View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Repositório da Universidade Nova de Lisboa

Miguel Pires Egídio Reis

Bachelor of Computer Science and Engineering

Blockchain-Enabled DPKI Framework

Dissertation submitted in partial fulfillment
of the requirements for the degree of

Master of Science in
Computer Science and Informatics Engineering

Adviser: Henrique João Lopes Domingos,
Departamento de Informática,
FCT/UNL

Examination Committee

Chairperson: Miguel Carlos Pacheco Afonso Goulão
Raporteur: Nuno Miguel Carvalho dos Santos
Member: Henrique João Lopes Domingos

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

September, 2019

Blockchain-Enabled DPKI Framework

Copyright © Miguel Pires Egídio Reis, Faculty of Sciences and Technology, NOVA University Lisbon.
The Faculty of Sciences and Technology and the NOVA University Lisbon have the right, perpetual and without geographical boundaries, to file and publish this dissertation through printed copies reproduced on paper or on digital form, or by any other means known or that may be invented, and to disseminate through scientific repositories and admit its copying and distribution for non-commercial, educational or research purposes, as long as credit is given to the author and editor.

This document was created using the (pdf)LaTeX processor, based in the "novathesis" template[1], developed at the Dep. Informática of FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

To Mariana.

Acknowledgements

Firstly, I want to thank my thesis advisor Prof. Henrique João Domingos, for his support and advice. Working with him was truly an honor. I also want to thank every teacher I had during my journey in Faculty of Sciences and Technology of the NOVA University of Lisbon, I learned immensely from them. I will cherish the moments I lived during the last years for the rest of my life. I look forward to apply what I have learned in my professional career.

I thank my parents, sister, and close friends, for their support and encouragement through stressful times. Without them, these years would be incredibly harder. Finally, I want to thank some of my closest colleagues Rui Louro, Rafael Figueiredo, and Arthur Rocha, for our fun experiencies and projects.

Abstract

Public Key Infrastructures (PKIs), which rely on digital signature technology and establishment of trust and security association parameters between entities, allow entities to interoperate with authentication proofs, using standardized digital certificates (with X.509v3 as the current reference). Despite PKI technology being used by many applications for their security foundations (e.g. WEB/HTTPS/TLS, Cloud-Enabled Services, LANs/WLANs Security, VPNs, IP-Security), there are several concerns regarding their inherent design assumptions based on a centralized trust model.

To avoid some problems and drawbacks that emerged from the centralization assumptions, a Decentralized Public Key Infrastructure (DPKI), is an alternative approach. The main idea for DPKIs is the ability to establish trust relations between all parties, in a web-of-trust model, avoiding centralized authorities and related root-of-trust certificates.

As a possible solution for DPKI frameworks, the Blockchain technology, as an enabler solution, can help overcome some of the identified PKI problems and security drawbacks. Blockchain-enabled DPKIs can be designed to address a fully decentralized ledger for managed certificates, providing data-replication with strong consistency guarantees, and fairly distributed trust management properties founded on a P2P trust model. In this approach, typical PKI functions are supported cooperatively, with validity agreement based on consistency criteria, for issuing, verification and revocation of X509v3 certificates. It is also possible to address mechanisms to provide rapid reaction of principals in the verification of traceable, shared and immutable history logs of state-changes related to the life-cycle of certificates, with certificate validation rules established consistently by programmable Smart Contracts executed by peers.

In this dissertation we designed, implemented and evaluated a Blockchain-Enabled Decentralized Public Key Infrastructure (DPKI) framework, providing an implementation prototype solution that can be used and to support experimental research. The proposal is based on a framework instantiating a permissioned collaborative consortium model, using the service planes supported in an extended Blockchain platform leveraged by the Hyperledger Fabric (HLF) solution. In our proposed DPKI framework model, X509v3 certificates are issued and managed following security invariants, processing rules, managing trust assumptions and establishing consistency metrics, defined and

executed in a decentralized way by the Blockchain nodes, using Smart Contracts. Certificates are issued cooperatively and can be issued with group-oriented threshold-based Byzantine fault-tolerant (BFT) signatures, as group-oriented authentication proofs. The Smart Contracts dictate how Blockchain peers participate consistently in issuing, signing, attestation, validation and revocation processes. Any peer can validate certificates obtaining their consistent states consolidated in closed blocks in a Meckle tree structure maintained in the Blockchain. State-transition operations are managed with serializability guarantees, provided by Byzantine Fault Tolerant (BFT) consensus primitives.

Keywords: PKI, DPKI, Blockchains, Distributed Ledger, Consortium Blockchains, Hyperledger- Fabric (HLF) Platform , Byzantine Fault Tolerance (BFT), BFT Consensus, Group-Oriented Cooperative Multisignatures (COSigs), Threshold Digital Signatures.

Resumo

As soluções PKI (ou infraestruturas de chave pública) baseiam-se na utilização de assinaturas digitais e suportam o estabelecimento de chaves públicas ou demais atributos de confiança associados à identificação de entidades principais, para uso como provas de autenticação. Uma PKI suporta a emissão, gestão ou revogação de certificados normalizados para autenticação desses atributos, sendo a norma X.509v3 o padrão prevalente. Os certificados são emitidos em cadeias, a partir de entidades centrais instituídas como autoridades raiz da certificação (CAs), sendo aceites como entidades de confiança. As CAs são constituídas por empresas, sectores governamentais ou outras organizações, atuando como entidades terceiras, externas aos principais para os quais os certificados são emitidos, e independentes do contexto de interação entre esses principais. As funções de uma CA são similares às padronizadas em *frameworks* para soluções PKI, sendo o padrão PKIX uma referência relevante. Embora tais soluções sejam geridas transversalmente aos sistemas que utilizam os certificados, os seus serviços estão na base de confiança destes sistemas.

As funções de uma CA ou de soluções PKIs estão assim na base de confiança de protocolos de segurança de comunicações (ex: HTTPS, TLS, IPSec, VPNs), aplicações de segurança (ex: Web/HTTPