodes, or the infrastructure of a node might be compromised by another actor. Determining the bad actors brings us to BGP.

The thesis will use the Practical Byzantine Fault Tolerance (PBFT) algorithm proposed by Castro, and Liskov in various scenarios, to determine the conclusion of an algorithm in certain cases [52]. The PBFT proposes the following formula to calculate the number of allowed faulty nodes in an environment that has N number of nodes.

$$N = 3f + 1, \text{ where } f \text{ is number of faulty nodes}$$

3.5) Automatic certificate management environment (ACME)

Certificate issuance is conventionally a manual task that service owners execute step by step to receive an issued certificate for their services. Once a certificate is issued, system administrators manually import the certificate into their hosts to use in services for protocols such as HTTPS. The ACME [53] is proposed to automate this process, initially for Let's encrypt certificate authority. However, the environment is a flexible environment and can be used by all certificate authorities.

Following are the requires steps executed by the ACME before and during certificate issuance:

- Step 1: Initial key generation and handshake with Let's Encrypt CA infrastructure. The ACME requires the website owner to create a key pair coupled with contact information. Using the information and the key the ACME initiates a handshake with the CA. In the handshake, ACME client signs the messages using the generated key. Later, all message exchange with the CA will be executed using the key pair generated in this step
- Step 2: Proving ownership of a domain. Following are the substeps
 - The ACME first sends a request to CA to initiate the authorization process for a domain.tld
 - The CA server requests to prove that ACME controls the web server of the domain.tld, and CA requests the web server of the domain to put a particular resource on a specific URL (a nonce challenge response, such as requesting to put 5544D2Sf on <http://domain.tld/X26T53fd>)

- After ACME placing the resource under the requested directory, it informs CA to check the URL
- CA checks the URL, and if verification is finalized, the ACME client with the initial key is now authorized for the domain domain.tld
- Once the ACME client that holds the initial key authorized for the domain.tld, it can send CSR to CA server, and CA server will issue a certificate for the domain.tld and send it to ACME.

The ACME protocol relies on a single challenge response executed between only one server and the ACME client, through an open channel. The thesis will use an improved version of the ACME protocol. We name the improved version as the APKME protocol. The APKME uses two key pairs instead of one. Additionally, the APKME verifies the public key and the identity of the subjects by multiple challenge-response verifications executed by multiple independent parties compare to single verification executed in the ACME protocol.

Chapter 4) ConsensusPKI – A Blockchain based public key ecosystem

In previous chapters, we have given detailed information about the structural flaws of the current X.509 PKI ecosystem as we currently use on the internet and discussed the scientific efforts to remedy the issues. When we go back to the first days of the PKI ecosystem, we can draw the internet landscape at that time as follows:

- Connectivity bandwidth was minimal
- Processor power and random memory size of end user systems was deficient compared to today's standards
- Number of connected devices were minimal
- Blockchain was not yet introduced

Given the landscape, the logical choice was to introduce a trusted third party that verifies the identities on the internet. Starting from SSL V1, all successors of all security protocols on the internet inherited the trust model introduced in the early days of the internet. As we use the PKI in many places, it becomes every day even more difficult to change this backbone infrastructure in the cyberspace. As a result, scientific research focuses on solving the structural flaws of the PKI ecosystem for some scenarios and mainly by using its core idea, the trust.

The focus of this thesis is to create a new infrastructure that would solve all infrastructural flaws of the current PKI infrastructure, and it changes or replaces the elements of the current infrastructure in the new ecosystem. This new infrastructure is called ConsensusPKI because it uses a public consensus model instead of a trust model, which was the core of the widely used X.509 PKI ecosystem.

The ConsensusPKI aims to use the stakeholders of the current PKI to create a practical and inviting ecosystem for the scientific and business community, that might be hesitant to adopt a new infrastructure.

4.1) Ecosystem focused solution & general characteristics

In this thesis, we focus on the ecosystem using a holistic approach, and thereby it differs from the other approaches that focus on specific issues. As a remedy to the trust related flaws of the X.509, in ConsensusPKI there will be no trust or any hierarchical transfer of trust. The role of CA will change from a trusted third party to the public key verifier and evidence maintainer. Because there will be no trust, there will be no signatures on the public keys (current form of certificate issuance). The certificates in ConsensusPKI will be linked to cryptographic evidence and proofs of an identity, which is linked to a particular subject (a domain name). In the new ecosystem, there will be only one valid identity (public key) for a specific subject for any given moment. During certificate issuance, the service owner must not only demonstrate that he/she controls the subject (the domain name) but also demonstrate that he/she possesses the private key part of the public key he/she would like to have verified. In contrast to X.509 certificate issuance, the demonstration of the control on the subject will be executed many times (the number of times depends on the assurance required), and the collected evidence will be linked to the issued certificate and will be part of the protocol (such as TLS) handshake.

Further, the ecosystem consists of blockchains that are linked to each other. The blockchains will keep the data as evidence for the certificates. The blockchains are organized in a way that storing the certificate data by the CAs is not needed. Furthermore, the data size will not increase continuously, as ConsensusPKI holds a mechanism to prune unnecessary data of expired certificates. The certificate path processing is replaced by online public key query protocol while maintaining the privacy of the end users in a way that profiling of web usage of the end users will not be possible by an actor.

Since the data in the blockchain is the core of the ConsensusPKI ecosystem, adding data into blockchains, and querying data from the blockchains are critical processes. In the ConsensusPKI, all processes around the data are strictly controlled. The ConsensusPKI delivers required algorithms for CAs to execute to add certificate information into blockchains. Further, ConsensusPKI delivers an algorithm to update the CA public keys on end-user systems. Finally, an algorithm is delivered to query and verify the public key of a specific subject to be used during connection handshake (such as TLS). The following sections describe those sub-elements of the ConsensusPKI.

4.1.1) Verifiable evidence instead of signatures

One might think that a signature is always more valuable than evidence. However, the digital signatures on certificates on the X.509 ecosystem are the sole evidence that a single certification authority verified the public key in a certificate signing request (CSR) file. Furthermore, in the current PKI ecosystem, there is no evidence of proper verification bound to a signature, in fact, no one knows how CA verified the CSR of a particular certificate. As it is not required to demonstrate evidence of proper verification, a forged certificate can be easily created by certification authorities in the X.509 scheme.

ConsensusPKI ecosystem has an entirely different approach to the certificate issuance process. A certificate in ConsensusPKI is issued once it is added to related blockchain (which is equal to the signing of a certificate in x.509). For a certificate to be added to the blockchain, it is required to have multiple verifiable pieces of evidence of proper verification. As there is no signature, but multiple verifications and evidence of those verifications, a revocation of a CA certificate will affect **only** the CA, and the certificate revocation will only impact the CA to no longer participate in certificate issuance or the public key query processes of the ConsensusPKI.

There are three groups of evidence in the ConsensusPKI:

- Evidence collected during certificate issuance to prove that the requestor of the certificate verification had control over the subject and demonstrated that he/she has control over the key pair is being verified on the server of the subject
- Evidence that the verified key of the subject exists in the public blockchain
- Evidence that a CA has not been honest, or it has been compromised

4.2) A General view of the ConsensusPKI

The following sections describe the components of ConsensusPKI. The components are grouped under the following categories:

- The data structure: the Blockchains, the Merkle trees, the certificates, the data fields and their relationship with each other
- The algorithms to verify the Public keys before it is added to the blockchains
- The protocols to append the data into the Blockchains

- The algorithms to store, query and update the CA public keys on the end-user systems
- The algorithms to query and check the validity of a certificate on an end-user system

The elements of ConsensusPKI falls under three main categories. The first one is the public evidence datastore kept in blockchain form (on CA infrastructures) to be used for the cryptographic proof of the public keys. The second one is the public evidence datastore required on end-user systems for anchored CA nodes. The third one is the algorithm to replace X.509 path processing, and fetch algorithm to retrieve the latest version of the anchored node blockchain to the end user system. Following are the blockchains, their roles, and how they are linked to each other:

- Blockchains for the regular subjects (regular certificates such as domain.tld)
 - Root Blockchain: the root blockchain is responsible for finalizing a year for a particular certificate blockchain once all certificates expire for a year. it keeps one block for every year, and the block represents all regular certificates for that past year
 - Certificate Blockchains: Certificate data is split into multiple blockchains based on the expiry year, and once CAs verify a public key, the related information is put to the blockchain that holds the records for the year that the certificate will expire. For example, if a certificate will expire in 2019, the related data will be put in the certificate blockchain for 2019
 - Evidence Blockchain: During the certificate issuance, CAs will no longer sign public keys, but they will verify the public key of a service. During the verification process, multiple pieces of evidence will be collected and stored in this blockchain, specific for the certificate verification request (a small blockchain for each certificate verification request (CVR)). Data in the blockchain will form a merkle tree be part of final certificate Merkle tree where the root of, will be stored in the certificate blockchain
- Blockchains for the Anchored verifiers (Blockchain nodes)
 - Root Blockchain: The role is similar to the root blockchain for regular certificates, but it is specific to blockchain nodes (the CA certificates)
 - Certificate blockchains: This blockchain will be similar to the regular certificate blockchains with one difference. There will be only one CA certificate in each block of the blockchain, and the certificate of the CA including the public key is kept in the block. The certificates of the CAs are used during verifications and proofs of regular or CA certificates
 - Evidence Blockchain: It will serve the same purpose as the evidence blockchain of regular certificates. However, it is specific to the blockchain node (CA) certificates
- Merkle Hash trees
 - Certificates: Each block of certificates blockchains will contain one merkle root hash that represents 1024 certificates. A Merkle tree is constructed of 2048 leaves, where 1024 leaves are the hash values of certificates and the other 1024 leaves are the root values of the related evidence Merkle tree of a certificate
 - Evidence: The blocks of an evidence blockchain of a particular certificate will form a Merkle tree with 16 or 32 leaves (each block is one leaf), and the Merkle

root of the evidence will be part of the Certificate Merkle tree and a Certificate blockchain

The root blockchain is a remedy for sizing issues that a single blockchain would suffer. All certificates to the linked certificate blockchain must expire before a block can be added to the root blockchain. Further, each block of the root blockchain will store only one Merkle root for the data it represents. Calculating the Merkle root requires the block headers of all blocks of the linked certificate blockchain to be used as leaves of a Merkle tree. Once a block is added to root blockchain, related certificate blockchain can be archived from the data store of a CA (Blockchain node).

Apart from the blockchains and Merkle trees, we developed a tool which is deployed in the OS on the end users systems to verify the certificates received from a service (such as website www.domain.tld). We call the algorithm used in the tool PKQuery algorithm. This tool replaces the current X.509 path processing algorithm and will use randomly selected CAs to query the public key proof of a certificate. The PKQuery client will not only query and verify the public keys of services, but it will also detect and inform when a CA is not honest. In all of the executions and data exchange, the privacy of the end users is preserved. Figure 9 illustrates a general view of the ConsensusPKI. In the next sections, we present detailed information about the ecosystem. The Appendix B: Data model overview for the CA Certificates, and the Appendix C: Data model overview for the Regular Certificates illustrates the data model of the ConsensusPKI in a nutshell.

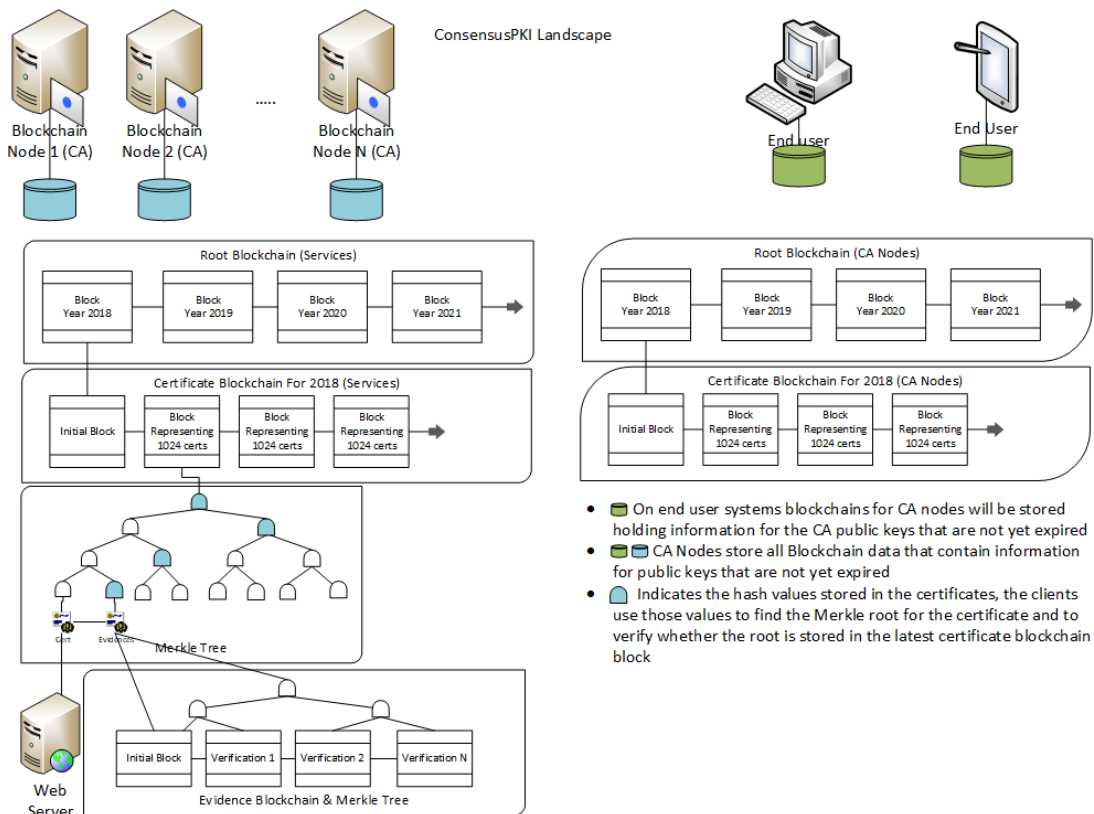


Figure 9: ConsensusPKI landscape

4.3) Blockchains, Merkle Trees & Certificate Structures

Current PKI infrastructure is a complex environment of loosely coupled elements that lack integrity checks on those elements. We propose the following sub-structures that are linked to each other to create a full functioning PKI ecosystem with multiple interconnected blockchains. Compared to one append-only blockchain, these multiple interconnected blockchains do not suffer sizing issues and are much easier to manage. The proposed structure creates the possibility to prune data that is no longer required to keep, increasing the scalability of the ecosystem. Following is a brief introduction to the new sub-structures:

The **Root Blockchain** is a blockchain that stores the Merkle root of all certificates and evidence information that was processed for a given year in the past. The organization of the Root Blockchain is based on the expiry year of the certificates. For every calendar year, there exists one block in the root blockchain. It is allowed to append data only to one of the three independent certificate blockchains that are not yet linked to the root blockchain (as long as the valid certificates exist). The non-linked blocks are for the certificates for the current year, the next year, and the year after that. Once all the certificates in a non-linked block expire, the chain will be connected to the root blockchain by calculating first the Merkle root for that year, followed by the block hash. Next, a new non-linked chain can be created. Figure 10 demonstrates the structure of a chain. For example, for the expiry year 2018, the current year is equal to 2018, and it is allowed to append data to the current year block till the end of 2018. The block in red is a representation of the expired certificates of the previous year and appending data to the certificate blockchain is no longer possible for that particular year. The block header of the red block in the root blockchain below is calculated. As the block in the root blockchain is finalized, it is no longer required to keep the data of expired certificates, and the data for the linked certificate blockchain can be purged or archived. It is only possible to append certificates into the certificate blockchain of non-linked chains of the root blockchain. As illustrated in Figure 10, it is possible to add data into the certificates blockchains that are linked to the blocks of the root blockchain illustrated in green below (based on expiry year). There exists only one constraint; if there exists a certificate for a subject in a certificate blockchain, it is only allowed to add a new certificate for the same subject in the same linked certificate blockchain or the next certificate blockchains. As an example, there exists a certificate for domain.tld that will expire in 2019 and a new certificate can be added in the certificate blockchain for 2019 or later, but not in the certificate blockchain for 2018. The link is the previous (initial) block header of the first block of the certificate blockchain for the year that the root blockchain represents.

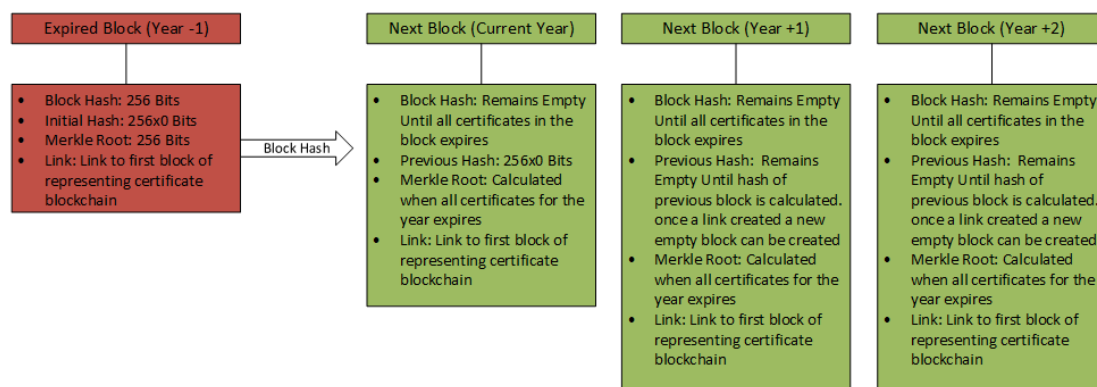


Figure 10: Structure of the Root Blockchain

The **Certificate Blockchain** is responsible for keeping the Merkle root of the certificate data. Each block will keep the Merkle root for a fixed pre-agreed number (for example 1024) of certificates, and **hashes** of all CNs belonging to the certificates represented in the block. Hashes of CNs will be used as an index to find the correct path of a valid certificate while maintaining the **privacy** of the end users. For every year that exists in the root blockchain, a separate blockchain will co-exist. Figure 11 illustrates an example certificate blockchain. The initial hash, for example for the year 2018 is H(2018) which is the link to the Root Blockchain.

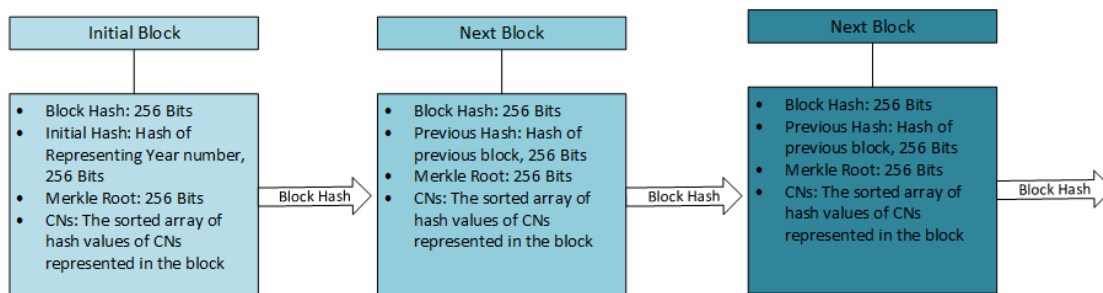


Figure 11: Example Certificate Blockchain for a year N

Evidence Blockchain is a temporary blockchain and is created separately for every Certificate Verification Request (CVR). The evidence data in every verification executed by a CA will be kept in this blockchain during the verification process of a CVR. Once a CVR finalizes, the evidence blockchain will be linked to the issued certificate as extended information. The link will be created through a Merkle root. The block hashes of this blockchain will be part of a Merkle tree, and only the Merkle root of this tree will be needed during a protocol handshake, such as the TLS handshake. Figure 12 demonstrates an Evidence Blockchain.

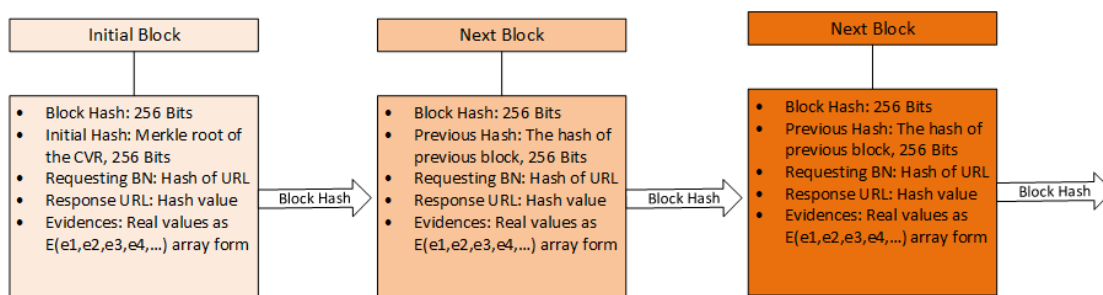


Figure 12: Example Evidence Blockchain for a CVR

4.3.1) Root Blockchain structure

The purpose of the root blockchain is to reduce the total storage requirement of a CA. The size of the bitcoin blockchain grows on every block added. The bitcoin structure is handy to calculate the balance of a specific bitcoin wallet. If we would apply a similar structure to the ConsensusPKI, the total size of the blockchains will become an issue after some time. The root blockchain serves as a data integrity solution for the certificates that are stored in a year in the past. If someone would like to check whether a certificate was in the blockchain for a given year, he/she can use the copy of the certificate blockchain data for that year and check the certificate whether it was stored in the related block. For CAs in the ConsensusPKI, the information for the expired certificates have limited use, and it is better to archive and remove it from the production systems. When a year ends, there will be no certificate issuance for that year anymore. The CAs can purge the linked certificate blockchain safely after a block is added to the root blockchain for that year. The PoW will be used to achieve consensus among the CAs to add a block to the root blockchain. The following data are required for the PoW to be executed:

- Previous hash: the block hash value of the previous block of the root blockchain
- The link: link is the hash value of the year that the block represents, which is also the initial hash value of the certificate blockchain of that year
- Block creation timestamp: it is the network timestamp when the block is created. The value is included in the calculation of the block header
- Nonce: it is an arbitrary number for the consensus mechanism to work. For example, the blockchain network might require finding a block header that starts with certain data (such as two ones "11")
- The Merkle root for the certificate blockchain that is linked to the block

Construction of the Merkle tree to add a block

The block header for each block of the certificate blockchain representing the past year will be a leaf for a Merkle tree. From left to right, the consecutive block headers will form a Merkle tree starting from the initial block. After construction of the Merkle tree, the Merkle root will be calculated. The required total number of calculations depend on how many blocks exists in the certificate blockchain. As we proposed to keep a fixed number of certificates on each block (1024 certificates) of the certificate blockchain, the required total number of hash calculations to find the Merkle root is one less than the number of blocks. After calculating the Merkle root, the block will be added into the root blockchain by PoW. The incentive for CAs to search for the special block is the constraint, that limits maximum non-linked certificate blockchains. As long as the block for the past year certificates is not added to the root blockchain, the CAs will not be able to process certificate verification requests with a two-year expiry time, causing them to lose revenues. Figure 13 illustrates the construction of the Merkle tree.

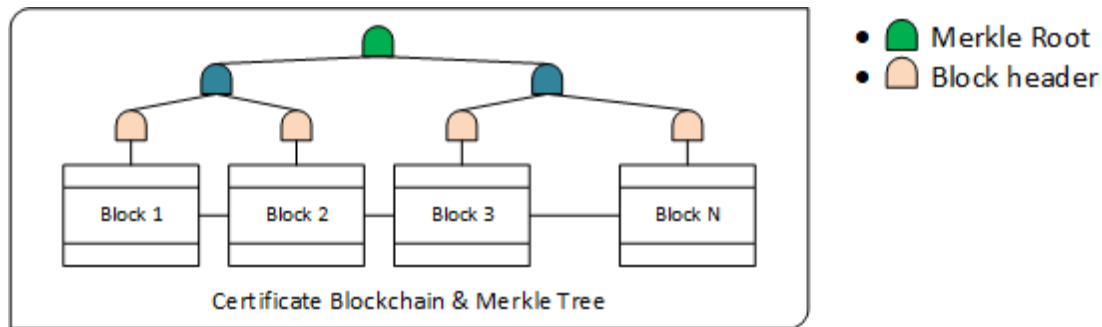


Figure 13: Certificate Blockchain Merkle Tree

4.3.2) Certificates Blockchain

The certificate blockchains serve as the consistency proof mechanism of the ConsensusPKI. Each block of the certificate blockchain will hold a record that represents a fixed number of certificates (such as 1024). Following are the fields that will be kept in this blockchain and their purposes:

- Previous hash: the block hash value of the previous block
- Block creation timestamp: it is the network timestamp when the block is created. The value is included in the calculation of the block header
- Nonce: it is an arbitrary number for the consensus mechanism to work. For example, the blockchain network might require finding a block header that starts with certain data (such as two ones “11”)
- Subjects: a sorted array of the hash values of subjects. (all CN and CAN fields in all certificates represented in the block)
- Merkle root: the Merkle root of the certificate Merkle tree of the block

The ConsensusPKI requires all certificates that are represented in the blockchain to be verified before they can be added into a block. The verification mechanism will be transparent to all CAs, and the evidence collected in a specific blockchain will be kept in the memory pool of all nodes so that they can be included in a block. The initial block of each certificate blockchain will be the hash value of a certain year such as $H(2018)$ where H is a hash function.

Block addition process

Before a certificate can be added to a block, it must be verified, and the evidence must be controlled for the consistency check. A certificate is not issued until it is not in a block. Certificates issuance requires CAs to place the certificates in the blockchain. Following are the steps executed by a blockchain node when a block header is calculated (assuming that the network decided to include 1024 certificates in a single block):

- 1024 **verified** certificate requests will be selected from the memory pool
 - All CN and CAN values will be collected in an array
 - Hash Fingerprints of all certificates will be calculated
 - All evidence, for all certificates, will be checked for consistency
 - For every evidence blockchain, the block headers will form a Merkle tree
 - Merkle root will be calculated for every evidence blockchain
- The certificate Merkle tree will be constructed
- Subjects array will be constructed

- Hash values of all CN and CANs will be calculated
- The new array is sorted
- Using the PoW mechanism, CAs calculate the block header

Once the block is added into the certificate blockchain, the issuing CAs of all certificates in the block can create a certificate file containing all required information. The CA that finalizes the PoW must broadcast a message that includes the nonce, the block timestamp, and the list of the certificates. The message must include the order of the certificates in the certificates Merkle tree, for the block to be verified and included in the blockchain by all other CAs.

4.3.3) Evidences Blockchain

The evidence blockchain is a proof mechanism used during the certificate issuance when the issuing and the participating CAs verify certificate verification requests. When a service owner would like to have a public key to be verified, he/she needs to send a CVR to a CA of his/her choice. Upon receiving a validation request, the CA will lead the verification process for the certificate issuance. Following lists the process and how the data is stored in the evidence blockchain:

- The CA will execute the initial verification, and upon success, it will broadcast a message that contains the following:
 - The original CVR
 - The initial block of the evidence blockchain
 - A list of **randomly selected** participating CA blockchain nodes
- Each CA in the list will repeat one by one the verification step, and upon success, each CA will add a block into the evidence blockchain containing the result of the verification and the evidence
- The issuing CA will check whether the required number of participating CAs finalized the additional verifications. Upon completion, the data will be posted to the memory pool to be added into the related certificate blockchain

The required number of verifications will be a pre-agreed number. Based on the earlier discussed ZKP and PBFT fundamentals, the advised number of consecutive verifications can be 16 out of 20 or 32 out of 40 verifications. For example, let's assume 16 consecutive verifications are required. Based on the ZKP, 16 consecutive verifications out of 20 (the first one is the issuing CA) different CAs, gives the certainty of the truth as $\sim 1 - (1.5/100000)$. From the perspective of PBFT, it is not required to have the remaining four verifications since it is less than the maximum acceptable false verifications based on the following equation:

The total number of nodes = 20 = 3f + 1 where f is the maximum number of allowed false verifications. 6 false verifications are acceptable in the above scenario.

Following data will be required in the evidence blockchain for the collected evidence to be verified when required:

- Previous hash: the block hash value of the previous block
- The requesting CA (Blockchain Node): the CN of the requesting CA (such as node1.blockchainnode.tld)

- The response URL: the response URL of the service owner's verification system (APKME) which is responsible for answering the challenges received from the CAs
- Block creation timestamp: this is the network timestamp when the block is created. The value is included in the calculation of the block header
- The evidence: an array of evidence with the pre-agreed order as such {e1, e2, e3, e4,...}. By using the evidence, anyone can verify that all public key verifications took place

4.3.4) Certificates Merkle Tree

The certificates Merkle tree is a mechanism to reduce the required storage. The blocks of the certificate blockchain will not keep the full data, but it will keep the root of a Merkle tree where certificates and their evidence are organized in a way that, it is possible to prove that a certificate is in a Merkle tree. On the leaf level, one leaf holds the fingerprint of the Certificate data, and the sibling of the leaf holds the Merkle root of the evidence Merkle tree. By doing so, the evidence is linked to the certificates, and its existence can be proven. Figure 14 illustrates the structure for four certificates instead of a pre-agreed number (proposed 1024), but the methodology will be the same. In the figure, the Certificate A requires the values e, n and m for its existence to be proved in the Merkle tree. During the construction of the Certificates Merkle Tree, the certificates will be placed in the leaves of the tree based on the order that a CA desires. It can be a lexicographic or random order of the hash fingerprint of the certificates. When a CA broadcasts the PoW result, the broadcast message must contain the order of the certificate Merkle tree for PoW to be confirmed by other CAs.

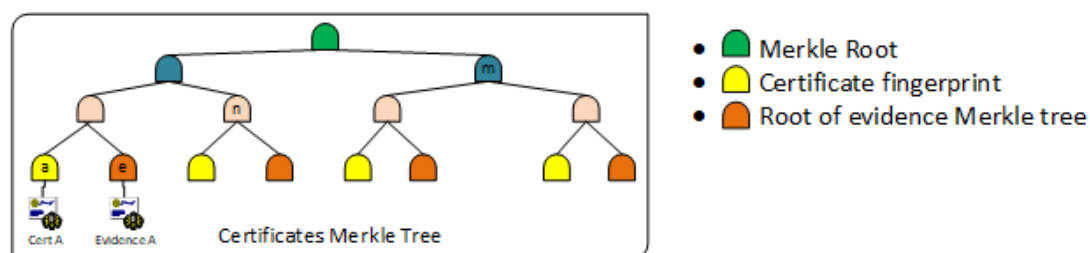


Figure 14: Certificates Merkle Tree

4.3.5) Evidences Merkle Tree

Evidence Merkle tree is again a construction to reduce the required storage. Instead of keeping all evidence as a whole, the evidence Merkle tree is constructed from the block headers of the Evidence blockchain. The Merkle root of the evidence Merkle tree is put as a sibling to the certificate hash, and therefore it is possible to prove its existence. The construction structure is very similar to the yearly Merkle root which is calculated to add a block to the Root blockchain, but the evidence Merkle tree is specific for the evidence of a single certificate. Figure 15 illustrates the organization of the evidence blockchain and the Merkle tree construction.

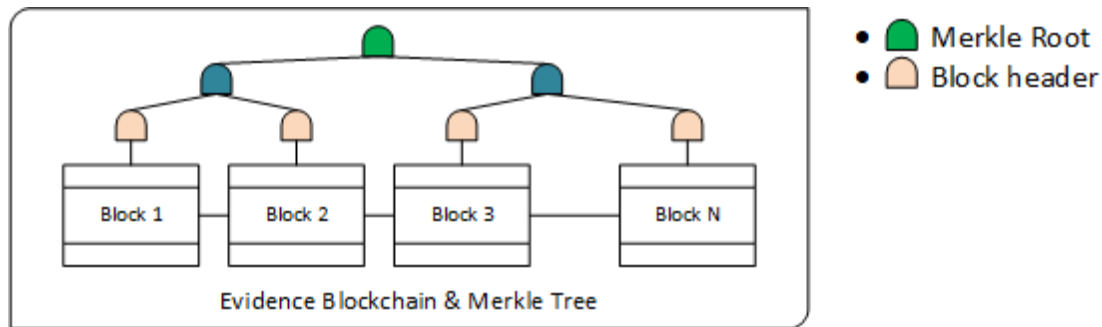


Figure 15: Evidence Blockchain and Merkle Tree

4.3.6) Root Blockchain structure for Blockchain Nodes

The X.509 PKI requires public keys of the trusted root certificates to be available on end the user systems for X.509 path processing mechanism to work. In X.509 the trusted root certificates are independent of each other, and there is no mechanism to verify whether a root certificate should be trusted or not. The trusted certificate store on end-user computers exists as a remedy to define the start of the trust. In contrast, in the ConsensusPKI there is no trust, and the role of a CA changes from trusted third party to Blockchain maintainer. For the **message authentication**, it is required to have the valid public keys of all CAs to be available on the end-user systems. The public keys of CAs will be used by end users to verify the authenticity of a message received from a CA when a public key is queried. Furthermore, it is also required to have the public keys of the CAs available to all CAs, since certificate issuance is a collective effort and the messages exchanged during the certificate issuance need to be authenticated using the public keys of the CAs. We propose a solution to store and distribute the public keys of the CAs similar to the regular certificates with the following differences:

- The root blockchain for CAs will contain a flag to distinguish it from the Certificate root blockchain. The flag is called CA=true
- The difficulty level of the PoW algorithm will be considerably higher than the root blockchain for certificates

4.3.7) Certificates Blockchain for Blockchain Nodes

Similar to the certificate blockchain for regular subjects, the certificate blockchain for the CAs will be organized by the expiry year of a public key of the CAs. The data structure of the blockchain is similar with the following differences:

- Each block will contain one certificate, and there will be no certificate Merkle tree. Instead, the certificate will be kept directly in the blocks. As a result, the public keys of all CAs will be kept directly in the blockchains to decrease the execution time of the message authentications
- There will be only one Merkle tree to store the root of evidence Merkle tree representing the evidence collected during the provisioning of the CA in the block

4.3.8) Evidence blockchain and Merkle tree for Blockchain nodes

Evidence blockchain and the Merkle tree will be constructed similarly to its sibling in the blockchain for the regular certificates. The only difference could be the number of required verifications. The certificate authorities are a significant aspect of the ConsensusPKI. Increasing the number of verifications to 64 would make it very hard for

attackers to launch a successful attack in order to add a fake CA. Meanwhile, 64 verifications increase the certainty of a public key of a CA to $1-(1/2)^{64}$.

4.3.9) Certificate structure & fields

The structure of the certificates in ConsensusPKi will be similar to X.509 certificates. However, the data end extensions will serve the processes of the ecosystem. First of all, the certificate will contain the part of the CVR with the following information:

- The public key
- The signature of the public key signed by the domain key
- An identifier to retrieve the domain key, hash function, encryption, and signing schemes
- An expiry date
- Common Name (CN)
- Common Alternative Names (CANs)
 - CAN is an array and can have multiple records
- Extended information supported by the BCN to help visitors to identify the service

The information in the CVR alone is not sufficient for the certificate to be verified by the end user systems during indication and interpretation. The following information will be needed:

- All other related Merkle nodes to construct the Merkle root
- An indication of the location of the certificate in the certificate Merkle tree

The data stored in the certificate file will be used during the interpretation of the end-user systems. The end user system will find the Merkle root using the Merkle proof in the certificate file. The calculated Merkle root must be equal to the value stored in the latest certificate blockchain block of which the certificate is included for the subject (domain name).

4.4) Certificate issuance: Zero-knowledge identity proof backed by Blockchain & Consensus

As explained in the previous sections, there are two essential aspects the certification process should verify. The first one is to verify the identity of the requester of the certificate verification. The participating parties to the verification process must check whether the requestor is the entity or person that controls the domain name in the certificate verification request (the CN field). The second aspect is whether the requestor indeed possesses the private key (the pair of the public key that he/she would be using, for example, for an HTTPS deployment) and whether he/she deployed the private key to be used.

We propose the following solution as an answer to both the requirements mentioned above. We will use the following functions and variables as building blocks for our solution:

- K_{pub} = Public key for identity verification. The controller of the domain name must create a DNS TXT record holding this public key under a special section that the domain owner prefers. As an example, it can be placed [identifier]._identitykey.domain.tld.

- Identifier = the identifier of the DNS TXT record for the verifier to locate the K_{pud} using a DNS query
- K_{prd} = Private key and the pair of K_{pud}
- K_{puw} = Public key to be used for HTTPS deployment of the domain. The domain owner must demonstrate that he/she possesses the private part of this public key.
- K_{prw} = Private key and the pair of K_{puw}
- BCN = Blockchain network of certificate authorities
- $\text{BN}_{(i)}$ = A blockchain node and a certificate authority
- $\text{N}_{\text{pu}(i)}$ = Public key of the identity verifying $\text{BN}_{(i)}$, the key is publicly known and verified. The verification process will be explained in the next section where we explain the trust anchoring process in the new PKI ecosystem
- $\text{N}_{\text{pr}(i)}$ = Private key and the pair of $\text{N}_{\text{pu}(i)}$
- MP = Memory pool, responsible for the management of the identity verification queues that are not yet added into blockchains. Every certification requests a $\text{BN}_{(i)}$ receive; a message will be first broadcasted to the BCN. The memory pool mechanism will work similar to the bitcoin network [14]
- $\text{WN}_{(i)}$ = Nonce that client (APKME) chooses during the verification by $\text{BN}_{(i)}$. The server of the domain owner acts as a client during verification
- $\text{SN}_{(i)}$ = Nonce that $\text{BN}_{(i)}$ chooses during the verification
- $\text{CN}_{(i)}$ = Domain address that APKCE Server must be deployed and holds the K_{pud} , K_{prd} , K_{puw} , K_{prw} , identifier, for the domain CN

The scenario: A domain name [domain.tld] owner would like to have the public key of his/her service under www subdomain to be verified (in current PKI terminology certificate issuing) and relevant information must be added to the blockchain, so that the visitors of his/her service can verify that the public key is the verified public key of the service. Interpretation on the client side will be explained in section 4.6) Indication and interpretation on the client side: The public key query protocol.

Setup:

Step 1: Domain owner sets up his/her automatic public key management environment (APKME) which is an improved version of ACME [53] on the same server that he/she would like to host his/her service. APKME will execute the tasks required by the new ecosystem, and requires the following information to be able to execute the tasks:

- K_{pud} , K_{prd} , K_{puw} , K_{prw} , the identifier
- Hash function identifier (such as SHA256)
- Signing scheme identifier (such as RSA)
- encryption scheme identifier (such as RSA)
- $\text{N}_{\text{pu}(i)}$ of all CAs (as it is publicly available through anchoring mechanism)

Step 2: Domain owner creates a DNS TXT record under identifier._identitykey.domain.tld that contains the K_{pud} . Further, the domain owner creates an additional DNS A record for the URL of the APKME environment for identifier.domain.tld for all domains in the CVR file. The CAs will send their challenges to the host in the DNS A record under identifier.domain.tld

Step 3: The domain owner creates a certificate verification request (CVR) with the following information

- Kpuw
- A signature sign(Kpuw) signed using Kprd
- Identifier
- Identifier of the hash function to be used in verification (such as SHA256)
- Identifier of the signing scheme to be used in verification (such as RSA)
- Identifier of the encryption function to be used in verification (such as RSA)
- Common Name (CN), for this example www.domain.tld
 - CN is an array and can have multiple records (in that case DNS TXT must be added under the same identifier for all CNs)
 - Multiple CN records can be under same domain or different domains
- Extended information supported by the BCN to help visitors to identify the service

Verification request:

Website owner sends the CVR to $BN_{(i)}$ of his/her choice for the verification process to start. During this step, $BN_{(i)}$ might request payment for the provided service and would require the website owner to select a validity duration (limited by the possible maximum duration by the BCN) for the K_{puw} . The possible expiry date for the certificate must be later than the latest expiry dates of the current certificates of all CNs in CRV file. As an example, let's assume that the service owner has tree certificates and would like to merge them into one certificate with CANs. The Expiry date of the new certificate must be later than all expiry dates of the certificates already issued for the CANs. This is a critical design choice as ConsensusPKI holds only one valid certificate for a subject at any moment.

Timestamping, adding to the queue

The issuer $BN_{(i)}$ will first timestamp the CVR. The issuer BN will first execute the initial verification. Upon success, it will create the first block of the evidence blockchain of the CVR. Furthermore, it will broadcast a message to BCN for other participating blockchain nodes to execute additional verifications. Upon finalization of the required number of verifications the issuing $BN_{(i)}$ will finalize the verification executing the following steps:

- Finalization of evidence blockchain and construction of evidence Merkle tree
- Calculation of root of the evidence Merkle tree
- Broadcasting a message to BCN containing the CVR, evidence Merkle root and evidence Blockchain to be added in the related certificate blockchain based on the expiry year

Initial Verification

In the verification process, APKME acts similar to an oracle, which is usually an information source, but APKME will solve simple puzzles that every $BN_{(i)}$ sends to the APKME. By solving the puzzles, APKME will demonstrate that it controls the CN and holds the K_{prd} and K_{prw} . Following are the verification steps:

- **Step 1:** $BN_{(i)}$ will take a hash of all CNs in the CVR separately and sort the hash values to determine the CN verification order. $BN_{(i)}$ will use the hash function identified in CVR. Each $CN_{(i)}$ in the CVR will be verified separately. The following steps will be executed for every $CN_{(i)}$
- **Step 2:** $BN_{(i)}$ queries the DNS TXT record to retrieve the K_{pud} using the $CN_{(i)}$ and the identifier in the CVR

- Step 3: $BN_{(i)}$ verifies the signature in the CVR file using K_{pud} . If this step fails for any $CN_{(i)}$, the verification for CVR fails, for all CNs in the CVR
- Step 4: $BN_{(i)}$ chooses a random nonce for $CN_{(i)} = SN_{(i)CN(i)}$
- Step 5: $BN_{(i)}$ calculates the hash of $SN_{(i)CN(i)} = H(SN_{(i)CN(i)})$ using the hash function identified in CVR
- Step 6: $BN_{(i)}$ encrypts the $SN_{(i)CN(i)}$ using $K_{pud} = Enc_1(H(SN_{(i)CN(i)}))$ using the encryption algorithm identified in CVR
- Step 7: $BN_{(i)}$ encrypts the $Enc_1(SN_{(i)CN(i)})$ using $K_{puw} = Enc_2(Enc_1(H(SN_{(i)CN(i)})))$ using the encryption algorithm identified in CVR
- Step 8: $BN_{(i)}$ signs the $Enc_2(Enc_1(SN_{(i)CN(i)}))$ using $N_{pr(i)} = Sign_1(Enc_2(Enc_1(H(SN_{(i)CN(i)}))))$ using the signing scheme identified in CVR
- Step 9: $BN_{(i)}$ creates a message $M\{s1,s2\}$ where
 $s1 = Enc_2(Enc_1(H(SN_{(i)CN(i)})))$
 $s2 = Sign_1(Enc_2(Enc_1(H(SN_{(i)CN(i)}))))$
- **Step 10:** $BN_{(i)}$ sends the message M to APKME hosted on $CN_{(i)}$
- Step 11: $CN_{(i)}$ verifies the signature $s2$ on message M using $s1$ and the $N_{pu(i)}$. if this fails $CN_{(i)}$ does not respond to the message
- Step 12: $CN_{(i)}$ decrypts the second encryption in $s1$ first to find $Enc_1(H(SN_{(i)CN(i)}))$ using K_{prw} that it possesses
- Step 13: $CN_{(i)}$ decrypts the first encryption find $H(SN_{(i)CN(i)})$ using K_{prd} that it possesses
- Step 14: $CN_{(i)}$ chooses a random nonce $WN_{(i)}$ (each client nonce $WN_{(i)}$ must be used only once)
- Step 15: $CN_{(i)}$ adds the nonce $WN_{(i)}$ to $H(SN_{(i)CN(i)}) = WN_{(i)} || H(SN_{(i)CN(i)})$
- Step 16: $CN_{(i)}$ calculates the hash of $WN_{(i)} || H(SN_{(i)CN(i)}) = H(WN_{(i)} || H(SN_{(i)CN(i)}))$ using the hash function identified in CVR
- Step 17: $CN_{(i)}$ signs $H(WN_{(i)} || H(SN_{(i)CN(i)})) = Sign_{c1}(H(WN_{(i)} || H(SN_{(i)CN(i)})))$ using K_{prd} that it possesses
- Step 18: $CN_{(i)}$ encrypts the $WN_{(i)}$ to find $Enc_{c1}(WN_{(i)})$ using the public key of the node $N_{pu(i)}$
- Step 19: $CN_{(i)}$ signs $Enc_{c1}(WN_{(i)}) = Sign_{c2}(Enc_{c1}(WN_{(i)}))$ using K_{prd} that it possesses
- Step 20: $CN_{(i)}$ creates a reply message $R = \{m1,m2,m3,m4\}$ where
 $m1 = Enc_{c1}(WN_{(i)})$
 $m2 = Sign_{c2}(Enc_{c1}(WN_{(i)}))$
 $m3 = H(WN_{(i)} || H(SN_{(i)CN(i)}))$
 $m4 = Sign_{c1}(H(WN_{(i)} || H(SN_{(i)CN(i)})))$
- **Step 21:** $CN_{(i)}$ sends the message R to $BN_{(i)}$
- Step 22: $BN_{(i)}$ checks the signatures $m2$ and $m4$ in the message R using the K_{pud} of the $CN_{(i)}$. if signature verification fails means R did not come from $CN_{(i)}$, or there is a configuration issue on CNs system, verification will fail for all CNs in the CSR
- Step 23: $BN_{(i)}$ decrypts the $m1$ in the message to find the client nonce $WN_{(i)}$ using its own private key $N_{pr(i)}$
- Step 24: $BN_{(i)}$ knows the its selected random nonce $SN_{(i)CN(i)}$ and will construct the following $m3' = H(WN_{(i)} || H(SN_{(i)CN(i)}))$
- **Step 25:** $BN_{(i)}$ checks if $m3 = m3'$, if they are equal, the verification is successful
- Step 26: $BN_{(i)}$ will repeat the initialization for all CNs in the CVR from step 2 to step 25

- Step 27: if verification is successful for all CNs in the CRV, BN_(i) will broadcast successful verification to BCN. BN_(i) will create the first block of verification chain for the CRV request

Figure 16 illustrates the initial verification for a single domain CN. When CVR holds multiple CNs, the verification needs to be executed for all CNs. After successful verification of all CNs in the CVR file, the issuing CA will broadcast the message for other participating CAs to execute their verifications.

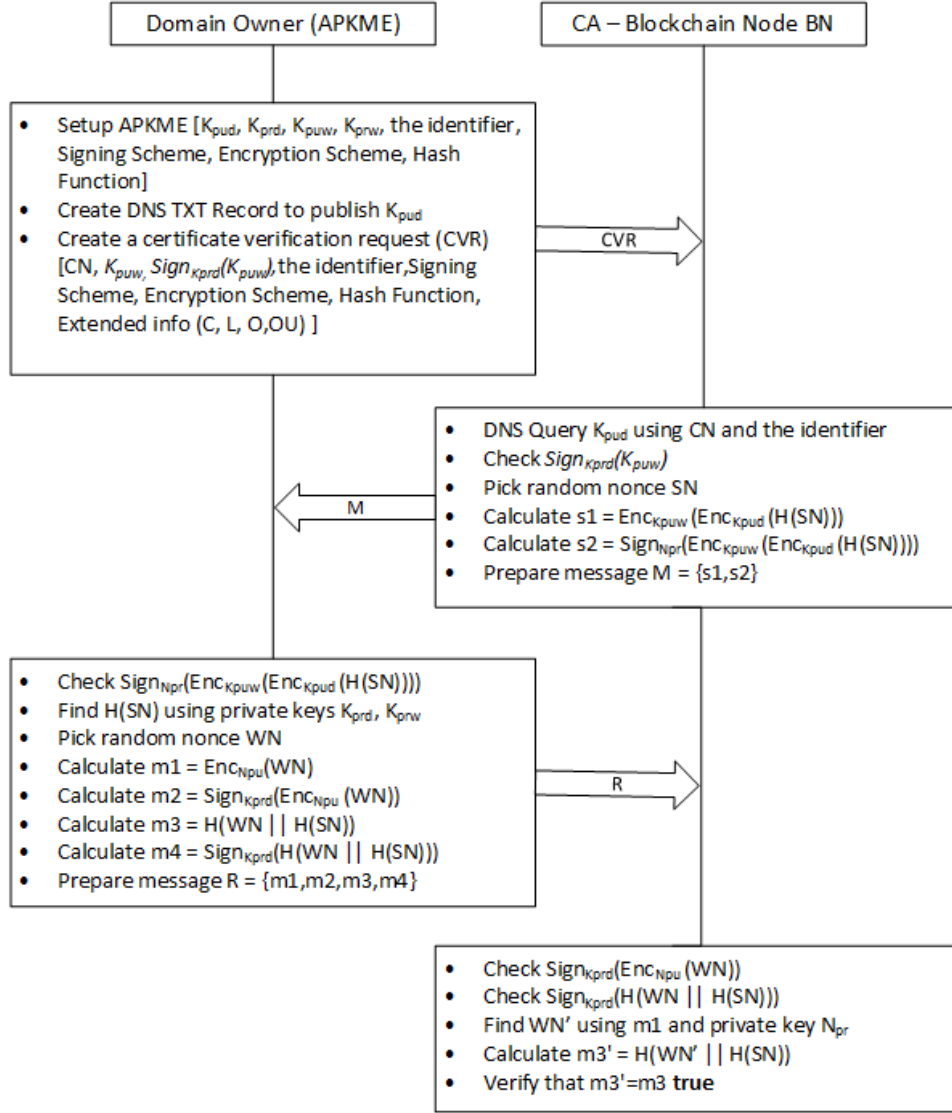


Figure 16: Single Domain and Public Key Verification

Although the initialization process is protected using cryptographic primitives, it relies on the information in two DNS records:

- A record (host IP address) of all CNs in the CVR
- TXT record for the $_identitykey$ (K_{pud}) of the CNs in the CVR

Since the DNS is not a secure protocol, MITM attacks are possible where the attacker sits between the web server (APKME), DNS server of a CN and a verifying BN. As a remedy to a MITM attack, in order to guarantee the security of the certification process, we propose repeating the verification (after successful initial verification) from many geographically

disperse locations using different BNs. The collaborating BNs, each applies a verification separately, collectively come into a **consensus** that the K_{puw} in the CVR belongs to the CNs in the CRV file.

The verification process (initial and additional) has four important states. The very first state is the step 1 of the verification process, which is the delivery step of the CVR file to the CA. The second state is the step 10 of the verification process, the delivery of the message M to the APKME environment of the domain owner. The third state is the step 21 of the verification process, the delivery of the message R to the CA. The final state is the step 25 of the verification process where the CA checks whether received message verifies.

Evidence

After successful verification of a public key for all CNs in the CVR file, each CA will add the following information for all CNs as evidence to a block of the evidence blockchain for the CVR. Using the evidence anyone can validate that the verification by the related CA took place. Controlling the evidence as a final verification check is also a prerequisite adding a certificate into the certificate blockchain. The final verification check will be done by the CA that adds the related block to the certificate blockchain. The following are the pieces of evidence to be added to the evidence blockchain as data in an array form:

- CN = Common name that the verification was successful
- SN = Nonce value that the BN sent to the APKME
- s1 = message that was sent to the APKME by the BN
- s2 = the signature of the message s1
- WN = Nonce value that the APKME used in the message sent to BN
- m3 = constructed, the response message sent from APKME to BN
- m4 = the signature of message m3
- K_{pud} = Domain public key retrieved from DNS query for CN

For example, the format of the evidence data can be formed as follows:

```
{
{CN1, {CN1, SN1, s11, s21, WN1, m31, m41, Kpud1}},
{CN2, {CN2, SN2, s12, s22, WN2, m32, m42, Kpud2}}
,.....}
```

Additional verifications

Once the issuing CA executes the initial verification, the additional verification will be executed by the participating CAs. Each CA will execute the verification identical to the initial verification and add a new block to the evidence blockchain with the evidence collected during the verification process. The issuing CA will follow the verifications by the participating CAs, as soon as the required number of verifications are completed. The issuing CA will send a broadcast message to the Blockchain network to add the certificate in the memory pool so that the final verification can be executed, and the certificate can be placed into the related certificate blockchain.

Final verification

The final verification is the verification of the evidence placed in the memory pool for a certificate to be added into the certificate blockchain. The required data is available as follows:

- CVR file contains the public key K_{puw} and all CNs and CANs
- Public keys of CAs are available throughout the anchoring
- Verification challenge and response messages and their message signatures are available in the evidence

Final verification checks must be executed for all evidence as follows:

- For all evidence K_{pud} is equal
- Check whether verification is executed for all CN and CANs in the CVR file
- Construct $s1'$ using the SN, K_{pud} , K_{puw} and check if $s1' = s1$ in the evidence
- Check the signature $s2$ using the public key of the verifying CA and $s1$
- Construct $m3' = H(WN || H(SN))$ and check if $m3' = m3$ in the evidence
- Check the signature $m4$ using K_{pud} and $m3$

Once the above final checks are executed for all certificates and evidence, the certificate and evidence Merkle trees can be constructed, followed by finding the block header for the block. The issuing CA will monitor this process after a certificate is added to the blockchain, the issuing CA will generate the certificate file and send it to APKME of the service owner.

The time gap between certificate issuance and the deployment

ConsensusPKI ecosystem allows only one certificate to be valid at any given moment. In practice, the time gap between issuance and the deployment of the certificate to the servers of the service owner might create unavailability for the service until APKME deploys the certificate into the server of the domain. As a remedy to the time gap problem, we propose to wait for one additional block to be added to the certificate blockchain, in order for the certificate to be the only valid certificate for the subject. During this transition time, the previous certificate and the newly issued certificates will be both valid, and the response message for the public key queries will include the Merkle root for both certificates. Alternatively, the APKME may wait to deploy the new certificate after a subsequent block is added to the certificate blockchain.

4.4.1) Certificate issuance for CAs

The certificate issuance process is similar to regular certificate issuance with the following differences:

- CA blockchain network is a permissioned blockchain network, and only provisioned CA operators are allowed to join to network. This constraint creates an incentive for CAs to protect their infrastructure.
- CA certificates are significant certificates in ConsensusPKI, as it sits in the core of the ecosystem. Thereby, it is advised to have a more significant number of validations for CA certificates. ConsensusPKI proposes 64 validations for CA certificates
- There is no parallel certificate issuance for CAs. This constraint is to narrow the possible attack surface to the ecosystem
- There will be one certificate in each block of the certificate blockchain for the CAs, which will also hold the public key of the CA

4.5) Anchoring of Public keys of CAs on End-user systems

ConsensusPKI requires the public keys of the CAs to be publicly available for underlying structures to function. Following are the required data on end-user systems:

- Root blockchain for Blockchain Nodes (CAs)
- Valid Certificate blockchains for Blockchain Nodes

An important aspect is that the certificate blockchains might be updated quite regularly. As a result, end-user systems must check for the updates regularly. Since the certificate blockchains are append-only, only the new blocks are required to be downloaded to the end user systems.

4.5.1) Distribution of Root and Certificate blockchains for CAs to End-user systems

The initial distribution of the root and currently valid blockchains should be done through the browser or OS vendors. After initial distribution, the block fetch structure must be executed regularly. The block fetch is similar to the Public Key Query protocol and consists of the following steps:

- The end-user system picks four random CAs from the blockchain nodes using the certificate blockchain data available on the end user system
- The end-user system sends block fetch requests to all randomly selected CAs
- The query response must include the signature of the responding CA
- The end-user system verifies the signature in the response
- The end-user system checks whether all responses are equal for all received blocks and whether PBFT is satisfied
- The end-user system updates the local copies of the certificate blockchain, after checking the integrity of the blocks using the earlier stored blocks and the newly received blocks

The following table illustrates a possible scenario of the CA blockchain heights on the end-user system and the CA systems:

Current Year		Current Year + 1		Current Year + 2	
Local	Real	Local	Real	Local	Real
1050	1055	306	370	120	180

Figure 17 illustrates a single messaging between an end-user system and CAs to fetch the newly added blocks to the certificate blockchains for CAs.

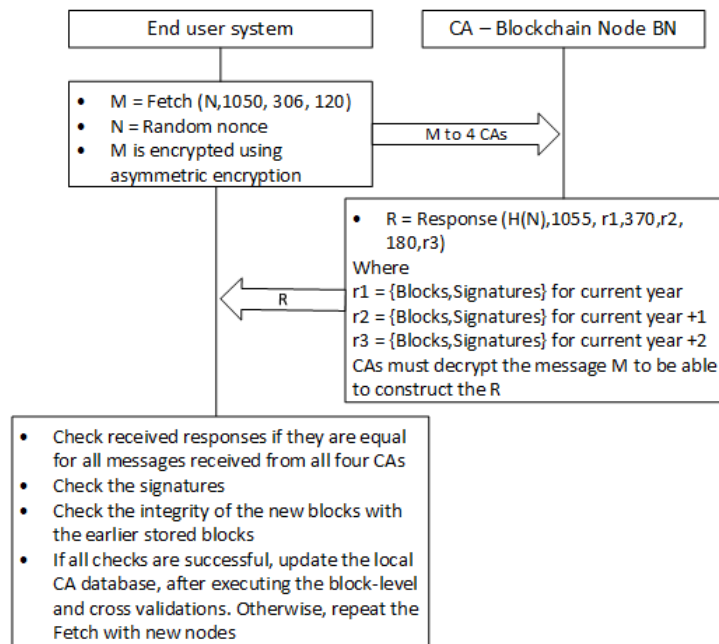


Figure 17: Fetch CA Certificate Blockchains

4.6) Indication and interpretation on the client side: The public key query protocol

The most crucial process of the ConsensusPKI is the indication and interpretation of the certificates on the client side. The validation process is independent of the communication protocol. When a client receives a certificate, the following algorithm is executed to check whether the certificate is a valid certificate:

- The client checks whether the certificate is not expired using the expiry date on the certificate
- The client checks whether the certificate holds a CN or CAN record for the service that the client is communicating to
- The client will select a random nonce N (pre-agreed size, for example, 8 bytes)
- The client sends a PKQuery request to four randomly selected CAs.
 - The client will construct a message $Q = N || H(\text{subject})$, where H is the identified hash function in the certificate, and the subject is the address of the service in lowercase (a domain name) that the client is communicating to. The client will send the encrypted query text = $\text{enc}(Q)$ using asymmetric encryption and the public key of each CA separately. The CA must decrypt the message to construct the N and $H(\text{subject})$. The CA server must query its local database using the index $H(\text{subject})$ to find the representing Merkle Root (MR) in the certificate blockchains, where MR is the latest Merkle Root for the $H(\text{subject})$
- Each CA server replies with a signed message $H(N || H(\text{subject}) || \text{MR})$ to the client
- The client constructs the Merkle root r' using the fields in the certificate and calculates $R' = H(N || H(\text{Subject}) || r')$
- The client then executes the following checks using the received responses from the CAs where responses are $\{R_{CA1}, R_{CA2}, R_{CA3}, R_{CA4}\}$ and the $R = \{H(N || H(\text{Subject}) || \text{MR}), \text{Sign}(H(N || H(\text{Subject}) || \text{MR}))\}$ and $\text{MR} = \text{Merkle root for the latest block where } H(\text{subject}) \text{ is an index}$

- The client checks whether the signatures on the responses verify
- The client checks the following $R_{CA1} = R_{CA2} = R_{CA3} = R_{CA4} = R'$

If all checks verify, the client application accepts the communication as **secure** and continues to the protocol handshake.

Query cache on end-user systems

It is arguably, whether it is necessary to query CA repositories on every TLS connection to the same service again and again. It would be very convenient for the end-user system (for example the browser) to have a caching mechanism for the PKQuery results. Before conducting a PKQuery for a service, the end-user system should first check whether a record on cache exists and if so, the certificate received from the server of the subject must match with the cached record. The cache mechanism is subject to another research.

Other cases

- If the signature on any of the CA responses does not check or a response from a CA is not received timely, the query message is sent to another randomly selected CA. The PKQuery protocol requires at least four responses with a valid signature
- If $R_{CA1} = R_{CA2} = R_{CA3} = R_{CA4} \neq R'$ client application indicates the connection as **not secure**
- If one of the CA responses is not equal to the other responses, the PBFT will be applied with $n = 3f + 1$, where n is the number of queries and f is the allowed number of faulty messages. The connection will be indicated as **secure** if remaining responses are equal to R'
 - The client will construct a message $F = \{CA, N, H(\text{Subject}), R_{CA}\}$ and send it to three other CAs as an honesty check request. The other CAs will conduct an honesty check for the CA that has sent the faulty message to the end user system. If the CA is not honest, the public key of the non-honest CA will be invalidated
- In all other cases, the connection will be indicated as **not secure**, and the client will send an honesty check request for all messages received from the CAs. In this case, each CA will receive an honesty check request containing the responses of the other CAs

4.7) The honesty check and revocation of public keys for CAs and regular services

The CA will apply the following checks upon receiving an honesty check request:

- Signature validation of the honesty check message should verify. The response of the non-honest CA message must have a valid signature, signed by the non-honest CA.
- Forward lookup: the forward lookup is identical to a normal PKQuery. The CA that checks the honesty will compare the result to its local copy and check whether it is equal.

If the forward lookup check does not validate, the CA that checks the honesty creates an evidence blockchain and broadcasts detection of a non-honest CA message. In the broadcasted message the following information will be included:

- Original honesty check request received from the end user system

- The initial block of the evidence blockchain
- List of **randomly selected** participating CAs to validate the honesty check

4.7.1) Non-honest CA validation

The CAs that validates the non-honest CA will execute identical checks to the initial honesty checks and add the result in the evidence blockchain. Following are the pieces of evidence that need to be stored in the evidence blockchain:

- CN = Common name of the non-honest CA
- Original PKQuery request that the client sent to the CA
- H(Subject) = The faulty record
- R = Original message that the non-honest CA sent to the end user system
- FL = Result of the forward lookup

Upon completing the number of necessary honesty checks, a revocation request will be sent to the memory pool to invalidate the public key of the CA. The invalidation request is handled similar to the certificate issuance, but with an empty public key for the non-honest CA. Once a block is added into the certificate blockchain for the CA, the public key of the non-honest CA will be invalid.

4.7.2) Revocation of Regular Public Keys

In the ConsensusPKI, there is only one valid public key for a subject at any given moment. In any case, issuing a new certificate with a new public key will automatically invalidate the old public key and the certificate. Additionally, the certificate issuance structure of the ConsensusPKI serves the purpose of requesting to update the public key with an empty value. In this case, the service owner must send a CVR to a CA with an empty public key. The process is identical to the certificate issuance but will result in invalidation of the compromised certificate.

Chapter 5) Validation

In this chapter, we will analyze the underlying structures and implementation aspects of the ConsensusPKI ecosystem. The analysis will be split into two sections. The first part is the analysis of the data model and the concept implementation. In the second part, the formal validation of the underlying PKQuery protocol will be discussed. The analyses in the chapter assume two separate adversaries coexists and would attack the sub-elements of the ConsensusPKI. The first one is a Dolev-Yao adversary [54] that controls the part of the network of the client, the subject server (web server), or CA infrastructure where he/she can create, intercept, update, delete exchanged messages between the parties. The second adversary is a malicious CA node that would like to create a fake certificate for a subject. The chapter is constructed as follows. First, in Section 5.1, we will discuss the data model and the implementation aspects of the ConsensusPKI including the message broadcasting and memory pool management among the CAs, and the requirements for the cryptographic primitives and the hash functions. In Section 5.2, We will discuss the initialization of the ConsensusPKI ecosystem. In Section 5.3, the implemented prototype will be discussed. Finally, in Section 5.4, we will discuss the formal validation of the PKQuery protocol executed using the tamarin prover [55].

5.1) Implementation Aspects

This section describes the implementation aspects of the ConsensusPKI. Examples in the section use SQL databases as a datastore to give a clear view of the data structure of the ConsensusPKI. Implementors can use any data storage, such as document or in-memory databases, and it is assumed that the datastore is protected against local attacks, and the only mechanism to update the local data stores at the end user systems is the Fetch mechanism discussed in section 4.5) Anchoring of Public keys of CAs on End-user systems. The CAs will maintain the datastore in the CA structure through the protocols of the ConsensusPKI.

Correctly implementing ConsensusPKI requires fully understating the indication and interpretation mechanism of ConsensusPKI. Following foundational elements are required on end-user systems for PKQuery to function correctly:

- We assume that the ConsensusPKI ecosystem is initialized and many CAs are provisioned
- Initial CA root blockchain, CA certificate blockchain, and CA subject databases are distributed to the client machines, and client machines have local access to the public keys of the CAs using the local datastore

Following sections, we will first summarize the data structure to explain the intended implementation.

5.1.1) Considerations on CA Datastore on end-user systems

In this section, we will discuss the building blocks of the data structure of the ConsensusPKI to give a clear view of the data structure on the end user systems. Following pseudocodes are used during the prototype implementation to create the data structures. We will use the pseudocodes to explain the intended implementation.

The script in Figure 18 is pseudocode to create the `ca_rootblockchain` table in a SQL database:

```

CREATE TABLE `ca_rootblockchain` (
  `Height` INT NOT NULL,
  `PreviousBlockHeader` VARCHAR NOT NULL,
  `Link` VARCHAR NOT NULL,
  `BlockHeader` VARCHAR NOT NULL,
  `MerkleRoot` VARCHAR NOT NULL,
  `Nonce` VARCHAR NOT NULL,
  `BlockTimestamp` TIMESTAMP NOT NULL,
  INDEX `Height` (`Height`),
  INDEX `PreviousBlockHeader` (`PreviousBlockHeader`),
  INDEX `BlockHeader` (`BlockHeader`),
  INDEX `MerkleRoot` (`MerkleRoot`));

```

Figure 18: Pseudocode to create the ca_rootblockchain table

Although the ConsensusPKI requires ca_certificateblockchains to be a separate blockchain for every year based on expiry, it is possible to keep the data in a single database table. For simplicity, the implemented prototype used a single table. The year column in the table indicates the ca_certificateblockchain that the related record belongs to. In the prototype, all columns in the records are used to calculate the block headers.

The script in Figure 19 is pseudocode to create the ca_certificateblockchain table in a SQL database:

```

CREATE TABLE `ca_certificateblockchain` (
  `Year` INT NOT NULL,
  `Height` INT NOT NULL,
  `PreviousBlockHeader` VARCHAR NOT NULL,
  `BlockHeader` VARCHAR NOT NULL,
  `CACertificate` LONGTEXT NOT NULL,
  `MerkleRoot` VARCHAR NOT NULL,
  `Nonce` INT NOT NULL,
  `BlockTimestamp` TIMESTAMP NOT NULL,
  INDEX `Height` (`Height`),
  INDEX `PreviousBlockHeader` (`PreviousBlockHeader`),
  INDEX `BlockHeader` (`BlockHeader`),
  INDEX `MerkleRoot` (`MerkleRoot`),
  INDEX `Year` (`Year`);

```

Figure 19: Pseudocode to create the ca_certificateblockchain table

Figure 20 illustrates an example dataset on ca_certificatablockchain from the implemented prototype. As an example, the block year 2018 with height 6 has no relationship with the first record of the year 2019 since they are different blockchains and in the year 2018, it is still possible to add blocks to the blockchain of 2018. Furthermore, blocks in the same year have a relationship with each other with the blockheader = previousblockheader equation. As an example, the previousblockheader of 2019 with height 3 is same with the blockheader of 2019 with height 2 (please note: to save time during the data generation for the prototype, the PoW is not applied to calculate the block headers and random nonce and merkle root values are used).

Year	Height	PreviousBlockHeader	BlockHeader	CACertificate	MerkleRoot	Nonce	BlockTimestamp
2,018	2	95aee8a2754c5510bd48...	f1ed05a8c83dc63...	{V:"4",...	929a09886475aa8...	456,...	2018-11-25 1...
2,018	3	f1ed05a8c83dc6321bdd0...	7bb757b53ad6d5...	{V:"4",...	8b9cdf2c4ebf4d86...	563,...	2018-11-25 1...
2,018	4	7bb757b53ad6d5d11d06f...	f5fc85aed91e69b...	{V:"4",...	cfbc050163cf3434...	15,4...	2018-11-25 1...
2,019	2	5a542e12200be0618f7cf...	a6d9fc7fc200275...	{V:"4",...	701e634bf94845b...	595,...	2018-11-25 1...
2,018	5	f5fc85aed91e69bac07fd0...	271641248fab3e...	{V:"4",...	336404bddd14968...	352,...	2018-11-25 1...
2,018	1	7cd760b107b8fce7278f6...	95aee8a2754c55...	{V:"4",...	5e00a26adb897f0...	895,...	2018-11-25 1...
2,019	3	a6d9fc7fc200275c4138d...	f2def4a223fb009...	{V:"4",...	43eea10034c64e2...	595,...	2018-11-25 1...
2,019	4	f2def4a223fb009fe88002...	7856145c77640b...	{V:"4",...	b49f0f850e8a65b2...	595,...	2018-11-25 1...
2,019	1	0ca561624ac48c8fff4891...	5a542e12200be0...	{V:"4",...	f7058158bd9cd3b9...	595,...	2018-11-25 1...
2,018	6	271641248fab3e2d2146e...	c9657bd49f6cb04...	{V:"4",...	755f3b1e093b218...	595,...	2018-11-25 1...

Figure 20: Example dataset on ca_certificateblockchain table

The script in Figure 21 is pseudocode to create the ca_subjects table in a SQL database:

```
CREATE TABLE `ca_subjects` (
  `YEAR` INT NOT NULL,
  `HEIGHT` INT NOT NULL,
  `SUBJECT` VARCHAR NOT NULL,
  INDEX `YEAR` (`YEAR`),
  INDEX `HEIGHT` (`HEIGHT`),
  INDEX `SUBJECT` (`SUBJECT`),
);
```

Figure 21: Pseudocode to create the ca_subjects table

The subject column in the table stores the **cleartext** value of the CA subject such as node1.consensuspki.org. The subject value will be used in the PKQuery request to determine the CA server that the client sends the query request.

The script in Figure 22 is a pseudocode to create the vw_ca database view in a SQL database:

```
CREATE VIEW `vw_ca` AS
select ca_subjects.SUBJECT,ca_certificateblockchain.CACertificate from ca_certificateblockchain,ca_subjects
where ca_certificateblockchain.year = ca_subjects.year and ca_certificateblockchain.Height=ca_subjects.height
order by ca_certificateblockchain.year desc,ca_certificateblockchain.height desc;
```

Figure 22: Pseudocode to create the vw_ca view

The view vw_ca lists all CA records descending from the latest records. For every subject, the very first retrieved record is the valid record for the subject CA, and other possible records for the CA is considered **not valid**. This constrained can be built into the SQL view or can be handled by the client application. In the prototype PKQuery client application, we chose to implement the constraint in the client application to keep the data model simple. Figure 23 illustrates an example dataset used in the prototype. The illustrated dataset merges the two blockchains for the year 2019 and 2018 and orders the recordset so that the valid records for a subject comes in the record set as the first record.

YEAR	HEIGHT	SUBJECT	CACertificate
2,019	4	node10.consensuspki.org	{V:"4",H:"SHA256",VT:"2019-12-27T13:28:06....
2,019	3	node9.consensuspki.org	{V:"4",H:"SHA256",VT:"2019-12-21T13:28:06....
2,019	2	node8.consensuspki.org	{V:"4",H:"SHA256",VT:"2019-11-28T13:28:06....
2,019	1	node7.consensuspki.org	{V:"4",H:"SHA256",VT:"2019-10-27T13:28:06....
2,018	6	node6.consensuspki.org	{V:"4",H:"SHA256",VT:"2018-12-29T13:28:06....
2,018	5	node5.consensuspki.org	{V:"4",H:"SHA256",VT:"2018-12-27T13:28:06....
2,018	4	node4.consensuspki.org	{V:"4",H:"SHA256",VT:"2018-12-25T13:28:06....
2,018	3	node3.consensuspki.org	{V:"4",H:"SHA256",VT:"2018-12-24T13:28:06....
2,018	2	node2.consensuspki.org	{V:"4",H:"SHA256",VT:"2018-12-23T13:28:06....
2,018	1	node1.consensuspki.org	{V:"4",H:"SHA256",VT:"2018-12-21T13:28:06....

Figure 23: Example dataset on vw_ca view including the year and height of the blocks

5.1.2) Considerations on the CA Certificate file structure

For simplicity, the developed prototype used JavaScript Object Notation (JSON) [56] data structure as a CA certificate as illustrated in Figure 24. Although not required, we used “.jcr” extension for the certificate files. The field explanations in the .jcr files for the CA certificates are as following:

- V = Version (version 4 is used as an example for the ConsensusPKI certificates)
- H = Hash algorithm to be used in the block header calculation
- VT = Valid through (the validity of the public key in Coordinated Universal Time (UTC) format)
- CN = Common name of the CA node (DNS name)
- SP = Service port (The port number PKQuery requests must be sent. The field is not required if a port number would be standardized for PKQuery protocol)
- PublicKey = the public key of the CN in pem format (base64)

```
{
  "V": "4",
  "H": "SHA256",
  "VT": "2018-12-21T13:28:06.419Z",
  "CN": "node1.consensuspki.org",
  "SP": 5501,
  "PublicKey": "-----BEGIN PUBLIC KEY-----
\nMIIIBjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAOoOMBpLUIwFuHtSJLyIv\niRT1L6ISg9Yp2wqAVpdgIVpi
GsRsKUd8RDHryiif/3cGb19c6ht0q1HfkgLNGXJR\n6f2oPL8D4Ez+rwGQpS85XF7qMzWeD+pF++I7TJxewUVGstWLi3
VlkU7IEH7sIl8i\nXTDNVoOJrwn5dc4j0cu+fi7i8OJKpo2KiMgPuGoWVnQ/J45XqtIVLA65RRGYgzy\nn2j9kfxVD3Y0n6
dJil1i2L5LC/u5Vt175nYLSlaQzTCNBLlwCoGP0Fusf6UWJU7g\nnpsiysiZJ8iDJGgj+aeFHC4orewy6geTiVGxWzoxhQij3
m1UJXiUSTpzSZ0pHs+FW\nQwIDAQAB\n-----END PUBLIC KEY-----"
}
```

Figure 24: An example CA Certificate used in the prototype

It is intended that block header generation to add a record to the `ca_certificateblockchain` table is executed using the **complete** certificate file. The data integrity in the `ca_certificateblockchain` is only guaranteed if the **complete** certificate file is used during the block header calculation.

The above-explained data structure regulates the equal treatment of all CAs by the end user systems. Compare to X.509; there is no trust path or hierarchy among the CAs. CAs are not trusted even though their public keys are in the `ca_certificateblockchain`. In the PKQuery protocol, the responses of the CAs are compared with each other, and PBFT helps to indicate a faulty CA.

5.1.3) Considerations on Datastore on CA infrastructure for Regular Certificates

The core data representing all regular certificates are maintained in the CA infrastructure. The data is the core element of the ecosystem, and proper maintenance of the data requires a good understanding of the structure. As the CAs will manage the certification process, they require the following data structures to be deployed under their infrastructure.

Since the `rootblockchain` and `ca_rootblockchain` tables serve the same purpose but for a different type of certificates, their data structure is identical. Furthermore, the `subjects` and `ca_subjects` tables serve also the same data structure. However the `subjects` table for the regular certificates can hold multiple records for each certificate, and `H(subject)` is kept

in the subjects table instead of the cleartext subject value. Figure 25 illustrates an example dataset in the subjects table.

consensuspki.subjects: 28,775,523 rows total (approximately), limited to 1,000

YEAR	HEIGHT	SUBJECT
2,018	1	f7c5c741f4983e1f86cd0ec1a0a562e9737fe1c47f9a2fa6ee476d8df493edaf
2,018	1	44b884ff0203e94af08ce311d426f66dd6620f9a6412c10f477da4efcf5fe3aa
2,018	1	aaba252bf857e5f8f9bc38fa5fee32452b881b25549e1e97aa005c79403aa85b
2,018	1	2e922d05a789f9166a16a3c48d1fe1fc9664acaaeb620b0e5d39d0d5681a0c85
2,018	1	28b2e4060862ff3154a8137b2181a9d6aaf1ef15c7f9614b3c19580f7675e5cc

Figure 25: An example dataset in the subjects table

Following script is pseudocode to create the certificateblockchain table in a SQL database:

```
CREATE TABLE `certificateblockchain` (
  `Year` INT NOT NULL,
  `Height` INT NOT NULL,
  `PreviousBlockHeader` VARCHAR NOT NULL,
  `BlockHeader` VARCHAR NOT NULL,
  `MerkleRoot` VARCHAR NOT NULL,
  `Nonce` INT NOT NULL,
  `BlockTimestamp` TIMESTAMP NOT NULL,
  INDEX `Height` (`Height`),
  INDEX `PreviousBlockHeader` (`PreviousBlockHeader`),
  INDEX `BlockHeader` (`BlockHeader`),
  INDEX `MerkleRoot` (`MerkleRoot`),
  INDEX `Year` (`Year`));
```

Figure 26: Pseudocode to create the certificateblockchain table

The certificateblockchain and ca_certificateblockchain tables are very similar tables, and the only difference is that the certificateblockchain table has no certificate column.

Comparison of Figure 25 and Figure 23 visualizes the difference in the structure of the data that is distributed to the end user systems and the data kept in a CA infrastructure.

In the PKQuery response message the CA must send the following response message R to the client:

$$R = h(h(\text{nonce}) || h(h(\text{subject}) || \text{MerkleRoot}))$$

In the prototype, the following database view vw_response is implemented as a helper to quickly retrieve the h(h(subject) || MerkleRoot) value using the h(subject) as search text. Although, each database system might handle such construction differently, and the implementers might choose a different structure to construct the response message R, the code in Figure 27 can be used as a reference model to construct the h(h(subject) || MerkleRoot) part of the response message.

```
CREATE VIEW `vw_response` AS
select subjects.SUBJECT,
upper(sha2(concat(hex(subjects.SUBJECT), hex(certificateblockchain.MerkleRoot)),256)) Hash
from subjects, certificateblockchain
where subjects.YEAR = certificateblockchain.Year
and subjects.HEIGHT = certificateblockchain.Height
order by certificateblockchain.year desc, certificateblockchain.height desc
```

Figure 27: Pseudocode to create the vw_response in MariaDB

There will be two additional tables that CAs need to have to keep collected evidence temporarily until the corresponding block is added into the related certificate blockchain. Although the data structure is the same, we prefer to separate the tables as we implemented for all other tables. The script in Figure 28 is pseudocode to create the `ca_evidenceblockchain` table in a SQL database.

```
CREATE TABLE `ca_evidenceblockchain` (
  `EvidenceBlockChainID` VARCHAR NOT NULL,
  `Height` INT NOT NULL,
  `PreviousBlockHeader` VARCHAR NOT NULL,
  `BlockHeader` VARCHAR NOT NULL,
  `Evidence` LONGTEXT NOT NULL,
  `Nonce` INT NOT NULL,
  `BlockTimestamp` TIMESTAMP NOT NULL,
  INDEX `Height` (`Height`),
  INDEX `PreviousBlockHeader` (`PreviousBlockHeader`),
  INDEX `BlockHeader` (`BlockHeader`),
  INDEX `EvidenceBlockChainID` (`EvidenceBlockChainID`)
);
```

Figure 28: Pseudocode to create the `ca_evidenceblockchain` table

An important aspect of the evidenceblockchains is that they hold an EvidenceBlockChainID on each record so that the evidenceblockchains can be distinguished between each other. The EvidenceBlockChainID field is the hash digest of the C value in the initial CVR file. The evidence field holds real data exchanged between the APKME of the subject and the CA. As an additional proof that the CA executed the challenge-response veridiction, the CA that executes the verification must calculate the block header by using PoW.

5.1.4) Considerations on Regular Certificate file structure

In the ConsensusPKI regular certificates are not kept in the certificate blockchain. Every block in a certificatelockchain represents 1024 certificates that form 2048 leaves Merkle tree together with the root of the Evidence Merkle tree. For a certificate properly to be indicated on the client systems, the Merkle proof of its existence in the certificateblockchain is required as a separate part in the certificate file. The client expects the following response message $h(h(\text{nonce}) \parallel h(h(\text{subject}) \parallel \text{MerkleRoot}))$ from the PKQuery request. The client must construct the $h(h(\text{subject}) \parallel \text{MerkleRoot})$ part of the message using the received certificate file from the server of the subject during the TLS handshake. The data structure of the regular certificates must enable such construction. Figure 29 illustrates an example regular certificate that contains two part. The first part C is similar to CA certificate and holds basic records including the public key to identify the certificate. The second part P is the Merkle proof of the certificate. Using the data in the C and the P, the client on the end-user system can construct the $h(h(\text{subject}) \parallel \text{MerkleRoot})$ part of the expected PKQuery response message. In the PKQuery prototype, the python code in Figure 30 is used to construct the Merkle root from the Merkle proof in the P part of the certificate file. The targethash in the function is the hash value of the data in the C field. The very first record in the array P is the Merkle root of evidence Merkle tree of the certificate, and it is always in the right side of its sibling, the hash value of the data in the C field.

```
{
  "C":{
    "V":"4",
    "H":"SHA256",
    "VT":"2020-12-25T13:28:06.419Z",
    "CN":"www.nos.nl",
    "PublicKey":"-----BEGIN PUBLIC KEY-----
\\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAE3xNcYiyA37crJ8Ql6UGT\\nU4k9830a+QAJ59+TXa/F86
9HepmGK4MwHG17KRGXUdtYKUYldvAXTwWftZCSTR\\nVPb0kxQEJUDJ/+6lQBuoQvsEtv7tvUO70VoHKT0Qp
Lga6CmiJpXcwWFFw/iu4xd\\nny1TZPLK9zSZSN2WqCtzCTG0RxE/ZJyNsOF6VT0uheoBm4afLEqBc2plklw+tfKZF\\nO3
e3bC/HaQZwTmC3g93Q0haXg8R+TFMnf7jFkqwfmlXjRPam+bV3CASAmgfbpm\\n\\nix2ADoKmHea9b5vvrkhiOV4B
35YFzJ2/tIXMOA//I6JaZgG1MPcZ5VQYN8ke4AbD\\n3QIDAQAB\\n-----END PUBLIC KEY-----\\n"
  },
  "P":{
    { "right":"b5e81dbb23f50a6cae431e75e2d2664a80ba74be3d58487f469cb434c5ec6d3b"},
    { "left":"55488bf646a053126752b618eac1bec9b4e174528b811d1ab8ae8a229e85c7e8"},
    { "right":"133130a5fba728aabe14a0d1621a951180caf83c056b4f0c836464f27c3608bb"},
    { "left":"2439565925345a7e5939bebdd92922eefcd449c5f0c57e5010a6677f56497921"},
    { "right":"c44e6bfd9d0be0eae2254035bce9a7e09040e0ae079230f6b8903dadfb52fd23"},
    { "left":"764f390e052d369661569c088a55c0fb3c59916adb0b981777d9a7f88bf58c7c"},
    { "left":"764f390e127ea6d442f1975519b880f9b7a61af3c191277166ff1e11f2e7aaef"},
    { "left":"02ca69eaadc8f43e6288d8fc83f79692e86b2ea297871d274bbd5e40fbb2607e"},
    { "left":"d4e0fcc4b665c24789288a0af8a5e85dc5bfc6a50960c8098cbddb8e758ed90"},
    { "left":"fa992cb5aae43fe6751dcbcb63b316c4f56f421fa32389db557e281975cee01c"},
    { "right":"84bfd6ffbfef2dfd7e56e725161e13009867b743dd645c06ba1ce56d43fef13e"}
  ]
}
```

Figure 29: An example Regular Certificate used in the prototype

```
def calculate_hash(binaryValueToHash):
    sha256_digest = hashlib.sha256(binaryValueToHash).hexdigest();
    return sha256_digest;

def get_MerkleRootFromProof(targetHash,proof):
    s1 = json.dumps(proof)
    myInitHash = targetHash;
    myProofs = json.loads(s1)

    initilized = False;
    myFinalHash = b'';
    for singleProof in myProofs:
        mySibling = '';
        try:
            mySibling = bytearray.fromhex(singleProof['right']);
            if not initilized:
                myFinalHash = calculate_hash(bytearray.fromhex(myInitHash) + mySibling);
                initilized = True;
            else:
                myFinalHash = calculate_hash(bytearray.fromhex(myFinalHash) + mySibling);
        except KeyError:
            mySibling = bytearray.fromhex(singleProof['left']);
            if not initilized:
                myFinalHash = calculate_hash(mySibling + bytearray.fromhex(myInitHash));
                initilized = True;
            else:
                myFinalHash = calculate_hash(mySibling + bytearray.fromhex(myFinalHash));
    return myFinalHash;
```

Figure 30: The python code to construct the Merkle root from a Merkle Proof

5.1.5) Location and data of a Certificate in the Certificate Merkle tree

Using the example certificate in Figure 29, we will explain how the related data will be stored in the certificate Merkle tree when a certificate is issued. In Figure 31, the data

identified as C is the hash value of the data in the C field of the certificate, and the data identified as E is the very first record of the P array which is also the root of the evidence Merkle tree. The node n is on the left side of the C, and the node m is on the right side of the C. A certificate file in ConsensusPKI ecosystem must carry all nodes in the Merkle tree, so that it can construct the Merkle root represented in green.

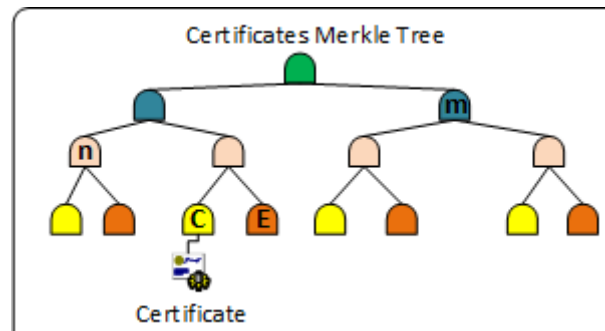


Figure 31: An example certificate Merkle tree

5.1.6) ConsensusPKI protocols in TCP stack

Based on the OSI model, the ConsensusPKI ecosystem is an application layer protocol suite [57]. Every sub-element of the ConsensusPKI have different requirements and satisfying those requirements need different approaches for the ecosystem to work as a whole.

First, the PKQuery protocol will be used by all end-user systems to validate a particular certificate. Minimizing the load on the CA infrastructure leads us to have certain design choices. Similar to the DNS ecosystem, the queries will be executed using a connectionless manner over datagrams through UDP messages. Since there is no connection; there is no guarantee that there will be a response to the request. The intended implementation of the ConsensusPKI requires, resending the query using additional randomly selected CA to meet the requirement of minimum four CA responses for interpreting a received certificate correctly. Another reason to choose the UDP datagrams instead of TCP messages is to reduce the response time of the PKQuery messages. In the implemented prototype we used UDP ports defined in the SP fields of the certificate, however, upon standardization, this can be a standard port such as UDP port 55.

Second, the APKME protocol is a protocol that will be used during certificate issuance. The requirements of APKME is different from PKQuery. In APKME, the CA servers must connect and exchange messages to validate the public key of the subject. It is required to have a guarantee on the exchanged messages; thereby the UDP datagrams cannot be used. Although the implemented prototype does not cover APKME protocol, it is intended to implement the APKME on TCP using the same standard port as the UDP port of PKQuery.

Third, the broadcast messages between the CA nodes is another aspect of the protocol suites of the ConsensusPKI that requires proper design choices to have a correctly functioning ecosystem. Since the ConsensusPKI ecosystem is a permissioned blockchain network that requires pre-provisioning of the nodes, the delivery of the broadcast messages must be guaranteed for all CAs to have the same memory pool and dataset at any given moment. Similar to bitcoin network all broadcast messages intended to be delivered through TCP connection. Achieving proper delivery of the messages requires heartbeat check to be executed among the CA nodes so that when a node becomes offline, the network would be notified. Another aspect of the broadcast messages is the authenticity of

the broadcasted messages. It is intended to have signatures of the sender CA on the broadcasted messages so that the receiving CA node can verify the authenticity of the message.

Fourth, the speed of the signing algorithms is a very crucial aspect for the convenience of the end users. Minimizing the time spend on PKQuery thereby requires using the fastest signing algorithm that meets the security requirements of the given moment. In the concept implementation, two separate experiments are executed. In the first experiment, RSA signatures are used with 2048 bits key size. In the second experiment, we used ECDSA with a sepc192r1 curve with 192-bit key size. The experiments showed that the total elapsed time is increased by 300% in ECDSA. The measured difference is significant, and as a result, the duration of the validation process becomes noticeable by the humans.

Fifth, ConsensusPKI uses asymmetric encryption (RSA) in several places. As we use asymmetric encryption, we are forced to consider the message sizes in the protocols. In the query requests of the PKQuery and Fetch protocols, and in the APKME protocol of the ConsensusPKI, the asymmetric encryption is used to guarantee the secrecy of the messaging. As we used RSA asymmetric encryption in our concept implementation, the total message size cannot be more than the key size. If the message requires padding, the allowed message size would be even less than the RSA key size.

Finally, in the core of the ConsensusPKI sits hash functions. We assume that the ecosystem uses ideal hash functions, also called random oracle [58]. Unfortunately, in practice, the implementors must consider that the used hash function can become unsafe to use. The found collisions in MD5, and SHA1 hash functions are a recent example of such circumstance. Besides security, there are three additional impacts of hash functions to the ConsensusPKI ecosystem. The first one, the speed of the hash function has an impact on the execution time of the PKQuery protocol. The second aspect is the digest size of the hash function changes the required data storage on the CA infrastructure. The final aspect is the digest size changes the message size of the protocols, changing the bandwidth requirements. The concept implementation used SHA256 hash function as a standard. However, the protocol suite is not bound to a fixed hash function, and when required the certificates indicate which hash function to use on particular protocol of the ConsensusPKI.

5.1.7) An Example Execution Flow of PKQuery

The intended implementation of the PKQuery protocol has four states. The following are the brief explanation of each state of the execution flow:

- State1: The client loads the locally available CA certificates into its memory. In the concept implementation, we use the vw_ca database view to load the CA certificates into the memory of the client
- State2: The client requests the certificate from the subject server and sends four parallel PKQueries to randomly selected CAs. Client stores the randomly selected nonce used in the PKQueries into its memory
- State3: The client receives the certificate from the server of the subject, and response messages from the CA servers {R1, R2, R3, R4}
- State4: The client checks the signatures on the response messages, and constructs the $R' = h(\text{nonce} + h(h(\text{subject}) + \text{merkleroot}))$ using the certificate and nonce stored in the State2. The connection is considered secure if $R' = R1 = R2 = R3 = R4$.

Please refer to section 4.6) Indication and interpretation on the client side: The public key query protocol for full specification and usage of fault tolerance.

Figure 32 illustrates execution steps and states of PKQuery. The Fetch algorithm of the ConsensusPKI is similar to the PKQuery, but the client interacts directly with the CA systems to update its local certificate database.

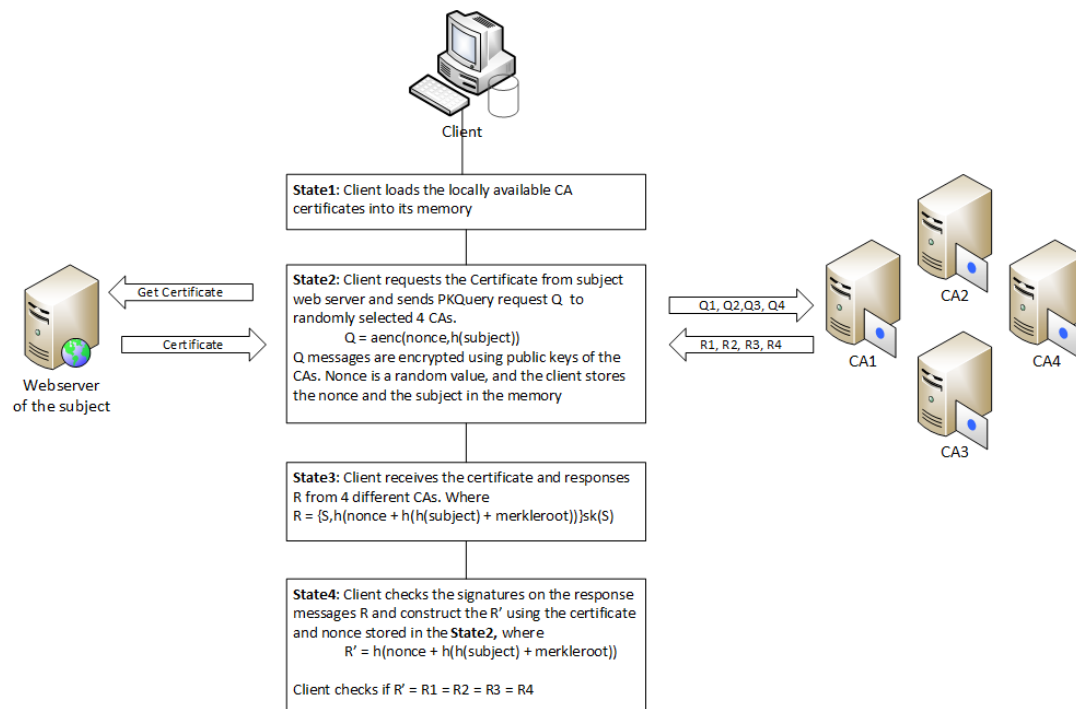


Figure 32: Intended execution steps and the States of PKQuery

5.1.8) Block Header Calculation and Consistency Check on Blockchain Blocks

5.1.8.1) Block level validation

In this section, we will analyze how the CAs should create block headers for both regular and CA certificates. The very first requirement to calculate the blockheader is to form the certificate Merkle tree. All of the CAs can participate to block generation process, and they must check the evidence of all verifications in order to add their certificates into the related blockchain. In the process, the CA must choose 1024 verified certificate request from the memory pool (most likely the verifying CA will prioritize its certificate requests over other requests) and must form a Merkle tree to calculate the Merkle root. Once the Merkle root is calculated, the blockheader can be calculated. Calculating the blockheader requires concatenated sorted subject data as a data block. The pseudocode in Figure 33 is a SQL function to retrieve the subject data as a single value to be used in a single blockchain block.


```

CREATE FUNCTION `GetOrderedSubjects`(`p_year` INT, `p_height` INT) RETURNS longtext
DETERMINISTIC
BEGIN
DECLARE OrderedSubjects LONGTEXT;
SET OrderedSubjects = "";
Select GROUP_CONCAT(s.SUBJECT ORDER BY s.SUBJECT SEPARATOR " ") into OrderedSubjects
from certificateblockchain c, subjects s
WHERE s.YEAR = c.Year AND c.Height = s.HEIGHT
and c.Year = p_year and c.Height = p_height
GROUP BY c.Year, c.Height ORDER BY c.Year,c.Height;
RETURN OrderedSubjects;
END

```

Figure 33: Pseudocode to retrieve Ordered Subjects in a single data block

For simplicity, in the concept implementation, we preferred to calculate the block headers in the MariaDB database itself. Implementers can choose to calculate the blockheaders on the application side. However, the SQL code in Figure 34 can be used as a baseline to calculate the blockheader. Furthermore, the same SQL code can also be used to check the integrity of the blocks as given in Figure 35. The pseudocode in Figure 35 should return **no records** in the entirely consistent dataset.

```

Select
sha2(concat(hex(PreviousBlockHeader),hex(MerkleRoot),Nonce,BlockTimestamp,Year,Height,GetOrderedSubject
s(Year,Height)),256)
as blockheader from certificateblockchain

```

Figure 34: Pseudocode to calculate the Blockheader

```

select * from certificateblockchain
where
sha2(concat(hex(PreviousBlockHeader),hex(MerkleRoot),Nonce,BlockTimestamp,Year,Height,GetOrderedSubject
s(Year,Height)),256)
<> blockheader

```

Figure 35: Block consistency check on certificate blockchain

Similar to above pseudocodes, the pseudocodes in Figure 36 can be used to calculate the blockheaders in the CA certificateblockchains. The only difference is that the pseudocodes include the CACertificate to calculate the blockheader.

```

CREATE FUNCTION `GetOrderedCASubjects`(`p_year` INT, `p_height` INT) RETURNS longtext
DETERMINISTIC
BEGIN
DECLARE OrderedCASubjects LONGTEXT;
SET OrderedCASubjects = "";
Select GROUP_CONCAT(s.SUBJECT ORDER BY s.SUBJECT SEPARATOR " ") into OrderedCASubjects
from ca_certificateblockchain c, ca_subjects s
WHERE s.YEAR = c.Year AND c.Height = s.HEIGHT
and c.Year = p_year and c.Height = p_height
GROUP BY c.Year, c.Height ORDER BY c.Year,c.Height;
RETURN OrderedCASubjects;
END

Select
SHA2(concat(hex(PreviousBlockHeader),hex(SHA2(CACertificate,256)),hex(MerkleRoot),Nonce,BlockTimestamp,Y
ear,Height,GetOrderedCASubjects(Year,Height)),256)as blockheader from ca_certificateblockchain

Select * from ca_certificateblockchain where
SHA2(concat(hex(PreviousBlockHeader),hex(SHA2(CACertificate,256)),hex(MerkleRoot),Nonce,BlockTimestamp,Y
ear,Height,GetOrderedCASubjects(Year,Height)),256) <> blockheader

```

Figure 36: Blockheader calculation and integrity check pseudocodes for CA_certificateblockchain

Above explained method could also be used to calculate and validate the blockheaders of the root blockchains. The only difference is the calculation of the Merkle root for the root blockchains.

5.1.8.1) Cross-validation among the blocks

The block level validation checks whether the stored blockheader represents the data in the database, but it does not validate whether the blockchain is intact or not. After block level validation, cross-validation must be executed to validate the integrity of the blockchain. The pseudocode in Figure 37 should return **no records** in the entirely consistent dataset.

```
Select * from certificateblockchain a, certificateblockchain b
where a.BlockHeader <> b.PreviousBlockHeader
and a.Year = b.Year
and a.Height = b.Height -1
```

Figure 37: Pseudocode for cross-validation among certificate blockchains

5.1.9) Example PoW algorithm to calculate a specific blockheader

The pseudocode in Figure 38 calculates a blockheader with three leading zeros '000'. The procedure accepts year and height as an input parameter and gives an output of nonce, block timestamp, and blockheader. For the simulations in the concept implementation, we did not use PoW to save time. However, in the real deployment, CAs must execute PoW to reach consensus. The parameters of the PoW is not defined, and the choice is left to CAs upon implementation.

```
CREATE PROCEDURE `PoWCertificateBlockchain` (
IN `p_year` INT,
IN `p_height` INT,
OUT `p_Nonce` INT,
OUT `p_BlockTimestamp` TIMESTAMP,
OUT `p_BlockHeader` VARCHAR
)
NOT DETERMINISTIC
BEGIN
DECLARE myNonce INT;
DECLARE myBlockTimestamp TIMESTAMP;
DECLARE myBlockHeader varchar;

SET myNonce = 0;

REPEAT

SET myBlockTimestamp = current_timestamp;
SET myNonce = myNonce + 1;
select
sha2(concat(hex(PreviousBlockHeader),hex(MerkleRoot)),myNonce,myBlockTimestamp,Year,Height,GetOrderedS
ubjects(Year,Height)),256) into myBlockHeader
from certificateblockchain
where year = p_year
and height = p_height;
UNTIL substr(myBlockHeader,1,3) = '000'
END REPEAT;
set p_Nonce = myNonce;
set p_BlockTimestamp = myBlockTimestamp;
set p_BlockHeader = myBlockHeader;
END
```

Figure 38: Pseudocode to calculate a blockheader starts with tree leading zeros '000'

Figure 39 is an example execution of the PoW pseudocode for the 5th block of the certificate blockchain of 2018. The output blockheader for the dataset has three leading zeros.

<pre>1 CALL `PowCertificateBlockchain`('2018', '5', @nonce, @blocktimestamp, @blockheader); 2 select @nonce, @blocktimestamp, @blockheader;</pre>		
Result #1 (3×1)		
@nonce	@blocktimestamp	@blockheader
4,387	2018-12-08 18:02:32	0001843710997f9eefb7b751f73e11cf062e67fce32177a9df466849553e39d9

Figure 39: Example execution of PoW

5.2) Considerations on the Blockchain Network

In this section, we will discuss the infrastructure of the blockchain network, and possible elements that blockchain nodes might choose to use.

5.2.1) Initialization of the Blockchain Network

The certificate issuance in the ConsensusPKI is a very rigorous process. It is required to have at least 16 CAs to issue a regular certificate. Provisioning a new CA requires 64 individual CA to verify the public key of the new CA. Further, PKQuery algorithm requires a minimum 4 CAs to respond to the queries. This rigorous ecosystem, makes it challenging to initialize the blockchain network. Following method could be followed to initialize the blockchain network if the initial number of nodes does not satisfy the requirements:

- If 64 separate CAs come together to initialize the network, start the blockchain network with 64 initial nodes
- If the number of initial members is less than 64 but more than 16, the network would be initialized with 16 CAs, and CA certificate issuance would require verification of all CAs in the network until the number of CAs would reach 64. During this transition period, regular certificates can be issued
- If the number of initial nodes is less than 16, the network would be initialized with all initial CAs, and CA certificate issuance would require all CAs to verify the certificate verification requests. Until the number of CAs would reach 16, no regular certificate would be issued
- The Blockchain network cannot be initialized if the number of initial nodes is less than 4

Upon satisfying the required number of 64 CAs, all CAs participated in the transition period of the network initialization must renew their certificates, in order to formally satisfy the 64 verifications of the CA provisioning.

5.2.2) CA infrastructure

Although the ConsensusPKI prefers all CAs to participate in all processes of the ecosystem, it is up to CAs to decide how they will organize internally to meet the requirements of the sub-protocols of the ecosystem. When we refer to a CA, it is a blockchain network node, and a CA entity can host several blockchain nodes, that they can divide some tasks among the nodes that they manage. In any case, the data that a blockchain node holds is the responsibility of that specific node since there is no hierarchy among the CA nodes. In this chapter, we will discuss the critical aspects of the CA infrastructure.

In ConsensusPKI, the certificates are not signed, and the public keys of the CAs are used only for message encryption and authentication. When a CA starts sending faulty messages, the processes of the ConsensusPKI detect the non-honest behavior, and the public key of the CA will be revoked by the blockchain network, by creating a new empty CA certificate for the non-honest CA subject. As a result, a compromised CA private key has only impact on the CA itself.

If an attacker successfully controls 16 CAs at the same time, he/she can create fake evidence and finally a fake certificate. Since there can be only one valid certificate at any moment for a subject, the created fake certificate will **automatically invalidate** the original certificate of the subject. As a result, the service of the real subject will stop working, since the newly created fake certificate is not deployed in the real web server of the subject. Although the execution of such an attack is challenging, it will be straightforward to detect the compromised CAs and revoke their certificate from the Certificate blockchain. The real subject of the fake certificate can invalidate the fake certificate by requesting a new certificate with an empty public key, or with new, or with the original public key. The newly issued certificate will be a different certificate than the original certificate of the subject.

The explained overall ecosystem creates less strict infrastructure for CAs to deploy. There are two very critical components of the CA infrastructure. The first one is the private key of the CA, and the second one is the blockchain datastore. The compromise of one of the elements would cause invalidation of the CA certificate. The decisive aspect of the deployed CA infrastructure is not the private key but the performance.

The concept implementation used a MariaDB SQL database and loaded the private key of the CA in the memory of the server. The implementors can have other approaches to store the data or their public keys such as hardware security modules, as long as their implementation satisfies the network requirements. Although it is not implemented in the proof of concept, it is possible to set a maximum acceptable response time for the PKQuery responses. If the client does not receive a response timely, (as an example, in 50 milliseconds depending on the distance between the client and the CA nodes), it can send the query request to another randomly selected CA. Setting such a limit in the response time will push the CAs to design their infrastructure to meet the expectations. We will discuss the results of the concept implementation in the next section.

Unfortunately, no one knows the exact number of valid certificates currently used on the internet [59]. One of the goals of the ConsensusPKI is to bring transparency on the valid certificates on the internet. Using a copy of a ConsensusPKI database, the number of valid certificates can be retrieved at any given moment. As we do not have the exact statistics, considering the increasing trend to use HTTPS, we assume for our calculations the following:

- There exist one billion valid certificates
- On average, a certificate holds three CN or CAN records
- On average the half of the certificates issued for one year, and the other half for two years
- SHA256 is used as the hash algorithm

Storage requirements on the CA infrastructure

The root blockchain holds one block per year, and the block size is ~ 108 bytes. The core data is on the certificate blockchains, and by using the above-assumed statistics the following can be calculated:

- The average number of blocks for a certificate blockchain: $((5 \times 10^8) + ((5 \times 10^8) \times \frac{1}{2})) \times \frac{1}{2} \sim 366000$ blocks (given, the half of certificates will appear in the database once a year and the other half once every two years)
- An average block keeps 3072 CNs or CANs as an index data, and for every record, subjects table is 40 bytes (32 bytes subject, 4 bytes height, 4 bytes year) and additional four 112 bytes for the block data = $(3072 \times 40 \text{ bytes}) + 112 \sim 120 \text{ kB}$
- The average certificate blockchain size per year $\sim 42 \text{ GB}$

The stored data on CA infrastructure will be maximum just before the block addition to the root blockchain. Even, when it is at maximum, the total storage requirement will be $\sim 126 \text{ GB}$ for the certificate blockchains. In this calculation, the database overhead is not included. The requirement can be slightly different on different database systems since each database engine handles index storages differently. For the MariaDB that we used in the concept implementation, the required storage size for the indexes was $\sim 120\%$ of the real data. Based on these figures in the real-world scenario, the storage requirement can be given as 92.5 GB of storage per year if MariaDB is used in the backend of the CA.

Storage requirements on end-user systems

Currently, there are ~ 1850 intermediate or root certificate authorities around the world [59]. We assume the following for our calculations:

- There exists 2048 CAs
- All CAs change their Public keys every two years
- All CAs have a 2048 bit RSA public key

Using the above assumptions on average the CA certificate blockchains will have 1024 blocks per year, and the total data in a block would be on average 512 bytes. This gives us total certificate blockchain size of 1.5 MB on end-user systems when it is maximum. Including the overhead that the used database systems might bring, the required storage is not a limiting factor for large deployments.

5.2.2.1) Storage of collected evidence

The evidence collected during the certificate issuance or revocation is required for a certificate to be included in a block in the certificate blockchain. After certificate issuance, the collected evidence records can only be used for the audit purposes. ConsensusPKI does not specify when the evidence records can be deleted. From a network security perspective, it would be a good practice to keep the evidence for some time (for example one week) to be able to answer the claims that it might be. Further, the evidence logs do not need to be kept in the live environment of the CA infrastructure and can be moved to another system once a certificate issuance is completed for a certificate request.

5.2.3) Bandwidth requirements

Another important aspect is required bandwidth on CA systems. An incoming PKQuery request is an asymmetrically encrypted message. When RSA encryption is used, even the plaintext is smaller than RSA key size; the expected ciphertext size will be as much

as the RSA key size. For 2048 bit RSA key the incoming message size will be 256 bytes per query. The outgoing message from the CA server to the client has a hash value and a signature. The hash value is for SHA256 is 32 bytes data, and the signature size is the key size of the RSA signing. The total outgoing message size will be 288 bytes per query. Since the required outgoing message size will be more than incoming message size, we will use the outgoing message size for our calculations. Assuming that there exists enough CPU and memory on the system, the maximum number of queries a CA server with 1Gbps bandwidth can handle can be calculated as follows:

The number of queries per second for one node = $(1024 \times 1024 \times 1024) / (288 \times 8) \approx 466000$

Using the above figure we can calculate the total number of requests per second that the CA blockchain network can handle. The following calculation assumes that the queries are randomly distributed as specified in the PKQuery protocol and all CA servers have 1Gbps bandwidth and enough hardware resources.

Total number of queries per second all node combined = $2048 \times 466000 \approx 954$ million

5.3) Prototype for PKQuery protocol of the ConsensusPKI

In this section, we will discuss the results of the simulation tests that we have conducted using the concept implementation of the PKQuery protocol of the ConsensusPKI. The simulation test aims to validate the requirement 7 and 8 of the thesis. We choose to implement the PKQuery protocol because it interprets the certificate file retrieved from the server of the subject and the responses of the CA nodes. The algorithm takes the final decision to accept a connection as secure between the server of the subject and the end user system¹. Since PKQuery protocol must be executed during protocol handshake, the time-to-decision duration will be a decisive factor for our evaluation.

The setup for the simulation: there exists **10 ConsensusPKI CAs** identified as {CA1, CA2, ..., CA10}. The client visits a website of a subject such as www.google.com and receives a certificate file. The received certificate file must be validated by PKQuery protocol in order client to accept the connection as secure. In the setup, two of the CAs (CA9 and CA10) are **compromised**, and they give invalid answers with valid signatures to the PKQuery requests. The CA9 and CA10 generate their responses **after** searching their databases. The CA1 is another malicious actor that enabled extra logging to log the incoming ciphertexts of the messages while giving honest responses to the PKQuery requests. The remaining CAs are entirely honest CAs, and they give valid answers to the PKQuery requests.

Test cases:

- Case 1: Execute PKQuery using the latest certificate of the subject
- Case 2: Execute PKQuery using an old certificate of the subject (The valid through a date not yet passed)
- Case 3: Execute PKQuery using and a certificate that does not belong to the subject
- Case 4: Execute PKQuery using a certificate that does not belong to the subject and the subject never had a certificate

Non-tested cases due to the scope of the protocol:

¹ Source code of the concept implementation and formal proof of the PKQuery on a tamarin prover model can be found under <https://www.github.com/volkankaya/ConsensusPKI>

- An expired certificate is not tested in any of the above cases since it only requires client-side validation, and not the PKQuery (the client-side validation using the VT field of the certificate)

Following are the components of the platform that we have developed our prototype:

- MariaDB version 10.3.10-MariaDB
- Python version 3.6 64 bit
- Python Merkletools library version 1.0.3
- Python Mysqlconnector library version 2.1.6
- Python Pycryptodome library version 3.7.0

The server that hosts the MariaDB database is an HP ProLiant micro server Gen10 with following specs:

- CPU: AMD Opteron X3216 (Dual-Core)
- RAM: 8GB DDR4
- Disk: Single disk OCZ-Vortex4 SSD
- Ethernet: 1Gbps ethernet connection
- OS: Windows 10 Professional

The single application server hosts the PKQuery server application for all of the CAs has the following specs:

- CPU: AMD Phenom II X4 955 (Quad Core)
- RAM: 8GB DDR3
- Disk: Single disk OCZ-Vortex4 SSD
- Ethernet: 1Gbps ethernet connection
- OS: Windows 10 Professional

For the simulation, we have created certificate data using the Alexa top 1 million sites dataset [60]. We have followed the following steps to generate random certificate data for the regular certificates to use in the simulation:

- Step 1: The Alexa data is imported into a temporary table, and an additional 9 million records are added using the original data and following new domains
 - Subject domain = www. + [domain in Alexa record]
 - Subject domain = m. + [domain in Alexa record]
 - Subject domain = 1 + [domain in Alexa record]
 - Subject domain = 2 + [domain in Alexa record]
 - Subject domain = 3 + [domain in Alexa record]
 - Subject domain = 4 + [domain in Alexa record]
 - Subject domain = 5 + [domain in Alexa record]
 - Subject domain = 6 + [domain in Alexa record]
 - Subject domain = 7 + [domain in Alexa record]
- Step 2: 15000 certificate blockchain blocks generated for the year 2018 using randomly selected domains from the dataset generated in step1. Each block consists of 1024 certificates and each certificate hold one subject domain

- Step 3: 10000 certificate blockchain blocks generated for the year 2019 using randomly selected domains from the dataset generated in step1. Each block consists of 1024 certificates and each certificate hold one subject domain
- Step 4: 6000 certificate blockchain blocks generated for the year 2020 using randomly selected domains from the dataset generated in step1. Each block consists of 1024 certificates and each certificate hold one subject domain

The total generated data represented more than 31740000 certificates. Figure 40 illustrates a sample recordset generated for one subject. The subject has seven certificates distributed in 3 different blockchains. Figure 41 illustrates the data of the latest block that the certificate of the same subject belongs to. It is expected from CAs to include the illustrated record in Figure 41 in their response to PKQuery requests for the subject.

YEAR	HEIGHT	SUBJECT
2,020	3,357	FF9D2A65AD7E24425C58E78E91E5CC26673A66FEDF8AD7E4F5803AD7DA2E8540
2,020	775	FF9D2A65AD7E24425C58E78E91E5CC26673A66FEDF8AD7E4F5803AD7DA2E8540
2,019	7,635	FF9D2A65AD7E24425C58E78E91E5CC26673A66FEDF8AD7E4F5803AD7DA2E8540
2,019	6,958	FF9D2A65AD7E24425C58E78E91E5CC26673A66FEDF8AD7E4F5803AD7DA2E8540
2,018	11,640	FF9D2A65AD7E24425C58E78E91E5CC26673A66FEDF8AD7E4F5803AD7DA2E8540
2,018	9,092	FF9D2A65AD7E24425C58E78E91E5CC26673A66FEDF8AD7E4F5803AD7DA2E8540
2,018	3,880	FF9D2A65AD7E24425C58E78E91E5CC26673A66FEDF8AD7E4F5803AD7DA2E8540

Figure 40: The generated record set for one subject

Year	Height	PreviousBlockHeader	BlockHeader	MerkleRoot	Nonce	BlockTimestamp
2,020	3,357	bc4ff85888c744a80e33093aed3f...	816cd1a125de4c4a...	114e1cbb910a182f3147...	1,063,748,355	2018-11-25 01:26:03

Figure 41: Example block data for the subject

Since the scope of the concept implementation does not cover the certification process, we used random Markle root values in the generated data to speed up the data generation process. For the simulations we created a tool to generate a valid certificate and handle all data generation process (normally the certificate issuance must be executed by the CAs as explained in section 4.4)), the certificate generation tool will be explained in the next section.

The data generation process for the regular certificates is followed by the generation of the data and keys for 10 ConsensusPKI CAs. For each CA we generated an RSA Key pair with 2048-bit key size. In the generated recordset, the certificates of the first six nodes belong to ca_certificateblockchain of 2018, and the remaining four nodes had certificate expiry year of 2019, and their certificates are stored in the ca_certificateblockchain of 2019.

Network setup

In the simulation setup, the CA servers and clients are in the same local network with 1Gbps Ethernet connection. Due to network setup, the response time does not include the network latency due to the distance between the systems.

The Scenario: The simulation assesses the following two scenarios to validate whether the PKQuery protocol is practical:

- Total elapsed time for CA to respond to a query
- Total elapsed time that client application decides whether the connection secure or not

We have conducted the simulation tests using RSA and ECDSA signatures. The results in the next sections are based on RSA encryption and signing.

5.3.1) Valid ConsensusPKI certificate generator

After data generation process, generation of real certificates is required to use it in the PKQuery simulation. As a remedy, we have created a tool that executes the following steps to generate a valid certificate for one subject on each execution. During the certificate generation for one subject, all other subjects in the certificate Merkle tree are randomly selected from the Alexa top 1 million sites data. The tool generates certificate files with the expiry year of 2020. Execution steps of the certificate generation tool are as follows:

- Step 1: Tool asks for the subject domain
- Step 2: Tool generates an RSA key pair with 2048-bit key size and stores both public and private keys in pem format. The certificate will contain the public key generated in this step.
- Step 3: Tool generates an RSA key pair with 2048-bit key size and stores both public and private keys in pem format for the domain. The CVR file will contain a signature signed by the key generated in this step.
- Step 4: Tool generates the full CVR file in ConsensusPKI format including the randomly selected identifier and signature with the domain key.
- Step 5: Tool randomly selects 2046 domains from Alexa top 1 million sites
- Step 6: Tool randomly selects a position in the certificate Merkle tree to position the certificate data of the subject
- Step 7: Tool constructs the certificate Merkle tree
 - Tool distributes the randomly selected sites, retrieved in Step 5, to the Merkle leaves
 - Tool places the hash of the CVR file in the position defined in Step 6
 - Tool places the random hash (instead of **evidence Merkle tree** root) as a sibling hash for the CVR file
- Step 8: Tool calculates the Merkle root and generates the Merkle proof for the certificate
- Step 9: Tool inserts the data into subjects table using the randomly selected subjects in step 5
- Step 10: Tool inserts the real subject entered in Step 1 into the subjects table
- Step 11: Tool inserts the certificateblockchain table the following data
 - Year, height, merkleroot, previousblockheader
- Step 12: Tool runs PoW for the newly inserted data for the certificateblockchain to find blockheader with leading three zeros '000'
 - PoW updates the nonce, blockheader, block timestamp of the block
- Step 13: Tool generates the certificate file with Merkle proof using the proof generated in step 8
- As a final check tool validates the Merkle root in the certificate file

Figure 42 illustrates an example output files of the tool. For example in the figure, we used the www.google.com as the subject domain name. In the normal certification process of the ConsensusPKI, the subject-owner has to send the generated signed CVR file to a CA of his/her choice to start the certification process. The subject owner has to place the domain public key in a DNS txt record under the identifier mentioned in the CVR file, such as

[identifier]._identitykey.domain.tld. Further, the subject-owner has to set up his APKME environment using the Domain Private Key and the Private Key files so that the challenges from the CAs can be answered correctly so that the certificate in the CVR file would be issued. The identifier in the certificate file will be used as the locator of the domain public key and the URL of the APKME environment(s) that CAs will send their challenges during the verification process.

www.google.com1544529059481CertificateFileWithPublicKeyRSAWithProof.jcrt	11-Dec-18 12:51 PM	JCRT File	2 KB
www.google.com1544529059481CertificateFileWithPublicKeyRSAUnsigned.jcrt	11-Dec-18 12:51 PM	JCRT File	1 KB
www.google.com1544529059481CVRFileWithPublicKeyRSASigned.jcrt	11-Dec-18 12:51 PM	JCRT File	2 KB
www.google.com1544529059481DomainPrivateKeyRSA.pem	11-Dec-18 12:51 PM	PEM File	2 KB
www.google.com1544529059481DomainPublicKeyRSA.pem	11-Dec-18 12:51 PM	PEM File	1 KB
www.google.com1544529059481PrivateKeyRSA.pem	11-Dec-18 12:51 PM	PEM File	2 KB
www.google.com1544529059481PublicKeyRSA.pem	11-Dec-18 12:51 PM	PEM File	1 KB

Figure 42: Generated files using the certificate generation tool

Figure 43 illustrates the subjects table after record generation. The certificate in the example is placed in the certificateblockchain of 2020 with height 6008 (the subject in the example is www.google.com).

subjects (3×3)		
year	height	subject
2,020	6,008	191347BFE55D0CA9A574DB77BC8648275CE258461450E793528E0CC6D2DCF8F5
2,020	3,886	191347BFE55D0CA9A574DB77BC8648275CE258461450E793528E0CC6D2DCF8F5
2,018	4,690	191347BFE55D0CA9A574DB77BC8648275CE258461450E793528E0CC6D2DCF8F5

Figure 43: Subjects table record set for www.google.com after certificate generation

Figure 44 illustrates the Merkle root and blockheader values in the certificateblockchain table for the generated certificate illustrated in Figure 42. Blockheader value has three leading zeros. The C and P fields in the generated certificate file must construct the Merkle root illustrated in the figure.

certificateblockchain (7×1)					
Year	Height	PreviousBlockHeader	Block-Header	MerkleRoot	Nonce
2,020	6,008	000969e7934ac8eb7c0...	000fa109fcd7b5...	99aa41a182ccc8465d83818ff0c6f68a9213f78c3fe662466e7bef0d4a4fd62f...	4,982

Figure 44: certificateblockchain record for the generated certificate

Figure 45 illustrates the content of an example generated certificate file with Merkle proof.

```
{
  "C":{
    "V": "4",
    "H": "SHA256",
    "VT": "2020-12-25T13:28:06.419Z",
    "CN": "www.google.com",
    "PublicKey": "-----BEGIN PUBLIC KEY-----
\\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEApdn+coZCISnJJ390EFN\\nrcTqVdGE+sYdS9+APU8FkA
4iMAf+1bCliVuJVxfyXklm7EtD75PmhZ1VXOm2uDnh\\n+3pwoghyu8TeRB2Gfixzh0XOnP4999r/3p5+/ZxMMx2sJe
BW5JHnbKCSkNX0ffFM\\nIUc9GBippEJTIYdRWi7GY6ukmgdaFeRp4vnD1K4DeqO0B+N1w3LVTRln9N0eKA28\\n/e
DMYIWZgXiGsABfmhdxV1SSm6H0pNmE+LiQ2FW6s1fnRYdmlUdsD88m236S08oQ\\n404hUlb2W/7i8RuyGkEh96B
PzHxyVmegsW6uhT9GTuMMt37+bSRNUtShwBrAWpqm\\nFwIDAQAB\\n-----END PUBLIC KEY-----\\n"
  },
  "P": [
    { "right": "909822c8eb1b8f6083e1a138e73aa3917af1e4be5274c2746222526f2553dff7"},
    { "right": "83c82b20107ecc78eed3cb134ca5a330d5fdb160c4a2ba06e0a8e3317994ffeb"},
    { "right": "65629d2b2880d186dfb86651e117d169fb3d58d6675bb243f00848947878af2f"},
    { "left": "8186b31e6e794c4ff247db046e35f5766eb823081bde71aa132d32cc07b306a3"},
    { "left": "bc874522f9b259406270e591a49ef971cca85b6ab2a78c99d2778c8884b1e7f1"},
    { "left": "fba114f1e005f7b45809aa8a3447d49caed613843fa92c392b6b8477f2000413"},
    { "right": "a0acf5b8054b343c44e7928785ef3704002862f1fb22dae156a87ea576c1b03e"},
    { "left": "1345fb785f4b5a4954413455a842902f07e7c8110755c5723923338c070214f4"},
    { "right": "9884b2a400a8b40ff32e1e3192e4660c6f3c74030ea9e986ecd22184f806f886"},
    { "right": "8a4e8caa35689b422bc62f41942d006e771a939e5fc056bb443e9b59d00ce39b"},
    { "left": "2398c36de0796568f92081ac20d793491cfcd8836a60bffa51120ef5c3042f0d"}
  ]
}
```

Figure 45: An example generated certificate file with proof

Figure 46 illustrates an example run of the CVR and certificate generation tool. The above-explained steps and related information are printed on the screen.

```
Command Prompt - python JCRandDataGenerator.py
Step1 Please enter a domainname to generate a certificate: www.google.com
Step2 is finilized: Web key for the certificate is generated
Step3 is finilized: Domain key for the CVR is generated
Step4 is finilized: Certificate W/O Proof and Signed CVR are generated
The Identifier for the APKME and domain publickey locator is: 8a36e2e473f6967b
Step5 is finilized: Random domains are selected
Step6 is finilized: The certificate is placed in the Merkle tree
Step7 is finilized: The merkle tree is constructed. The position of the certificate is: 1208
Step8 is finilized: Merkle Root of Certificate No: 1208
75858ab398a804edaac0d50e2c41a6e1cb77a1bbb07e131169eabb943075b6f6
Step 8: the Proof of Certificate No: 1208
[{'right': '909822c8eb1b8f6083e1a138e73aa3917af1e4be5274c2746222526f2553dff7'}, {'right': '83c82b20107ecc78eed3cb134ca5a330d5fdb160c4a2ba06e0a8e3317994ffeb'}, {'right': '65629d2b2880d186dfb86651e117d169fb3d58d6675bb243f00848947878af2f'}, {'left': '8186b31e6e794c4ff247db046e35f5766eb823081bde71aa132d32cc07b306a3'}, {'left': 'bc874522f9b259406270e591a49ef971cca85b6ab2a78c99d2778c8884b1e7f1'}, {'left': 'fba114f1e005f7b45809aa8a3447d49caed613843fa92c392b6b8477f2000413'}, {'right': 'a0acf5b8054b343c44e7928785ef3704002862f1fb22dae156a87ea576c1b03e'}, {'left': '1345fb785f4b5a4954413455a842902f07e7c8110755c5723923338c070214f4'}, {'right': '9884b2a400a8b40ff32e1e3192e4660c6f3c74030ea9e986ecd22184f806f886'}, {'right': '8a4e8caa35689b422bc62f41942d006e771a939e5fc056bb443e9b59d00ce39b'}, {'left': '2398c36de0796568f92081ac20d793491cfcd8836a60bffa51120ef5c3042f0d'}]
Step 9 and 10 are finilized: All subjects are inserted into the subjects table
Step 11 is finilized: the record is inserted into the certificateblockchain table
Step 12: Updating the temporary fields to start PoW
Step 12: PoW is Started
Step 12 is finilized: PoW is Finilized
Fingerprint of the Certificate No: 1208
0cfe10348377e7b608fd99611b69e9387f3967c2f162cad93c9b1c6f1df4df9a
Step 13 is finilized: The certificate and all related records are generated
The validation result of the Merkle root in the certificate file
True
Step1 Please enter a domainname to generate a certificate:
```

Figure 46: An example screenshot of the certificate generator

Figure 47 illustrates the content of a CVR file that domain owner needs to send to CA of his/her choice to initiate the formal certification process in a fully implemented scenario. The concept implementation does not cover the certification process itself and generates the certificate as if the certification is finalized. The steps between 5 and 13 of the certificate generation tool are the formal steps of the certification process. In the simulation, it is assumed implemented. The value in the **field I** indicates the identifier value and the value in the **field S** is the signature on the CVR file. The signature is generated using the value of **field C** as follows:

Signature = $\text{Sign}(h(C))_{k_{\text{prd}}}$ where k_{prd} = Private part of the domain key. The signature in the field is a hexadecimal string.

```
{
  "C":{
    "V": "4",
    "H": "SHA256",
    "VT": "2020-12-25T13:28:06.419Z",
    "CN": "www.google.com",
    "PublicKey": "-----BEGIN PUBLIC KEY-----
\\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEApdn+co0ZCISnJJ390EFN\\nrcTqVdGE+sYdS9+APU8FkA
4iMAf+1bCiiVuJVxfyXklm7EtD75PmhZ1VXOm2uDnh\\n+3pwoghyu8TeRB2GFixzh0XOnP4999r/3p5+/ZxMMx2sJe
BW5JHnbKCskNX0FFm\\nIUc9GBippEJTIYdRWi7GY6ukmgdaFeRp4vnD1K4DeqO0B+N1w3LVTRln9N0eKA28\\n/e
DMYIWZgXiGsABfmhdxV1SSm6H0pNmE+LiQ2FW6s1fnRYdmlUdsD88m236S08oQ\\n404hUlb2W/7i8RuyGkEh96B
PzHxyVmegsW6uhT9GTuMMt37+bSRNUtShwBrAWpqm\\nFwIDAQAB\\n-----END PUBLIC KEY-----\\n"
  },
  "I": "8a36e2e473f6967b",
  "S": "3abc1a5659584988d566f8b7a13529abefd39b0721c308f841698e68438128160729dd754d0debe5f681bf2
07afdea5d4b56e47e772af58149d381285ecd2b02ba1c7de0257c16bf2927ce24d019dce11d220b9c94f77e12c07d
3853dd6e7ed82c6e2a1817368997f4b5d2653c432ec548d2d0227d31d43d56daf9513f3b9f543e0b9f8c3b0da2287
95605850cf7f4f5ce77b70ef111262f53ad94ba1bfa026fc8942143af7bf6e463ad9aa55e60deb335dfabf727b665ef2
4f5ea45c8376a4fd57ef7aab57c14d5920661c013f048dbc66fb78fe93aaf30ef3cb5f00c51aabf7ddd0effca50a01048
0dc621bb5688d0e508ebe4c9c3c770ac6968285f9aba78"
}
```

Figure 47: A Signed CVR file to send to a CA to initiate the certificate issuance process

5.3.2) The CA Server Application for the PKQuery protocol

The server application is a simple application that accepts an encrypted message that contains the following: nonce + $h(\text{subject})$. In the simulation, we used 8 bytes nonce value and SHA256 as a hash function. In the full implementation, it is possible to agree on another fixed size nonce (The security parameters are not in the scope of the simulation test). The setup for the simulation gives us 40 bytes plaintext message size. The following are the boot process of the server application:

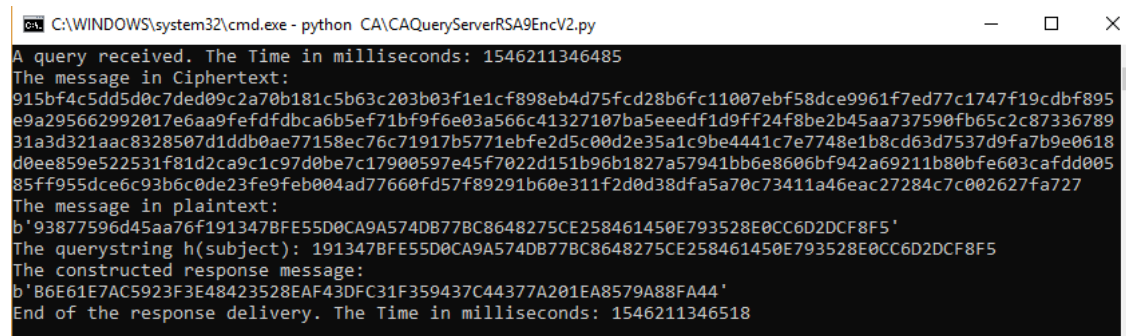
- Step 1: It reads variables for the binding address and port number to create a listening UDP server socket
- Step 2: it loads the private key stored in pem format
- Step 3: it loads the public key stored in pem format
- Step 4: it creates a decrypting object to handle RSA decryption for the incoming messages
- Step 5: it creates the multithread UDP listening server

The server application handles every incoming message with a separate thread for parallel processing. The following is the execution steps when a PKQuery message is received:

- Step 1: the datagram message is decrypted using the CAs private key

- Step 2: the plaintext found in Step 1 is split into two parts
 - First, 8 bytes are the client nonce
 - The remaining bytes are: the h(subject) value
- Step 3: a database query is executed using the h(subject) value on the vw_response database view, explained in the section 5.1.3) Considerations on Datastore on CA infrastructure for Regular Certificates. If the query finds a record, it retrieves only the first record as the valid record. The retrieved value is responseHash = h(h(subject)|| MerkleRoot)
- Step 4: the response message R is constructed using the responses in step 3 and nonce in step 2
 - $R = h(\text{nonce} + \text{responseHash})$
- Step 5: the signature value S is calculated using the R in step 4, and the private key the CA
- Step 6: the expected response message R+S is constructed and send as a response to the client

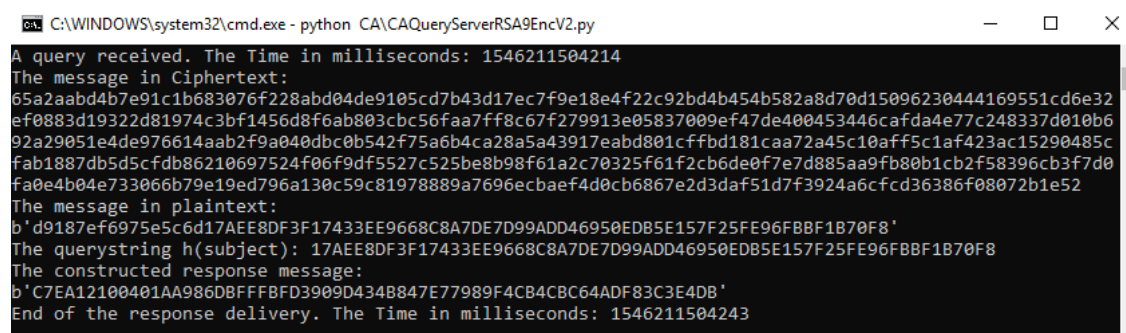
If the CA server application does not find a corresponding value in its database, it sends the h(nonce) as a response to complete the communication cycle. Figure 48 illustrates a valid response message, and Figure 49 illustrates an empty response message. In the empty response message, the constructed message is the hash value of the client nonce.



```

C:\WINDOWS\system32\cmd.exe - python CA\CAQueryServerRSA9EncV2.py
A query received. The Time in milliseconds: 1546211346485
The message in Ciphertext:
915bf4c5dd5d0c7ded09c2a70b181c5b63c203b03f1e1cf898eb4d75fcd28b6fc11007ebf58dce9961f7ed77c1747f19cdbc895
e9a295662992017e6aa9fefdfdbca6b5ef71bf9f6e03a566c41327107ba5eedf1d9ff24f8be2b45aa737590fb65c2c87336789
31a3d321aac8328507d1ddb0ae77158ec76c71917b5771ebfe2d5c00d2e35a1c9be4441c7e7748e1b8cd63d7537d9fa7b9e0618
d0ee859e522531f81d2ca9c1c97d0be7c17900597e45f7022d151b96b1827a57941bb6e8606bf942a69211b80bfe603cafd005
85ff955dce6c93b6c0de23fe9feb00ad77660fd57f89291b60e311f2d0d38dfa5a70c73411a46eac27284c7c002627fa727
The message in plaintext:
b'93877596d45aa76f1913478FE55D0CA9A574DB77BC8648275CE258461450E793528E0CC6D2DCF8F5'
The querystring h(subject): 1913478FE55D0CA9A574DB77BC8648275CE258461450E793528E0CC6D2DCF8F5
The constructed response message:
b'B6E61E7AC5923F3E48423528EAF43DFC31F359437C44377A201EA8579A88FA44'
End of the response delivery. The Time in milliseconds: 1546211346518
  
```

Figure 48: An example screen print of the CA PKQuery server with a valid response



```

C:\WINDOWS\system32\cmd.exe - python CA\CAQueryServerRSA9EncV2.py
A query received. The Time in milliseconds: 1546211504214
The message in Ciphertext:
65a2aabd4b7e91c1b683076f228abd04de9105cd7b43d17ec7f9e18e4f22c92bd4b454b582a8d70d15096230444169551cd6e32
ef0883d19322d81974c3bf1456d8f6ab803cbc56faa7ff8c67f279913e05837009ef47de400453446cafd4e77c248337d010b6
92a29051e4de976614aab2f9a040dbc0b542f75a6b4ca28a5a43917eabd801cfffbd181caa72a45c10aff5c1af423ac15290485c
fab1887db5d5cfd86210697524f06f9df5527c525be8b98f61a2c70325f61f2cb6de0f7e7d885aa9fb80b1cb2f58396cb3f7d0
fa0e4b04e733066b79e19ed796a130c59c81978889a7696ecbaef4d0cb6867e2d3daf51d7f3924a6cfd36386f08072b1e52
The message in plaintext:
b'd9187ef6975e5c6d17AEE8DF3F17433EE9668C8A7DE7D99ADD46950EDB5E157F25FE96FBBF1B70F8'
The querystring h(subject): 17AEE8DF3F17433EE9668C8A7DE7D99ADD46950EDB5E157F25FE96FBBF1B70F8
The constructed response message:
b'C7EA12100401AA986DBFFBFD3909D434B847E77989F4CB4CBC64ADF83C3E4DB'
End of the response delivery. The Time in milliseconds: 1546211504243
  
```

Figure 49: An example screen print of the CA PKQuery server with an empty response

5.3.3) PKQuery Client Application

The consumers of the ConsensusPKI ecosystem are the end the user systems. The application on the end user systems must validate the certificate and accept to use the certificate to interpret the connection as secure. The client simulation application executes the following steps during the bootstrap of the client application:

- Bootstrap: The PKQuery client application loads the valid CA certificates into its memory using the available local database and the vw_ca database view explained in the section 5.1.1) Considerations on CA Datastore on end-user systems. The client application loads only the latest valid certificate into the memory for every CA subject. In the intended full implementation, the client application should also execute below additional checks:
 - The expiry dates on the client certificates should be checked against the local time of the end user system
 - The integrity of the CA certificate datastore on the client should be checked using the integrity check functions explained in Figure 36: Blockheader calculation and integrity check pseudocodes for CA_certificateblockchain

For simplicity of the simulation above mentioned expiry date and integrity checks are not executed for the simulation and assumed the data in the simulation database is intact.

The following is the execution steps for the client application to verify a certificate file for a subject:

- Step 1: the client application asks for the subject as an input
- Step 2: the client application asks the certificate file as an input. It is assumed that the web server of the subject has sent the certificate file
- Step 3: the client application constructs following variables using the entered inputs
 - $H(\text{subject})$ using the entered subject. (lowercase of the entered subject is used)
 - MerkleRoot from the certificate file is constructed
- Step 4: $h(\text{subject}) + \text{MerkleRoot}$ is constructed
- Step 5: A random nonce is selected
- Step 6: The expected response R' is calculated as the following:
 - $R' = H(\text{nonce} + h(h(\text{subject}) + \text{MerkleRoot}))$
- Step 7: The PKQuery message in cleartext is generated as follows:
 - $Q = \text{nonce} + h(\text{subject})$
- Step 8: 4 CAs are randomly selected from the memory
- Step 9: 4 Query threads are generated, and requests are sent to each CA in a different thread
 - Before the requests are transmitted to the server of the CAs, the Q values are encrypted by asymmetric encryption using the public key of each CA loaded during the client application bootstrap
- Step 10: The client application waits to receive all responses from the CAs
 - The received response are $\{R_{CA1}, R_{CA2}, R_{CA3}, R_{CA4}\}$
- Step 11: The client application checks the signatures on the messages using the public key of each CA loaded during the client application bootstrap
- Step 12: The client application executes the PBFT algorithm to take a final decision
 - The normal expected result is $R' = R_{CA1} = R_{CA2} = R_{CA3} = R_{CA4}$

The illustrates an example execution case where all responses from CAs match with the proof of the certificate.

```

Command Prompt - python PKQ_RSA_ClientVencV3.py
Please Enter The Subject : www.google.com
Please Enter Certificate Filename : www.google.com1544532406805CertificateFileWithPublicKeyRSAWithProof.jcrt
PKQuery check is started. The time in milliseconds: 1544556398400
h(subject) + MR from Certificate :A1104809E6320C076766996E77DF04C40C4892B6C1CC8C7CF731FD542DA6A41
The HSubject: 191347BFE55D0CA9A574DB77BC8648275CE258461450E793528E0CC6D2DCF8F5
Randomly selected client nonce: 809F592649Fe8761
The expected response calculated using the proof in the certificate: BCA06794E6D073D705F3898FFB2E5341DAA737C3EBD2F401A7DC2E24D178D34D
The start of the PKQuery threads. The time in milliseconds: 1544556398404
The end of the PKQuery threads. The time in milliseconds: 1544556398496
The start of the validation checks. The time in milliseconds: 1544556398496
BCA06794E6D073D705F3898FFB2E5341DAA737C3EBD2F401A7DC2E24D178D34D
BCA06794E6D073D705F3898FFB2E5341DAA737C3EBD2F401A7DC2E24D178D34D
BCA06794E6D073D705F3898FFB2E5341DAA737C3EBD2F401A7DC2E24D178D34D
BCA06794E6D073D705F3898FFB2E5341DAA737C3EBD2F401A7DC2E24D178D34D
BCA06794E6D073D705F3898FFB2E5341DAA737C3EBD2F401A7DC2E24D178D34D
Signature Verifications are OK. The time in milliseconds: 1544556398501
All Responses are equal.
Connection is secure Certificate matches the Accepted response : BCA06794E6D073D705F3898FFB2E5341DAA737C3EBD2F401A7DC2E24D178D34D
The PKQuery check is finalized. The time in milliseconds: 1544556398502

```

Figure 50: An example execution of the PKQuery indicating certificate as valid

5.3.4) PKQuery simulation results

We run 12 rounds for each case, and during execution of the test cases the record counts on the subjects table are as following:

- Total number of records: 31.775.265
- Number of unique subjects: 9.460.850

Results for Case1:

Figure 51 illustrates the results for case 1. The numbers are elapsed time in milliseconds. The average of the CA response time is 66.4 milliseconds, and on average it took 102.7 milliseconds for the client to take a decision. In all cases, the connection accepted as secure, even though in **half of the cases** one of the responding CA was **not** honest. On detection of the non-honest CA, the client also generated evidence to send it to other CAs to initiate the honesty check and CA certificate revocation processes of the ConsensusPKI. For example, for the run number 1, the client sent the PKQueries to CA1, CA2, CA4, and CA9.

	CA1	CA2	CA3	CA4	CA5	CA6	CA7	CA8	CA9	CA10	Client
Run 1	69	70		70					65		95
Run 2		55		58	58			52			90
Run 3	86	55		65			54				103
Run 4					63	54		67		79	126
Run 5		55	64			54		60			90
Run 6		61	54			67	59				91
Run 7	98				90			97		109	128
Run 8		56				67		63	68		111
Run 9			58			64		61		67	96
Run 10	69		62					62	67		103
Run 11	66	61	67		64						96
Run 12		55	62		58	65					103
Average	77.6	58.5	61.2	64.3	66.6	61.8	56.5	66.0	66.7	85.0	102.7
Count	5	8	6	3	5	6	2	7	3	3	48

Figure 51: Simulation results for Case 1

Results for Case2:

Figure 52 illustrates the results for case 2. The average of the CA response time is 62.6 milliseconds, and on average it took 102.3 milliseconds for the client to take a decision. As expected, in none of the cases, the connection was interpreted as secure, since the certificate was not valid for the subject. In 6 of the cases the non-honest CAs gave responses, and in the run number 7, there was no consensus among the responses of the

CAs. An interesting result for the case is that even the connection for the client interpreted as **not secure**, the non-honest CAs are **detected**. On detection of the non-honest CA, the client also generated evidence to send it to other CAs to initiate the honesty check and CA certificate revocation processes of the ConsensusPKI. On the run 7, all of the involved CAs receives all evidence. Since the CA number 4 and 7 are honest CAs, the CA non-honesty check will confirm that they are honest.

	CA1	CA2	CA3	CA4	CA5	CA6	CA7	CA8	CA9	CA10	Client
Run 1			78	92	72	89					124
Run 2				58		66		77		71	111
Run 3			60	60		66				63	106
Run 4					56	53		51		52	102
Run 5		82	55	55		75					111
Run 6	82			56			56	55			107
Run 7				51			52		66	55	91
Run 8	55	56	55						51		93
Run 9		54		58			57	65			92
Run 10	61				59	67	78				92
Run 11		55	60			54	83				103
Run 12			57	62	60					68	95
Average	66.0	61.8	60.8	61.5	61.8	67.1	65.2	62.0	58.5	61.8	102.3
Count	3	4	6	8	4	7	5	4	2	5	48

Figure 52: Simulation results for Case 2

Results for Case3:

Figure 53 illustrates the results for case 3. The average of the CA response time is 58.1 milliseconds, and on average it took 93.3 milliseconds for the client to take a decision. As expected in none of the cases, the connection interpreted as secure, since the certificate was not valid for the subject. In all of the cases, a non-honest CA was detected. Similar to run 7 of case 2, there was no consensus among CAs for the run 4.

	CA1	CA2	CA3	CA4	CA5	CA6	CA7	CA8	CA9	CA10	Client
Run 1			51			55	54			58	75
Run 2	58	49			54					56	90
Run 3			60		61		63		66		93
Run 4					58		52		54	57	84
Run 5					70		65	65		69	97
Run 6			54	65		55			55		82
Run 7		59			60			54	56		83
Run 8					56	55	57			56	93
Run 9			57	57		56			57		104
Run 10			49		50		48			64	96
Run 11	69						60	62		60	116
Run 12			56	53			61			75	107
Average	63.5	54.0	54.5	58.3	58.4	55.3	57.5	60.3	57.6	61.9	93.3
Count	2	2	6	3	7	4	8	3	5	8	48

Figure 53: simulation results for Case 3

Results for Case4:

Figure 54 illustrates the results for case 4. The average of the CA response time is 62.4 milliseconds, and on average it took 94.8 milliseconds for the client to take a decision. As expected in none of the cases, the connection interpreted as secure, since there was no certificate issued for the subject. In 10 of the runs, a non-honest CA was detected, and there was no consensus among CAs for 4 of the runs. The case confirms that the non-honesty detection mechanism of the ConsensusPKI works even there exists no certificate for a queried subject.

	CA1	CA2	CA3	CA4	CA5	CA6	CA7	CA8	CA9	CA10	Client
Run 1		75		62		68				61	111
Run 2							60	63	59	62	96
Run 3	63			58					56	58	98
Run 4			62			60		58		56	81
Run 5	61		60		61					61	80
Run 6		51			66		51		49		81
Run 7			59		59			75		57	108
Run 8	54	53		54	52						70
Run 9	57					54		53	53		73
Run 10	83		77	81	80						116
Run 11	85							82	87	82	134
Run 12		60			73			59	57		89
Average	67.2	59.8	64.5	63.8	65.2	60.7	55.5	65.0	60.2	62.4	94.8
Count	6	4	4	4	6	3	2	6	6	7	48

Figure 54: Simulation results for Case 4

Overall Results:

Figure 55 illustrates the combined results of the simulation. The average of the CA response time is 62.4 milliseconds, and on average it took 98.3 milliseconds for the client to take a decision. It is measured that the response time of the CA1 is slightly higher than other CAs. This indicates that the extra logging creates extra latency. Since we had to measure the elapsed time for all CAs, we can conclude that measuring the results influences the results. In real life scenario with a production level hardware, where CAs do not log the requests and answer the queries honestly, the response time expected to considerably less than our simulation. Further, our language and platform choice influence the overall performance of the simulation. We expect to have better results with C++ since it removes the interpret to machine code conversion stage during the execution. Additionally, usage of an in-memory database, such as apache ignite, expected to increase the performance of the CA responses. Finally, the implementation itself is not fully optimized python code. The following implementation change in the implementation could decrease the elapsed time-to-decision on the client side. The client application in the concept implementation first parses the certificate and finds the expected response, and after that, it sends the PKQuery requests to the CA servers. An alternative to this process could be that the client application directly asks the PKQueries and creates another thread to parse and find the expected response. This will remove the sequential execution from the client application. Another full alternative could be to use multitasking in python instead of threads. The python multitasking implementation uses the advantage of multicore CPUs and expected to perform

better in multicore CPU systems. The implementors should consider that the multitasking does not use shared memory space and it brings its engineering challenges.

Finally, we can conclude the following:

- Although there is room for improvement, the average time-to-decision is an acceptable duration for clients to interpret the connection between the subject server and the client
- Although there is room for improvement, the average time-to-respond is an acceptable duration for servers to give a response to the PKQuery requests
- The PKQuery detected all non-honest CA responses successfully
- Due to **client nonce**, secure connection indicator is **different** on every PKQuery responses, even though the subject and the certificate does not change
- Response times of the CAs indicate that a CA might be collecting extra logs
- On roughly ~10% of the cases there was no consensus, even though the 20% of the CAs were not honest at the same time. Although 20% non-honest CA is a huge percentage (there exist ~1750 CAs at the moment, 20% means compromise of 350 CAs at the same time), the impact of such event for the end users as unavailability is **reduced** due to fault tolerance based on PBFT

	CA1	CA2	CA3	CA4	CA5	CA6	CA7	CA8	CA9	CA10	Client
<i>Average</i>	69.8	59.0	59.9	61.9	62.7	62.2	59.4	64.1	60.4	65.0	98.3
<i>Count</i>	16	18	22	18	22	20	17	20	16	23	192

Figure 55: Overall results of the simulation

Please see the Appendix A for the output of the some of the example simulations runs.

To confirm the effect of the measuring on the response times, we have conducted additional 12 runs (8 runs Case 1, and 4 runs Case 3) with the following setup:

- We updated the non-honest CA nodes respond honestly
- We removed all logging on server applications
- Logging on client site unchanged

The result confirmed that the logging on CA affects the response times. In 12 runs the maximum time-to-decision on the client side was 88 milliseconds and the minimum 70 milliseconds. The average time-to-decision dropped to 77.1 milliseconds for the client to take a decision. We can conclude that the effect of logging and measuring is ~21% in milliseconds on our previous setup.

5.4) Formal validation of PKQuery protocol using Tamarin prover

We have developed a tamarin prover model for formal validation of the PKQuery protocol. (URL for the source code is in the footnote 1 on page 77). We have tested the PKQuery protocol against the known attacks in a Dolev-Yao, adversary model. PKQuery must satisfy the following theorems in order to satisfy the security requirements in a Dolev-Yao adversary environment:

- The secrecy of the subject implies that the subject value is only known to the client
- The secrecy of the nonce value implies that the nonce value that the client sends to the CA server is only known to the client and the holder of the CA private key

- Message authentication implies that the only CA that received the request can reply with valid responses
- The authenticity of the client is used against known replay attacks and implies that only clients who have the public key of the CA can send authentic messages to the CA server
- Authenticity Injective of the client implies that there exists only one client that the CA sends its response

```

lemma Client_subject_secrecy:
  all-traces
  "~(∃ Server subject nonce merkle root #i #j.
    ((ValidCertIdentity( Server,
      h(<nonce, h(<subject, merkle root>)>)
    ) @ #i) ∧
    (K( subject ) @ #j)) ∧
    (¬(∃ #r. LtkReveal( Server ) @ #r))))"

```

Figure 56: lemma Client_subject_secrecy for PKQuery

```

lemma Client_nonce_secrecy:
  all-traces
  "~(∃ Server subject nonce merkle root #i #j.
    ((ValidCertIdentity( Server,
      h(<nonce, h(<subject, merkle root>)>)
    ) @ #i) ∧
    (K( nonce ) @ #j)) ∧
    (¬(∃ #r. LtkReveal( Server ) @ #r))))"

```

Figure 57: lemma Client_nonce_secrecy for PKQuery

```

lemma message_authentication:
  all-traces
  "∀ client nonce subject merkle root #i.
    (Authentic( client,
      h(<nonce, h(<subject, merkle root>)>)
    ) @ #i) ⇒
    ((∃ #j.
      (AnswerRequest( client,
        h(<nonce, h(<subject, merkle root>)>)
      ) @ #j) ∧
      (#j < #i)) ∨
    (∃ Client #r.
      ((LtkReveal( Client ) @ #r) ∧ (Honest( Client ) @ #i)) ∧
      (#r < #i))))"

```

Figure 58: lemma message_authentication for PKQuery

```

lemma Client_Authenticity:
  all-traces
  "∀ Server subject nonce merkle root #i.
    (ValidCertIdentity( Server,
      h(<nonce, h(<subject, merkle root>)>)
    ) @ #i) ⇒
    ((∃ #a.
      AnswerRequest( Server,
        h(<nonce, h(<subject, merkle root>)>)
      ) @ #a) ∨
    (∃ #r. (LtkReveal( Server ) @ #r) ∧ (#r < #i))))"

```

Figure 59: lemma Client_Authenticity for PKQuery

```

lemma Client_Authenticity_Injective:
  all-traces
  "∀ Server subject nonce merkle root #i.
    (ValidCertIdentity( Server,
      h(<nonce, h(<subject, merkle root>)>)
    ) @ #i) ⇒
    ((∃ #a.
      (AnswerRequest( Server,
        h(<nonce, h(<subject, merkle root>)>)
      ) @ #a) ∧
      (∀ #j.
        (ValidCertIdentity( Server,
          h(<nonce, h(<subject, merkle root>)>)
        ) @ #j) ⇒
        (#i = #j))) ∨
      (∃ #r. (LtkReveal( Server ) @ #r) ∧ (#r < #i)))"

```

Figure 60: lemma Client_Authenticity_Injective for PKQuery

Constraint system for the lemma message_authentication:

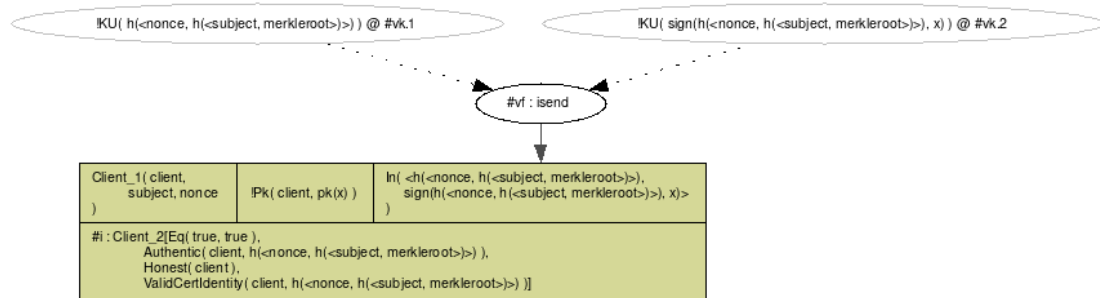


Figure 61: Constraint system for message_authentication

Solution:
formulas:
 \exists Server subject nonce merkle root #i #j.
 (ValidCertIdentity(Server,
 h(<nonce, h(<subject, merkle root>)>)
) @ #i) ∧
 (K(subject) @ #j)
 ∧
 \forall #r. (LtkReveal(Server) @ #r) \Rightarrow \perp
equations:
 subst:
 conj:
lemmas: $\forall x y \#i. (Eq(x, y) @ \#i) \Rightarrow x = y$

Figure 62: Solution of constraint system for message_authentication

Constraint system for the lemma Client_subject_secrecy:

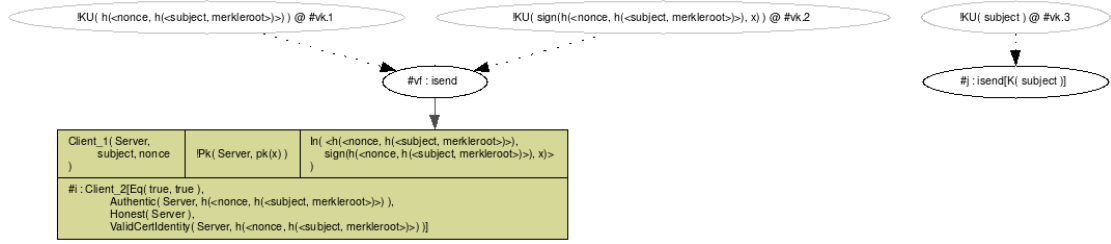


Figure 63: Constraint system for Client_subject_secretcy

Solution:

formulas:

\exists Server subject nonce merkleroot #i #j.
 (ValidCertIdentity(Server,
 h(<nonce, h(<subject, merkleroot>->-)
) @ #i) \wedge
 (K(nonce) @ #j)
 \wedge
 \forall #r. (LtkReveal(Server) @ #r) $\Rightarrow \perp$

equations:

subst:

conj:

lemmas: $\forall x y \#i. (Eq(x, y) @ \#i) \Rightarrow x = y$

Figure 64: Solution of constraint system for Client_subject_secretcy

Constraint system for the lemma Client_nonce_secretcy:

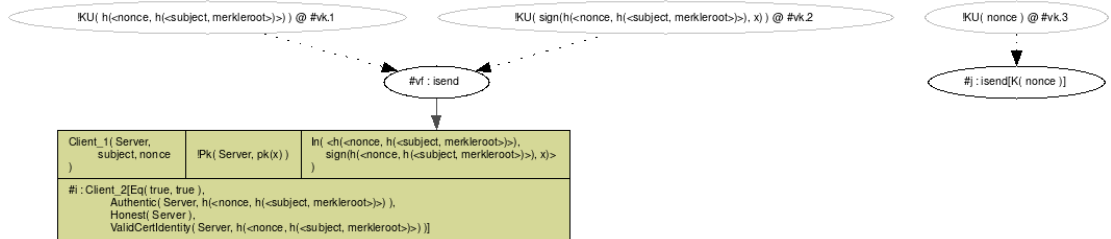


Figure 65: Constraint system for Client_nonce_secretcy:

Solution:

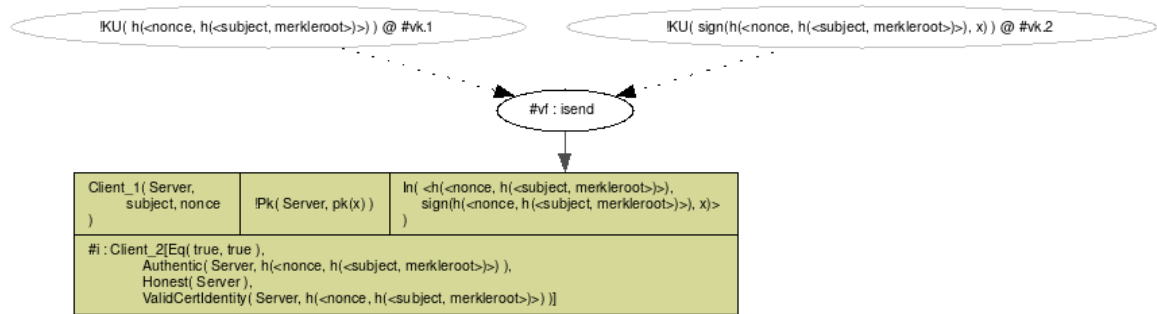
formulas:

$$\begin{aligned}
& \exists \text{ Server subject nonce merkle root } \#i. \\
& (\text{ValidCertIdentity}(\text{Server}, \\
& \quad \text{h}(\langle \text{nonce}, \text{h}(\langle \text{subject}, \text{merkle root} \rangle) \rangle) \\
& \quad) @ \#i) \\
& \wedge \\
& (\forall \#a. \\
& \quad (\text{AnswerRequest}(\text{Server}, \\
& \quad \quad \text{h}(\langle \text{nonce}, \text{h}(\langle \text{subject}, \text{merkle root} \rangle) \rangle) \\
& \quad \quad) @ \#a) \\
& \Rightarrow \\
& \quad \perp) \wedge \\
& (\forall \#r. (\text{LtkReveal}(\text{Server}) @ \#r) \Rightarrow \neg(\#r < \#i))
\end{aligned}$$

equations:

subst:

conj:

lemmas: $\forall x y \#i. (\text{Eq}(x, y) @ \#i) \Rightarrow x = y$ *Figure 66: Solution of the constraint system for Client_nonce_secrecy***Constraint system for the lemma Client authenticity***Figure 67: Constraint system for the client authenticity*

Solution

formulas:

$$\begin{aligned}
& \exists \text{ Server subject nonce merkle root } \#i. \\
& (\text{ValidCertIdentity}(\text{Server}, \\
& \quad h(\langle \text{nonce}, h(\langle \text{subject}, \text{merkle root} \rangle) \rangle) \\
&) @ \#i) \\
& \wedge \\
& (\forall \#a. \\
& (\text{AnswerRequest}(\text{Server}, \\
& \quad h(\langle \text{nonce}, h(\langle \text{subject}, \text{merkle root} \rangle) \rangle) \\
&) @ \#a) \\
& \Rightarrow \\
& \exists \#j. \\
& (\text{ValidCertIdentity}(\text{Server}, \\
& \quad h(\langle \text{nonce}, h(\langle \text{subject}, \text{merkle root} \rangle) \rangle) \\
&) @ \#j) \\
& \wedge \\
& \neg(\#i = \#j)) \wedge \\
& (\forall \#r. (\text{LtkReveal}(\text{Server}) @ \#r) \Rightarrow \neg(\#r < \#i))
\end{aligned}$$

equations:

subst:

conj:

lemmas: $\forall x y \#i. (\text{Eq}(x, y) @ \#i) \Rightarrow x = y$

Figure 68: Solution of the constraint system for the client authenticity

Constraint system for the lemma Client_Authenticity_Injective:

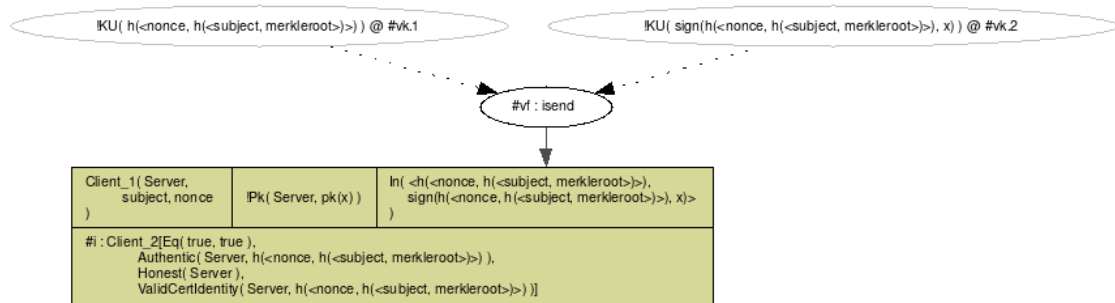


Figure 69: Constraint system for the Client_Authenticity_Injective

Solution

formulas:

$$\begin{aligned}
& \exists \text{ Server subject nonce merkleroot } \#i. \\
& \quad (\text{ValidCertIdentity}(\text{Server}, \\
& \quad \quad h(\langle \text{nonce}, h(\langle \text{subject}, \text{merkleroot} \rangle) \rangle) \\
& \quad) @ \#i) \\
& \wedge \\
& (\forall \#a. \\
& \quad (\text{AnswerRequest}(\text{Server}, \\
& \quad \quad h(\langle \text{nonce}, h(\langle \text{subject}, \text{merkleroot} \rangle) \rangle) \\
& \quad) @ \#a) \\
& \Rightarrow \\
& \exists \#j. \\
& \quad (\text{ValidCertIdentity}(\text{Server}, \\
& \quad \quad h(\langle \text{nonce}, h(\langle \text{subject}, \text{merkleroot} \rangle) \rangle) \\
& \quad) @ \#j) \\
& \wedge \\
& \quad \neg(\#i = \#j)) \wedge \\
& (\forall \#r. (\text{LtkReveal}(\text{Server}) @ \#r) \Rightarrow \neg(\#r < \#i))
\end{aligned}$$

equations:

subst:

conj:

lemmas: $\forall x y \#i. (\text{Eq}(x, y) @ \#i) \Rightarrow x = y$

Figure 70: Solution of the constraint system for the *Client_Authenticity_Injective*

Tamarin prover could not find any attack on PKQuery protocol and verified our theories. The details of the output of the tamarin prover tests can be examined by executing a test using the model file published with the source code of the concept implementation. Like PKQuery it is also possible to model other protocols of the ConsensusPKI using the tamarin prover. The fetch protocol is very similar to PKQuery, and we expect the same results for the fetch protocol. Modeling APKME challenge and responses during the certificate issuance can be a subject of another research.

Chapter 6) Evaluation

In this chapter, we will evaluate the results of the thesis in three sections. In section 6.1, we will evaluate whether the thesis achieved the goals of the research. Additionally, we will explain the impact of the ConsensusPKI to the end users and corporate networks upon standardization. In section 6.2, we will discuss the impact of the ConsensusPKI on the whole cyberspace, using the conceptualization of the J. van den Berg et al. [61]. Finally, in section 6.3 we will discuss the shortcomings of the ConsensusPKI.

6.1) Reflections on the ConsensusPKI

The goal of the research is to satisfy the following requirements determined by the structural flaws of X.509, the scalability of the ecosystem itself, and the convenience of the end users who consume the resources of the ConsensusPKI:

- Requirement 1: Transparency on all CA processes
- Requirement 2: Transparency on all public keys used in the cyberspace
- Requirement 3: Temper proof data structure
- Requirement 4: No chain reactions due to trust revocation
- Requirement 5: Temper proof certification process
- Requirement 6: Higher public key certainty
- Requirement 7: Scalable ecosystem
- Requirement 8: High performing

Below we outline the most important characteristics of the ConsensusPKI by which it meets the requirements mentioned above:

- Characteristic 1: The certificate issuance and the interpretation of the certificates require the collective effort of the CAs bringing transparency to all processes of the CAs. The certificate issuance process requires verification of the CVRs by multiple CAs, and all challenge-response verifications executed during the certificate issuance must be broadcasted to other CAs so that the verified CVRs can be put to the related certificate blockchain to issue a certificate for a verified CVR.
- Characteristic 2: The interpretation of the certificates requires CAs to allow their database to be queried by the client systems. The PKQuery brings two obligations to the CAs. First, the CAs must keep all blockchain data that represent all certificates in the ecosystem stored in the CA infrastructure. The second obligation is that the CAs must protect the blockchain data that they store so that they can respond honestly to the incoming queries. Non-honest behavior causes a CA to be removed from the CA network.
- Characteristic 3: The subjects table in the ConsensusPKI ecosystem has a crucial role in bringing transparency to the certificates available in the cyberspace. A single query on the subjects table gives the following information:
 - the total number of subjects that a valid certificate is bound to
 - the total number of the subjects that a certificate is issued to
 - the total number of certificates issued to a subject for a given period
- Characteristic 4: The certificates in the ConsensusPKI do not carry signatures of the issuing CAs. The end user systems require certificates to carry the Merkle proof for the certificate to be interpreted correctly. The ConsensusPKI requires CAs to be honest during the certificate issuance, the PKQuery and the Fetch processes of the

ConsensusPKI. Behaving non-honest on any of the processes causes a CA to be removed from the blockchain network that leads the CA to not to be able to issue certificates. Removing a CA from the network at any time have only impact on the CA itself.

- Characteristic 5: A subject can have only one valid certificate at any given moment. The constraint that limits the number of valid certificates removes the possibility of consumption of a fake certificate.
- Characteristic 6: ConsensusPKI requires automation of the issuance and the deployment of the certificates. The APKME protocol ensures that the certificate is appropriately issued. It is again the task of the APKME to activate the certificate when the certificate is issued for the subject. The APKME also removes the possibility of human errors on CA infrastructure.
- Characteristic 7: The core data of the regular certificate blockchain consists of many hash values that have only importance until a certificate is expired. The matrix structure of the certificate blockchains verified by a root blockchain removes the necessity to keep the certificate blockchain data forever. Further, there is no public key kept in the certificate blockchains. The characteristics of the data structure help ConsensusPKI to be a scalable and sustainable solution. The total required data storage for the ConsensusPKI is minimal, yet the ecosystem aims to serve the entire cyberspace.
- Characteristic 8: The concept implementation of the ConsensusPKI demonstrated that even modest size hardware is capable of timely answering PKQuery requests. There is no doubt that a production scale hardware can serve many requests concurrently and the whole CA network can serve the entire cyberspace.

Figure 71 maps the relationship between the requirements indicated as R1 to R8 and the essential characteristics of the ConsensusPKI.

	R1	R2	R3	R4	R5	R6	R7	R8
<i>Characteristic 1</i>	X	X	X	X	X	X		
<i>Characteristic 2</i>	X	X	X	X		X		
<i>Characteristic 3</i>		X	X		X	X		
<i>Characteristic 4</i>		X	X	X		X	X	X
<i>Characteristic 5</i>	X	X	X		X	X	X	
<i>Characteristic 6</i>	X	X	X		X	X	X	
<i>Characteristic 7</i>			X				X	X
<i>Characteristic 8</i>							X	X

Figure 71: The characteristics and the requirements matrix of the ConsensusPKI

On the other hand, the primary requirement of the ConsensusPKI is internet connectivity. When it comes to local, WAN, or LAN deployments, where there is no internet connectivity, the rigorous nature of the ConsensusPKI will create challenges for administrators to maintain the local ConsensusPKI infrastructure. As a remedy, we propose the standardization of the following TDLs to be used for the applications without internet connectivity: a) .local b) .lan c) .wan. Once it is standardized, the indication and interpretation algorithm on the end user systems can distinguish that it must query the local CA repositories instead of the internet CA repositories. In that case, another set of CA root and certificate blockchains must be distributed and maintained by the administrators of the end user systems.

Another aspect for local deployment is the required minimum number of CAs that responds to PKQuery requests. It is required to have at least four CA responses for PKQuery to function. When administrators do not want to deploy four hosts in their local ConsensusPKI deployment, they have an option to create multiple virtual hosts on the same hardware to serve as a local CA.

Moreover, there are ten additional hash values, the Merkle tree nodes, stored in the certificates of the ConsensusPKI. The Merkle tree nodes are required to construct the Merkle root on indication and interpretation by the end-user systems. The additional ten hash values increase the certificate file size by 320 bytes if the SHA256 hash function is used. Additionally, the PKQuery adds 2464 bytes data exchange during the connection handshake (given RSA with 2048-bit keys is used). The total additional 2784 bytes bandwidth increase creates additional 2784 kB for every 1024 connection handshake (such as TLS). On the other hand, there are also many fields in the X.509 certificate files that have no use in ConsensusPKI certificates. Removing the unnecessary X.509 fields will reduce the required bandwidth decreasing the total required bandwidth. The total extra required bandwidth is a neglectable increase for today's bandwidth standards.

The core of the ConsensusPKI is shaped by a simple change in the definition of the certification authorities. The certification authorities in X.509 are entities that we explicitly or implicitly trust. However, a CA in the ConsensusPKI is an entity that never lies. All processes and the data model of the ConsensusPKI are built on this simple principle. Every process of the ConsensusPKI checks this very core principle and removes the parties that do not hold the core principle. If a CA lies in the ConsensusPKI even a single time to an individual client, the ecosystems make sure that this CA is dismissed as an authority.

Finally, to the best of our knowledge, the following structures are introduced to the literature by the ConsensusPKI:

- The use of the Merkle proofs instead of signatures in the certificate files
- The use of Practical Byzantines Fault Tolerance algorithm during interpretation of the certificates
- The use of leading root blockchain to manage the data retention for blockchains
- The use of consensus algorithms verifying identities to issue certificates

6.2) The impact of the ConsensusPKI on the Cyberspace

The conceptualization of the cyberspace by J. van den Berg et al. describes the complex ecosystem and their interdependencies as illustrated in Figure 72. The current X.509 standards as a technical measure sit in the core of the cyberspace. We heavily rely on good functioning PKI. The X.509 standards are the ultimate tool to guarantee the confidentiality and integrity of the services across the globe. The X.509 itself has a socio-technical environment and finally governance layer around it. Changing a core infrastructure such as the X.509 would impact the cyberspace drastically. As an example, the X.509 standard has a legal counter meaning. The signatures on the certificates are legally binding and equivalent to the hand signatures. On the other hand, there are no signatures in the certificates of the ConsensusPKI. Upon standardization, we will require to reassess the way that we look at the PKI services, evidence, and their legal meanings. The biggest challenge is not changing the core, but understanding what the change means, how we will consume and govern it. The impact of such a foundational change requires rigorous transdisciplinary

research, but this requires exploring the cyber-landscape further, and it is currently not in the scope of this research.

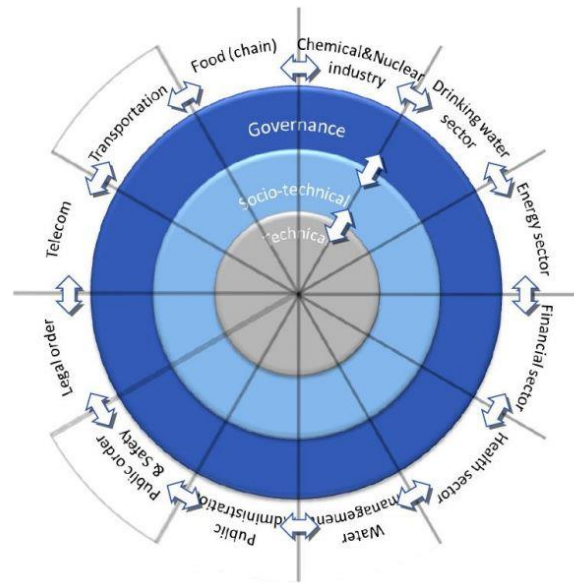


Figure 72: Conceptualization of the cyberspace

6.3) The shortcomings of the ConsensusPKI

The ConsensusPKI relies on currently secure cryptographic primitives. Many of the current cryptomographic primitives rely on prime factorization or discrete logarithm hard problems. It is expected that quantum computers would sharply reduce the time to solve the prime factorization or discrete logarithm problems. The invention of quantum computers would thereby impact, like all other currently used cryptosystems, the ConsensusPKI. On the other hand, the impact of the invention of the quantum computers on the hash functions is less drastic than primitives like RSA. However, the ConsensusPKI is not bound to a specific hash function or asymmetrical encryption primitive. As a countermeasure to the invention of the quantum computers, increasing the digest size of the hash function would satisfy the requirements for the hash functions for the Quantum era, but only if there exists a practical and fast enough quantum secure asymmetrical encryption and signing schemes.

Chapter 7) Conclusion

The ConsensusPKI demonstrated that the public blockchain supported data-driven PKI could solve all known structural flaws of the X.509 scheme. We have proposed a temper proof public blockchain based PKI ecosystem to be used across the cyberspace. The PBFT based consensus algorithms serve to detect non-honest or faulty CAs without the need to interrupt the end user communication. The ConsensusPKI has no trust and requires verification of the public keys each time it is interpreted. In ConsensusPKI, CAs are still required. However, they conduct a completely different role without explicit trust. The certificate issuance process of the ConsensusPKI is entirely transparent and rigorous verification process that guarantees that the requestors of the certificate issuance have full control over the subject. The constraint to have one valid certificate at any given moment guarantees that a fake certificate cannot be consumed, although it is very challenging to issue such certificate.

The most exciting aspect of the ConsensusPKI is that it replaces the signatures on the certificates with many verifiable pieces of evidence. The certificate issuance process in the ConsensusPKI requires the collection of evidence of multiple CA verifications of the identity of the subject and its public key. The certificate issuance process in ConsensusPKI guarantees that the participating CAs, during the issuance, are honest. As a result, revocation of CA certificate, after regular certificate issuance, has only on the CA itself.

Furthermore, the ConsensusPKI proposes full segregation of the internet and non-internet applications be deployed to use with a combination of the ConsensusPKI ecosystem. Constraining to use internet domains for non-internet applications help to minimize the change of misinterpretation of the certificates on end-user systems. The CA certificate store on the end user systems is protected using blockchain to remove the possibility of tempering root certificate store on end-user systems. CA certificate store based on blockchain is alone a vast improvement compares to in X.509 standard. Even though there exists a CA certificate store at the end user systems, the responses of the CAs as a reply to PKQuery requests are challenged with each other to guarantee certainty of a certificate, and thus confidentiality and integrity of communication in the cyberspace.

Finally, the success of the ConsensusPKI is entirely dependent on the CAs, browser, and OS vendors. Standardization of the ConsensusPKI requires a commitment of many different stakeholders, who must work in harmony to improve the protection of the confidentiality and integrity of the communications across the cyberspace.

Chapter 8) Further study

Although we have proposed a comprehensive PKI ecosystem. The following researches would enhance and test our solution as a whole.

- Full implementation of the ConsensusPKI tamarin models to test the all underlying protocols
- Full implementation of the ConsensusPKI ecosystem including browser support to test it on real-life scenarios
- Research on the impact of the ConsensusPKI, upon standardization
- Research on the quantum computer safe ConsensusPKI

List of Figures

Figure 1: Relationship between the structural flaws and the pillars of the X.509 ecosystem	13
Figure 2: Summary of structural flaws of X.509 ecosystem	14
Figure 3: Information system research framework by Hevner et al.	17
Figure 4: Comparison of alternative solutions against the requirements of the thesis	22
Figure 5: Probability distribution of a ZKP system	25
Figure 6: DNS query response for TXR records of tudelft.nl	27
Figure 7: Example DKIM header included in an email	29
Figure 8: A Merkle Tree	37
Figure 9: ConsensusPKI landscape	43
Figure 10: Structure of the Root Blockchain	45
Figure 11: Example Certificate Blockchain for a year N	45
Figure 12: Example Evidence Blockchain for a CVR	45
Figure 13: Certificate Blockchain Merkle Tree	47
Figure 14: Certificates Merkle Tree	49
Figure 15: Evidence Blockchain and Merkle Tree	50
Figure 16: Single Domain and Public Key Verification	55
Figure 17: Fetch CA Certificate Blockchains	59
Figure 18: Pseudocode to create the ca_rootblockchain table	63
Figure 19: Pseudocode to create the ca_certificateblockchain table	63
Figure 20: Example dataset on ca_certificateblockchain table	64
Figure 21: Pseudocode to create the ca_subjects table	64
Figure 22: Pseudocode to create the vw_ca view	64
Figure 23: Example dataset on vw_ca view including the year and height of the blocks	64
Figure 24: An example CA Certificate used in the prototype	65
Figure 25: An example dataset in the subjects table	66
Figure 26: Pseudocode to create the certificateblockchain table	66
Figure 27: Pseudocode to create the vw_response in MariaDB	66
Figure 28: Pseudocode to create the ca_evidenceblockchain table	67
Figure 29: An example Regular Certificate used in the prototype	68
Figure 30: The python code to construct the Merkle root from a Merkle Proof	68
Figure 31: An example certificate Merkle tree	69
Figure 32: Intended execution steps and the States of PKQuery	71
Figure 33: Pseudocode to retrieve Ordered Subjects in a single data block	72
Figure 34: Pseudocode to calculate the Blockheader	72
Figure 35: Block consistency check on certificate blockchain	72
Figure 36: Blockheader calculation and integrity check pseudocodes for CA_certificateblockchain	72
Figure 37: Pseudocode for cross-validation among certificate blockchains	73
Figure 38: Pseudocode to calculate a blockheader starts with tree leading zeros '000'	73
Figure 39: Example execution of PoW	74
Figure 40: The generated record set for one subject	79
Figure 41: Example block data for the subject	79
Figure 42: Generated files using the certificate generation tool	81
Figure 43: Subjects table record set for www.google.com after certificate generation	81
Figure 44: certificateblockchain record for the generated certificate	81
Figure 45: An example generated certificate file with proof	82

Figure 46: An example screenshot of the certificate generator	82
Figure 47: A Signed CVR file to send to a CA to initiate the certificate issuance process	83
Figure 48: An example screen print of the CA PKQuery server with a valid response	84
Figure 49: An example screen print of the CA PKQuery server with an empty response	84
Figure 50: An example execution of the PKQuery indicating certificate as valid	86
Figure 51: Simulation results for Case 1	86
Figure 52: Simulation results for Case 2	87
Figure 53: simulation results for Case 3	87
Figure 54: Simulation results for Case 4	88
Figure 55: Overall results of the simulation	89
Figure 56: lemma Client_subject_secrecy for PKQuery	90
Figure 57: lemma Client_nonce_secrecy for PKQuery	90
Figure 58: lemma message_authentication for PKQuery	90
Figure 59: lemma Client_Authenticity for PKQuery	90
Figure 60: lemma Client_Authenticity_Injective for PKQuery	91
Figure 61: Constraint system for message_authentication	91
Figure 62: Solution of constraint system for message_authentication	91
Figure 63: Constraint system for Client_subject_secrecy	92
Figure 64: Solution of constraint system for Client_subject_secrecy	92
Figure 65: Constraint system for Client_nonce_secrecy:	92
Figure 66: Solution of the constraint system for Client_nonce_secrecy	93
Figure 67: Constraint system for the client authenticity	93
Figure 68: Solution of the constraint system for the client authenticity	94
Figure 69: Constraint system for the Client_Authenticity_Injective	94
Figure 70: Solution of the constraint system for the Client_Authenticity_Injective	95
Figure 71: The characteristics and the requirements matrix of the ConsensusPKI	97
Figure 72: Conceptualization of the cyberspace	99

Bibliography

- [1] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley and W. & Polk, Internet X. 509 public key infrastructure certificate and certificate revocation list (CRL) profile (No. RFC 5280), No. RFC 5280, 2008.
- [2] J. R. Prins and C. B. Unit, "Diginotar certificate authority breach'operation black tulip," Fox-IT, 2011.
- [3] N. S. van Der Meulen, "DigiNotar: Dissecting the First Dutch Digital Disaster," *Journal of Strategic Security*, vol. 6, no. 2, pp. 46-58, 2013.
- [4] S. B. Roosa and S. Schultze, "Trust darknet: Control and compromise in the internet's certificate authority model," *IEEE Internet Computing*, vol. 17, no. 3, pp. 18-25, 2013.
- [5] J. Klark and P. C. van Oorschot, "SoK: SSL and HTTPS: Revisiting past challenges and evaluating certificate trust model enhancements," *IEEE Symposium on Security and Privacy*, pp. 511-525, 2013.
- [6] C. Evans, C. Palmer and R. Sleevi, "Public key pinning extension for HTTP," (No. RFC 7469), 2015.
- [7] B. Laurie, A. Langley and E. & Kasper, "Certificate transparency," No. RFC 6962, 2013.
- [8] P. Hallam-Baker and R. Stradling, "DNS certification authority authorization (CAA) resource record," No. RFC 6844, 2013.
- [9] I. Dubrawsky, *How to cheat at securing your network*, Syngress, 2011.
- [10] E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi and B. Ford, "Keeping Authorities "Honest or Bust" with Decentralized Witness Cosigning," *Security and Privacy (SP)*, vol. 2016, no. IEEE Symposium, pp. 526-545, 2016.
- [11] Y. Gilad, A. Herzberg and H. Shulman, "Off-path hacking: The illusion of challenge-response authentication," *IEEE Security & Privacy*, vol. 12, no. 5, pp. 68-77, 2014.
- [12] Z. Durumeric, Z. Ma, D. Springall, R. Barnes, N. Sullivan, E. Bursztein, M. Bailey, J. A. Halderman and V. Paxsonk, "The Security Impact of HTTPS Interception," in *Proc. Network and Distributed System Security Symposium (NDSS)*, 2017.
- [13] C. Paar and J. Pelzl, *Introduction to public-key cryptography*, Berlin, Heidelberg: Springer, 2010.
- [14] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.
- [15] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance," in *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, New Orleans, USA, 1999.

- [16] L. Lamport, R. Shostak and M. Pease, "The Byzantine generals problem," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 4, no. 3, pp. 382-401, 1981.
- [17] A. R. Hevner, S. T. March, J. Park and S. Ram, "Design Science in Information Systems Research," *MIS Quarterly*, vol. 28, no. 1, pp. 75-105, 2004.
- [18] D. Basin, C. Cremers, T. H. J. Kim, A. Perrig, R. Sasse and P. Szalachowski, "ARPKI: Attack resilient public-key infrastructure," in *ACM SIGSAC Conference on Computer and Communications Security*, 2014.
- [19] M. Abadi, A. Birrell, I. Mironov, T. Wobber and Y. Xie, "Global Authentication in an Untrustworthy World," *HotOS*, 2013.
- [20] P. Eckersley and J. Burns, "The EFF SSL Observatory," EFF, 2010. [Online]. Available: <https://www.eff.org/observatory>. [Accessed 16 09 2018].
- [21] P. Eckersley, "Sovereign Key Cryptography for Internet Domains," 2011. [Online]. Available: <https://github.com/EFForg/sovereign-keys/blob/master/sovereign-key-design.txt>. [Accessed 16 09 2018].
- [22] T. H. J. Kim, L. S. Huang, A. Perrig, C. Jackson and V. Gligor, "Accountable key infrastructure (AKI): a proposal for a public-key validation infrastructure.," in *Proceedings of the 22nd international conference on World Wide Web, ACM*, pp. 679-690, 2013.
- [23] M. D. Ryan, "Enhanced Certificate Transparency and End-to-End Encrypted Mail," *NDSS*, 2014.
- [24] M. S. Melara, A. Blankstein, J. Bonneau, E. W. Felten and M. J. Freedman, "CONIKS: Bringing Key Transparency to End Users," *USENIX Security Symposium*, vol. 2015, pp. 383-398, 2015.
- [25] J. Yu, V. Cheval and M. Ryan, "DTKI: A new formalized PKI with verifiable trusted parties," *The Computer Journal*, vol. 59, no. 11, pp. 1695-1713, 2016.
- [26] C. Fromknecht, D. Velicanu and S. Yakoubov, "Certcoin: A namecoin based decentralized authentication system," *Massachusetts Inst. Technol., Cambridge, MA, USA*, vol. Tech. Rep., no. 6, 2014.
- [27] B. Leiding, C. H. Cap, T. Mundt and S. Rashidibajgan, "Authcoin: validation and authentication in decentralized networks," *arXiv preprint arXiv:1609.04955*, 2016.
- [28] S. Matsumoto and R. M. Reischuk, "IKP: Turning a PKI around with decentralized automated incentives," in *IEEE Symposium on Security and Privacy, IEEE*, 2017, May.
- [29] B. Qin, J. Huang, Q. Wang, X. Luo, B. Liang and W. Shi, "Cecoin: A decentralized PKI mitigating MitM attacks," *Future Generation Computer Systems*, 2017.
- [30] J. Chen, S. Yao, Q. Yuan, K. He, S. Ji and R. Du, "CertChain: Public and Efficient Certificate

Audit Based on Blockchain for TLS Connections," 2018.

- [31] F. B., "A Distributed X.509 Public Key Infrastructure Backed by a Blockchain," 2018.
- [32] X. Wang and H. Yu, "How to Break MD5 and Other Hash Functions," in *Annual international conference on the theory and applications of cryptographic techniques*, Berlin, Heidelberg, 2005.
- [33] W. Mehuron, "Digital Signature Standard (DSS)," *US Department of Commerce, National Institute of Standards and Technology (NIST), Information Technology Laboratory (ITL), FIPS PEB 186*, 1994.
- [34] T. Pornin, "Deterministic usage of the digital signature algorithm (DSA) and elliptic curve digital signature algorithm (ECDSA)," (*No. RFC 6979*), 2013.
- [35] A. Juels and J. G. Brainard, "Client puzzles: A Cryptographic countermeasure against connection depletion attacks," *NDSS*, vol. 99, pp. 151-165, 1999.
- [36] B. Groza and D. Petrica, "On chained cryptographic puzzles. In (pp. 25-26).," in *3rd Romanian-Hungarian Joint Symposium on Applied Computational Intelligence (SACI)*, Timisoara, Romania , 2006, May.
- [37] O. Goldreich and Y. Oren, "Definitions and properties of zero-knowledge proof systems," *Journal of Cryptology*, pp. 1-32, 1 7 1994.
- [38] P. Mockapetris and K. J. Dunlap, "Development of the domain name system," *ACM*, vol. 18, no. 4, pp. 123-133, 1988.
- [39] L. Dostálek and A. Kabelová, *DNS in action a detailed and practical guide to DNS implementation, configuration, and administration*, Birmingham, U.K.: Packt Pub., 2006.
- [40] J. Postel, *Domain Name System Structure and Delegation*, No. RFC 1591, 1994.
- [41] S. Cheshire and M. Krochmal, "DNS-based service discovery," (*No. RFC 6763*), 2013.
- [42] Google Inc., "Ownership verification," Google, 2018. [Online]. Available: https://support.google.com/webmasters/answer/9008080?hl=en&ref_topic=7440006. [Accessed 13 09 2018].
- [43] D. Crocker, T. Hansen and M. Kucherawy, "DomainKeys Identified Mail (DKIM) Signatures," (*No. RFC 6376*), 2011.
- [44] CA / Browser Forum, "Ballot 193 – 825-day Certificate Lifetimes," [Online]. Available: <https://cabforum.org/2017/03/17/ballot-193-825-day-certificate-lifetimes/>. [Accessed 05 Oct 2018].
- [45] R. Oppliger, *SSL and TLS: Theory and Practice*, Artech House, 2016.
- [46] S. Galperin, S. Santesson, M. Myers, A. Malpani and C. Adams, "X. 509 Internet public key infrastructure online certificate status protocol-OCSP," *No. RFC 6960*, 2013.

- [47] J. R. Vacca, *Cyber security and IT infrastructure protection*, Syngress, 2013.
- [48] V. Gramoli, "From blockchain consensus back to byzantine consensus," *Future Generation Computer Systems*, 2017.
- [49] F. Saleh, *Blockchain without waste: Proof-of-stake*, 2018.
- [50] P. Szalachowski, "Towards More Reliable Bitcoin Timestamps," *arXiv preprint arXiv:1803.09028*, 2018.
- [51] S. D. Mohanty, "Ordered Merkle Tree-a versatile data-structure for security kernels," *Doctoral dissertation, Mississippi State University*, 2013.
- [52] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," *OSDI*, vol. 99, pp. 173-186, 1999.
- [53] R. Barnes, J. Hoffman-Andrews and J. Kasten, "Automatic certificate management environment (acme)," *IETF Draft*, 2017.
- [54] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on information theory*, vol. 29, no. 2, pp. 198-208, 1983.
- [55] S. Meier, B. Schmidt, C. Cremers and D. Basin, "The TAMARIN prover for the symbolic analysis of security protocols," in *International Conference on Computer Aided Verification*, Berlin, Heidelberg, (2013, July).
- [56] D. Crockford, "The application/json media type for javascript object notation (json)," *No. RFC 4627*, 2006.
- [57] H. Zimmermann, "OSI reference model. The ISO model of architecture for open systems interconnection.," *IEEE Transactions on communications*, vol. 28, no. 4, pp. 425-432, 1980.
- [58] R. Canetti, O. Goldreich and S. Halevi, "The random oracle methodology, revisited," *Journal of the ACM (JACM)*, vol. 51, no. 4, pp. 557-594, 2004.
- [59] Z. Durumeric, J. Kasten, M. Bailey and J. A. Halderman, "Analysis of the HTTPS certificate ecosystem," *In Proceedings of the 2013 conference on Internet measurement conference ACM*, pp. 291-304, 2013, October.
- [60] AmazonAWS, "Alexa Top 1 Million," Alexa, [Online]. Available: <http://s3.amazonaws.com/alexastatic/top-1m.csv.zip>. [Accessed 15 September 2018].
- [61] J. Van den Berg, J. Van Zoggel, M. Snels, M. Van Leeuwen, S. Boeke, L. van de Koppen, J. v. d. B. B. van der Lubbe and d. B. T., "On (the Emergence of) Cyber Security Science and its Challenges for Cyber Security Education," in *In The NATO IST-122 Cyber Security Science and Engineering Symposium*, Tallin, 2014.

Appendix A: An example output for the simulation runs

Example client output for case 1:

```
Command Prompt - python PKQ_RSA_ClientVEncV3.py
Please Enter The Subject : www.google.com
Please Enter Certificate Filename : www.google.com1544532406805CertificateFileWithPublicKeyRSAWithProof.jcrt
PKQuery check is started. The time in milliseconds: 1544627224678
h(subject) + MR from Certificate :A1104809E6320C076766996E77DF04C40C4892B6C1CC8C7CF731F2D542DA6A41
The HSubject: 191347BFE55D0CA9A574DB778C8648275CE258461450E793528E0CC6D2DCF8F5
Randomly selected client nonce: 8dde1686542f88b9
The expected response calculated using the proof in the certificate: 983062326FA213D1269AE98660732393E073799F3C96EEA45FFF427A58FF5D63
The start of the PKQuery threads. The time in milliseconds: 1544627224689
Thread start. The time in milliseconds: 1544627224690
Thread start finished. The time in milliseconds: 1544627224699
The end of the PKQuery threads. The time in milliseconds: 1544627224795
The start of the validation checks. The time in milliseconds: 1544627224795
983062326FA213D1269AE98660732393E073799F3C96EEA45FFF427A58FF5D63
983062326FA213D1269AE98660732393E073799F3C96EEA45FFF427A58FF5D63
983062326FA213D1269AE98660732393E073799F3C96EEA45FFF427A58FF5D63
88889999E2B22ECC2A39AD85A2535EA1D5D15103E48B9D8938149F444439590
Signature Verifications are OK. The time in milliseconds: 1544627224796
All Responses are not equal.
Non-Honest CA Detected.
Non-Honest-CAs and Exchanged Messages:
deque([b'8dde1686542f88b9191347BFE55D0CA9A574DB778C8648275CE258461450E793528E0CC6D2DCF8F5', '88889999E2B22ECC2A39AD85A2535EA1D5D15103E48B9D8938149F444439590', ('node9.consensuspki.org', 5509), b'88889999E2B22ECC2A39AD85A2535EA1D5D15103E48B9D8938149F444439590', b'\x1004\x4b45b\x8cW\xce\x99\xcc\x3j\x90\x90\x16<\xb8\xd4,\xb9b\xe4w'\xed \x16g\x1c\xa5D\x18\xd2\x5d5<\&\x82(\xd71\x1b\x0b\xeeN\xfb\xeb*\xd3\xce0\x93\x19\xdeU\xa4\x8b\x5b\x8aK\xed\xcc0\x15j\xed\xa6?\xd9\x03^\xe8\x8a\x1c\x1c2\xee4\xd9\xd6,\xf7f\xa40V\x02\xe2Z\xdc0\xfb0\x7f\xecH\x9be\xfbI\x9e\x07\x06k\xa1\x96\x8a*Eq\xed\xfd\x1c2\x02r9V\xcd\x1bw1\x16\xfa\x9c\x83\xdf\xfa\x080\x15\x9c\xce\xdd4\xef\xddz\xec\x10\x94\xed\x8c\x16\xce\xce\xfb\x93\xee-G\x8b8j(\xa0\xa17\x1b\x16\x1c3\x1n\x1b4\xee\x15\x1c3\x1d9\x9d)\xab4\x7f\xa3\x9c\xce\xdd4\xef\xcd\Z\xdb\x110\x15M\x17+\x11\x845U\x88\x99\x03\xa98\x89\xea\x9\xeb\x8c'#\xcd\x1d9o\x06v\xa7n\x9aq\x0b8Q\x99\xbe\xfb6\xef\x9c\x7f\x9e\x16\x08\x10\x1f\x1e5\xcc\x1e2\xfd[\xb9f\x1d8\x0b\xdf\x1e5\x90V\xb2u\xfb0\x1d2\x1b7,\xf8\x1c7y\xfeZ\r:-\x1e" ]])
Connection is secure Certificate matches the Accepted response : 983062326FA213D1269AE98660732393E073799F3C96EEA45FFF427A58FF5D63
Detected Faulty response : 88889999E2B22ECC2A39AD85A2535EA1D5D15103E48B9D8938149F444439590
The PKQuery check is finalized. The time in milliseconds: 1544627224797
```

```
Command Prompt - python PKQ_RSA_ClientVEncV3.py
Please Enter The Subject : www.google.com
Please Enter Certificate Filename : www.google.com1544532406805CertificateFileWithPublicKeyRSAWithProof.jcrt
PKQuery check is started. The time in milliseconds: 1544627428892
h(subject) + MR from Certificate :A1104809E6320C076766996E77DF04C40C4892B6C1CC8C7CF731F2D542DA6A41
The HSubject: 191347BFE55D0CA9A574DB778C8648275CE258461450E793528E0CC6D2DCF8F5
Randomly selected client nonce: 36843254076251a0
The expected response calculated using the proof in the certificate: 033EA7BB30DB31B99C08572922062AD797920B2AE0F1E48F3D25BE5C2D6C053F
The start of the PKQuery threads. The time in milliseconds: 1544627428896
Thread start. The time in milliseconds: 1544627428897
Thread start finished. The time in milliseconds: 1544627428904
The end of the PKQuery threads. The time in milliseconds: 1544627428996
The start of the validation checks. The time in milliseconds: 1544627428996
033EA7BB30DB31B99C08572922062AD797920B2AE0F1E48F3D25BE5C2D6C053F
033EA7BB30DB31B99C08572922062AD797920B2AE0F1E48F3D25BE5C2D6C053F
033EA7BB30DB31B99C08572922062AD797920B2AE0F1E48F3D25BE5C2D6C053F
033EA7BB30DB31B99C08572922062AD797920B2AE0F1E48F3D25BE5C2D6C053F
033EA7BB30DB31B99C08572922062AD797920B2AE0F1E48F3D25BE5C2D6C053F
Signature Verifications are OK. The time in milliseconds: 1544627429000
All Responses are equal.
Connection is secure Certificate matches the Accepted response : 033EA7BB30DB31B99C08572922062AD797920B2AE0F1E48F3D25BE5C2D6C053F
The PKQuery check is finalized. The time in milliseconds: 1544627429001
```

Example client output for case 2:

```
Command Prompt - python PKQ_RSA_ClientVEncV3.py
Please Enter The Subject : www.google.com
Please Enter Certificate Filename : www.google.com1544531569490CertificateFileWithPublicKeyRSAWithProof.jcrt
PKQuery check is started. The time in milliseconds: 1544628659557
h(subject) + MR from Certificate :01CAF0175DE4121C4A6312863EDBFC2DF1AD07681D860CCFFFAEE25BE08546E1
The HSubject: 191347BFE55D0CA9A574DB778C8648275CE258461450E793528E0CC6D2DCF8F5
Randomly selected client nonce: 24ae6694bf8fdca1
The expected response calculated using the proof in the certificate: DA32E885D94F5759F78BDAF56B391983480B504FF5E621D975B13C60E54A0A8
The start of the PKQuery threads. The time in milliseconds: 1544628659565
Thread start. The time in milliseconds: 1544628659565
Thread start finished. The time in milliseconds: 1544628659571
The end of the PKQuery threads. The time in milliseconds: 1544628659664
The start of the validation checks. The time in milliseconds: 1544628659664
72DF65837237C69A8DFBFD29B7206F006E6DC46AB5A4747DF2DE1325111FCA25
72DF65837237C69A8DFBFD29B7206F006E6DC46AB5A4747DF2DE1325111FCA25
72DF65837237C69A8DFBFD29B7206F006E6DC46AB5A4747DF2DE1325111FCA25
72DF65837237C69A8DFBFD29B7206F006E6DC46AB5A4747DF2DE1325111FCA25
Signature Verifications are OK. The time in milliseconds: 1544628659668
All Responses are equal.
Connection is Not secure Cert does not matches the response. The Accepted response from the CAs : 72DF65837237C69A8DFBFD29B7206F006E6DC46AB5A4747DF2DE1325111FCA25
The Expected response from the Certificate: DA32E885D94F5759F78BDAF56B391983480B504FF5E621D975B13C60E54A0A8
The PKQuery check is finalized. The time in milliseconds: 1544628659670
```


Example client output for case 3:

```
Command Prompt - python PKQ_RSA_ClientVEncV3.py
Please Enter The Subject : www.nos.nl
Please Enter Certificate Filename : www.google.com1544532406805CertificateFileWithPublicKeyRSAWithProof.jcrt
PKQuery check is started. The time in milliseconds: 1544627549731
h(subject) + MR from Certificate :D19D4F89FC054F06040950F00350A22E9320A8E99AF12B0BAFFD1099F05D96C2
The HSubject: 2251395047A865C19AD7204D19FC2E689869196586387F5498FEFB6AEA432635
Randomly selected client nonce: 5898e90c27ff3e6a
The expected response calculated using the proof in the certificate: 08895E71A52ECB58E68FBBA9C9248553BF9DD8CF98E647D1ADAB86A6DF98DE38
The start of the PKQuery threads. The time in milliseconds: 1544627549732
Thread start. The time in milliseconds: 1544627549732
Thread start finished. The time in milliseconds: 1544627549736
The end of the PKQuery threads. The time in milliseconds: 1544627549839
The start of the validation checks. The time in milliseconds: 1544627549839
88889999E2B22ECC2A39AD85A2535EA1D5D15103E48B96D08938149F444439590
47F035FFC9283D2B34CFE225FF56F0E18C0563ECEDA1298DFADEA3AB40F967E
47F035FFC9283D2B34CFE225FF56F0E18C0563ECEDA1298DFADEA3AB40F967E
47F035FFC9283D2B34CFE225FF56F0E18C0563ECEDA1298DFADEA3AB40F967E
Signature Verifications are OK. The time in milliseconds: 1544627549841
All Responses are not equal.
Non Honest CA Detected.
Non-Honest-CAs and Exchanged Messages:
deque([[b'5898e90c27ff3e6a2251395047A865C19AD7204D19FC2E689869196586387F5498FEFB6AEA432635', '88889999E2B22ECC2A39AD85A2535EA1D5D15103E48B96D08938149F444439590', ('node9.consensuspki.org', 5509), b'88889999E2B22ECC2A39AD85A2535EA1D5D15103E48B96D08938149F444439590', b'\x1004\x45b\x8cW\xce\x9\xcc\x3f\x90\x90\x16<\xb8\xd4,\x9b\x4e4w\xed\x16g\x1c\x50\x18\xd2\xd5<&\x82(\xd7l\x1b\x0b\xeeN\xfb\xeb*\xd3\xce0\x93\x19\xde0\x4\x8b\x5\x8akT\xed\x0\x15J\x4\x6a6?\xd9\x03*\xe8\x8a\x1c\x2\xee4\xd9\xd6,\x7f\xa40V\x02\xe22\xdc0\xfb7\xecH\x9be\xfbf\x9e\x07\x06k\xa1\x96\x8a*Eq\xd7\xfb1\x2\x02r9V\xcd\x1bw1\x16\xfa\x9\x83\xdf\xfa\x080\x15\x9c\xce\xdd4\xef\xddz\xec\x23\x94\x7\x8c8i16\xce\xec\xfb\xa3\xee~G\x8bJ(\xa0\xa17\x1b\x16\x16\x3\n\x4\xee\x5\xa3\x9d\x9d)\xab4\x7\xa3\x9c\x9c\x9b9\xcbV|dZ\xdb\x110\x15M\x17+\x11\x845U\x88\x99\x03\xa98\x89\xea\x9\xeb\x8c8#\xcd\x90o\x06v\xa7n\x9aq\x0b8Q\x99\xbe\x6\xef\x9c\x7f\x9e\x61\x081\xfb1\x5\xcc\x2\xfb3[\xb9f\x8d\x0b\xdf\x5\x90V\x2u\xfb8\x3\xd2\x7,\xf8\x7y\xfe7\r:~\x1e']]])
Connection is Not secure Cert does not matches the response. The Accepted response from the CAs : 47F035FFC9283D2B34CFE225FF56F0E18C0563ECEDA1298DFADEA3AB40F967E
The Expected response from the Certificate: 08895E71A52ECB58E68FBBA9C9248553BF9DD8CF98E647D1ADAB86A6DF98DE38
Detected Faulty response : 88889999E2B22ECC2A39AD85A2535EA1D5D15103E48B96D08938149F444439590
The PKQuery check is finilized. The time in milliseconds: 1544627549847
```

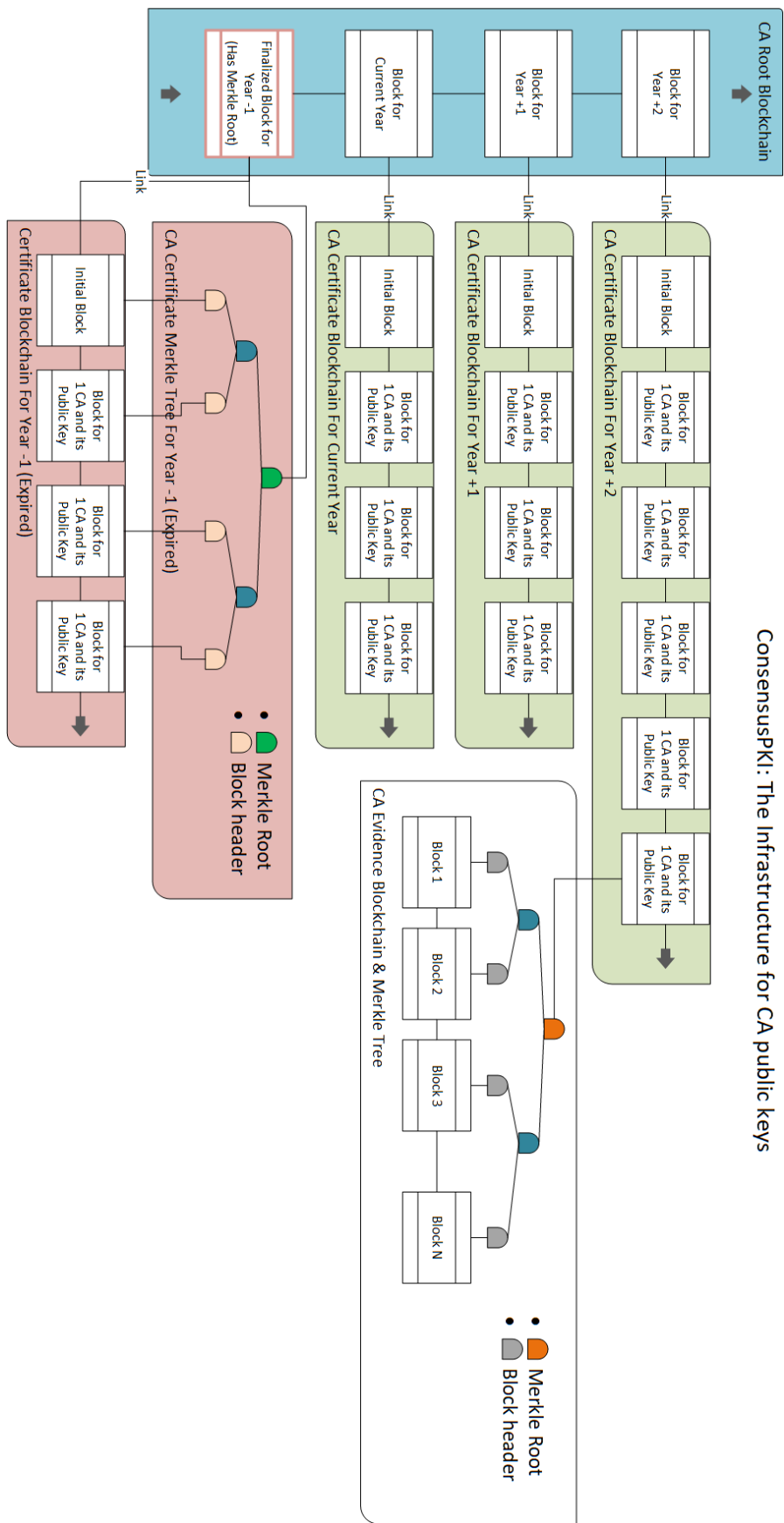
Example client output for case 4:

```
Command Prompt - python PKQ_RSA_ClientVEncV3.py
Please Enter The Subject : test.nodomain
Please Enter Certificate Filename : www.google.com1544532406805CertificateFileWithPublicKeyRSAWithProof.jcrt
PKQuery check is started. The time in milliseconds: 1544628538566
h(subject) + MR from Certificate :3D5E0A52EA4A43AE6E5C8CEDAD502753866B176B23A94099C54A7203187D445
The HSubject: 0C57EB5D44A9A0AE294F53608D12A4DD5682962FDCE5EC842A3866EF3D0FA439
Randomly selected client nonce: ec43c31564c7383b
The expected response calculated using the proof in the certificate: E513EF6FD978B9DF7EA66192E94DD47D896E9BC00BEC5ADC78DFDFE839355186
The start of the PKQuery threads. The time in milliseconds: 1544628538570
Thread start. The time in milliseconds: 1544628538570
Thread start finished. The time in milliseconds: 1544628538578
The end of the PKQuery threads. The time in milliseconds: 1544628538651
The start of the validation checks. The time in milliseconds: 1544628538652
8995CDA4BDF461F0A063A5D80EDF024093B2AB596B40E6C19307D242C71A18EA
8995CDA4BDF461F0A063A5D80EDF024093B2AB596B40E6C19307D242C71A18EA
8995CDA4BDF461F0A063A5D80EDF024093B2AB596B40E6C19307D242C71A18EA
77779999E2B22ECC2A39AD85A2535EA1D5D15103E48B96D08938149F444439590
Signature Verifications are OK. The time in milliseconds: 1544628538655
All Responses are not equal.
Non Honest CA Detected.
Non-Honest-CAs and Exchanged Messages:
deque([[b'ec43c31564c7383b0C57EB5D44A9A0AE294F53608D12A4DD5682962FDCE5EC842A3866EF3D0FA439', '77779999E2B22ECC2A39AD85A2535EA1D5D15103E48B96D08938149F444439590', ('node10.consensuspki.org', 5510), b'77779999E2B22ECC2A39AD85A2535EA1D5D15103E48B96D08938149F444439590', b'\x18\x1a\x38~\xb8\x89\x0f\xaf2\xbf\x17\xde0\xdfh/30\x1c1\x3\x1a\x8\x80c\xfb7\xcePj1\xa8~\x9a\x8a\x14c\xfb6\tw\xce&\x0g\x5\xa8\x818\x7i\x17?\xa0\xec5\x1e\x82\x8d7\xec\x96\x92G\x9b9\\x1d\xa5#~m\xad\x99yK8\x4v\xa8\x8b\xfb\x5cY\x1f13\x8I\xfb5\x8f\x83\x0bf\x9e3\xa8\xa6bo\x96\xad01r~\xc2~\xe9z~Q\x86\x82\xdc\xfa9\x08\xea\x8d\xda\x1d)\x1\xdfT\xbe\x91\xde\xad\xa0\x92\xbb\x2\xae\xdd0~%701u\x00g\xafh\xfbfY\x34\x1c664\x00\x9\xfb5\xfb1\xfb1~w*P\xfb3\x2\x0d7\xfb3\x18\x6e)\xa5\xaf\xa8\xba\x11\x5\xfb6)\x81k\x7b3\x168\xefp\x0c\x1a\x4\xfe\xfb9\x1b\xdd\xdd\x88\xda\x01\x8b\x5\x9d\\x8b\x8c\xda)3KG\x08-T\x96\xfb1j\x0e3\x7c7\x7f\xef0\x0c\x1eb\xa7\xec\x10a~c7~\x918\xbc7Y\xabj'\x8e\xa6\x98\xca\x3\x81\xfb2\x92~']]])
Connection is Not secure Cert does not matches the response. The Accepted response from the CAs : 8995CDA4BDF461F0A063A5D80EDF024093B2AB596B40E6C19307D242C71A18EA
The Expected response from the Certificate: E513EF6FD978B9DF7EA66192E94DD47D896E9BC00BEC5ADC78DFDFE839355186
Detected Faulty response : 77779999E2B22ECC2A39AD85A2535EA1D5D15103E48B96D08938149F444439590
The PKQuery check is finilized. The time in milliseconds: 1544628538660
```

Example client output when there is no CA consensus:

```
Command Prompt - python PKQ_RSA_ClientVEncV3.py
Please Enter The Subject : www.google.com
Please Enter Certificate Filename : www.google.com1544532406805CertificateFileWithPublicKeyRSAWithProof.jcrt
PKQuery check is started. The time in milliseconds: 1544628998618
h(subject) + MR from Certificate :A1104809E6320C076766996E77DF04C40C4892B6C1CC8C7CF731F2D542DA6A41
The HSubject: 191347BFE55D0CA9A574DB77BC8648275CE258461450E793528E0CC6D2DCF8F5
Randomly selected client nonce: ef99f71f7dbd18db
The expected response calculated using the proof in the certificate: 8A28FA0A14739B29C49F98048B6F30C0D0E7CA60A2DD49D6E1091A291A6B6612
12
The start of the PKQuery threads. The time in milliseconds: 1544628998622
Thread start. The time in milliseconds: 1544628998622
Thread start finished. The time in milliseconds: 1544628998630
The end of the PKQuery threads. The time in milliseconds: 1544628998706
The start of the validation checks. The time in milliseconds: 1544628998706
88889999E2B22ECC2A39AD85A2535EA1D5D15103E48B96D08938149F444439590
8A28FA0A14739B29C49F98048B6F30C0D0E7CA60A2DD49D6E1091A291A6B6612
8A28FA0A14739B29C49F98048B6F30C0D0E7CA60A2DD49D6E1091A291A6B6612
77779999E2B22ECC2A39AD85A2535EA1D5D15103E48B96D08938149F444439590
Signature Verifications are OK. The time in milliseconds: 1544628998711
All Responses are not equal.
Multiple Non Honest CA Detected connection is not secure
Non-Honest-CAs and Exchanged Messages:
deque([[b'ef99f71f7dbd18db191347BFE55D0CA9A574DB77BC8648275CE258461450E793528E0CC6D2DCF8F5', '88889999E2B22ECC2A39AD85A2535EA1D5D15103E48B96D08938149F444439590', ('node0.consensuspki.org', 5500), b'88889999E2B22ECC2A39AD85A2535EA1D5D15103E48B96D08938149F444439590', b'8'x100j4\xb45b\8cW\Xce\xd9\Xcc\Xa3j\X00\X90\X16<\xb8\Xd4,\X9b\XedW\Xed_\X16g\X1c\Xa5D\X18\Xd2\Xd5<8\X82(\Xd7\X1b\X0b\XeeM\Xfb\Xeb*\Xd3\Xce0\X93\X19\XdeU\Xa4\X8b\Xb5\X8aK\XedX\Xc0\X15j\Xea\Xa6? \Xd9\X03^\Xeo\X8a\X1c\Xc2\Xee4\Xd9\Xd6,\X7f\Xa0U\X02\Xe2\Xdc0\Xf0\Xb7\XecH\X9be\XfbI\X9e\X07\X06k\Xa1\X96\X8a^Eq\Xd7\Xf1\Xc2\X02r9V\Xcd\X1bw1\X16\Xfa\Xc9\X83\Xdf\Xfa\X80\X15\X9c\Xce\Xdd4\Xef\Xdd2\Xec\Xd3\X94\Xe7\Xc8:1f6\Xce\Xec\Xfb\Xa3\Xee~G\Xb8j(\Xa0\Xa17\X1b\X16\Xc3\N\Xb4\Xee\Xb5\Xa3\Xd9\X9d)\Xab4\Xb7\Xa3\Xc9\Xb0\Xb9\XcbV\Z\Xdb\X11@ \X15M\X17+\X11\X845U\X88\X99\X03\Xa9&\X89\Xea\X9e\Xeb\Xc8'#\Xcd\Xd9o\X06\Xa7n\X9aQ\X0b&Q\X99\Xbe\Xf6\Xef\X9c\X7f\X9e\Xc6\Xeo\X81\Xf1\Xe5\Xcc\Xe2\Xf3[\Xb9F\Xd8\X0b\Xdf\Xe5\X90V\Xb2u\Xf8\Xb3\Xd2\Xb7,\Xf8\Xc7y\XfeZ\r=-\X1e"]], [b'ef99f71f7dbd18db191347BFE55D0CA9A574DB77BC8648275CE258461450E793528E0CC6D2DCF8F5', '8A28FA0A14739B29C49F98048B6F30C0D0E7CA60A2DD49D6E1091A291A6B6612', ('node7.consensuspki.org', 5507), b'8A28FA0A14739B29C49F98048B6F30C0D0E7CA60A2DD49D6E1091A291A6B6612', b'2\Xa0W\Xef\X114\Xc5\Xf8\X07\X89\Xeo\XefX\Xe7\Xb4\X0f\Xbby\X12\X1c\X9e#^\Xf3*\X8cF\X9b\N\XbFmI\Xf2p\r\Xc0\Ny\X14\Xa1\Xd5>\\YI\Xb5\Xbc\X9e\X0f\X16\Xa1\Xc0\Xe3\X93\mz\Xe3z[:#\Xca\X8cF\X94\Xdc\Xef\Xb3\X94\Xfd\Xeb\X17^o\Xbdu\Xa9\Xaa\Xd89T\X036R8KYvQ\Xfd\Xa0g\Xf1\X17\Xb8\Xd0\Xf9U\Xe7\X9a\Xe6S\Xc2\X98U31\X1f\Xa8a\X85\X003\XdeZ\Xef'\Xa2\Xf6\X9e\Xeo\Xbd0\Xfaf\Xc0:\Xb9\Xa4S\Xa6\Xd7\Xa1\X05\Xd4tj\Xfdp\Xf7E\Xef*\X8ea\Xdc\Xcc\Xa33)\Xfa%\X01\X8d\Xf61\Xfc\Xd9K2\Xed\X82\Xac'\Xe8P>\Xad0\XddG1\Xac\Xd3\X0b\X83\Xe6\Xd9\X15\Xf9\Xa5\X1c\Xef<5g\Xaay\Xa8\X85\Xfe\Xf58Ejt\Xr\X81\Xb9%\Xf2\X84xg\Xeb\Xa52d\Xfe4\Xeb\X0b\X1a\X82[\X00Np?e%\X8f\X91Ta!\Xc5\X0f\X11b\r\Xa4\Xc0\X98\Xa5\Xe7e\Xdfw#B\A\Xdfu\Xf0(e"]], [b'ef99f71f7dbd18db191347BFE55D0CA9A574DB77BC8648275CE258461450E793528E0CC6D2DCF8F5', '8A28FA0A14739B29C49F98048B6F30C0D0E7CA60A2DD49D6E1091A291A6B6612', ('node4.consensuspki.org', 5504), b'8A28FA0A14739B29C49F98048B6F30C0D0E7CA60A2DD49D6E1091A291A6B6612', b'\Xab\Xa89fg\X8a\Xd3\Xc5=. \Xa7\X80l\Xd1\X11\X8d\X86Vb\Xc8\X8d\Xa1\X81\XebA\Xf3\X99th\Xa9\X01\X9f\Xf5N\Xf7\Xcd\Xa3\Xd1\Xdb\X08)\Xe2\Xe3\X19\X0f\Xfc\X98\Xa6A\Xb5\X4\X86\Xbd\Xa7Y\X8f=\Xf9\X07\X95W\Xc0\X1e\Xb9J\Xac\X82\Xaeb\Xbb\XafW\X084vI\X82jC \Xfa7\Xf73U\Xb9\Xc1\Xa2b\Xac\X9a\X83\X11\X9c\X1c\X99Z!\X98H^\Xf9\X90Q\X8ae\X84\Xaas\X85\X98\Xc6\X0c\N\X18\Xae\Xb2\Xe6\X01\Xea\Xb6\Xa9~\Xe9%\X07\Xf5\Xff\Xbbf\Xbf\Xb5\X19\Xa1\Xc98 \Xeb\XfcM\X91\Xc8\Xdb\X971:\X1b\X8f^\Xcf\X1aU\Xe8,\Xd7k\Xcb\Xt!\Xdf\X8b1\Xbea\Xeb!\X8b\Xeex\Xb5\X92W\Xe0sqS\X0e\X190,.wqd#\X18\X03n\Xb2\X9a\X00\Xaf\Xfc5\X8e\X8e\X0e\Xc1\Xa2'\X14\Xa9X\Xf8\Xc9\X02\Xed\X18\Xdd\Xfb\Xc6\Xf7 \Xfd\Xd15IA;\D\Xae\Xda\Xe8\Xae\X0c\Xe9'\Xed]TK\Xd6\XbaY\Xfc\Xc9d8\X9e8\X86\Xb1\Xbe\X1f\X99\Xe8n\X96\Xc0"]], [b'ef99f71f7dbd18db191347BFE55D0CA9A574DB77BC8648275CE258461450E793528E0CC6D2DCF8F5', '77779999E2B22ECC2A39AD85A2535EA1D5D15103E48B96D08938149F444439590', ('node10.consensuspki.org', 5510), b'77779999E2B22ECC2A39AD85A2535EA1D5D15103E48B96D08938149F444439590', b'8'x18\Xa38~\Xbb\X89\X0f\XafZ\Xbf\X17\XdeD\Xdfh\30\Xc1\Xd3\X1a\Xe8\X80c\Xf7\XcePjI\Xa8~\X9a\X8a\X14c\Xf6\X1w\Xce& \Xd0g\Xb5\Xa8\X818\Xc7i\X17? \Xa0\Xecs\X1e\X82\X8d7\Xec\X96\X926x\Xb9\X1d\Xa5#~m\Xad\X99\K8\Xb4v\Xa8\X8b\XbF\Xc5Y\X1f13\Xc8I\Xf5\X8f\X83\X0bf\Xe3\Xca&\Xa6bo\X96\Xa4\X01~\Xc2~\Xe9z=0\Xb6\Xb2\Xdc\Xfa9\X08\Xea\Xd8\Xda\Xd1)%\XdfT\Xbe\X91\Xde\Xad\Xa0\X02\Xbb\Xc2\Xae\Xdd0^%701u\X00g0\Xafh\Xf3#Y\Xc34\Xc664\X00\Xc9\Xf5\Xf1\X1f\XwP\Xf3\Xc2\Xd07\Xf3\X18\Xe6]\Xa5\Xaf\Xa8\Xba\X11\Xb5\Xf6]\X81k\Xb7\Xb3\X16&\Xefg\X0c\X1a\Xa4\Xf6\Xf9\X1b\Xdd\Xdd\X88\Xda\X01\Xb8\Xa5\X0d\Xbb\X8C\Xda)3KG\X08~T\X96\Xf1J\X0eJ\Xc7\Xc7t\X7f\Xef0\Xeo\X0c\X1eb\Xa7\Xec\X10a~c7=\X918\XbcJY\Xabj'$\X8e\Xa6\X98\Xca\Xb3\X81\Xf2\X92"]])
There is no Consensus among the CAs
The PKQuery check is finilized. The time in milliseconds: 1544628998714
```

Appendix B: Data model overview for the CA Certificates



Appendix C: Data model overview for the Regular Certificates

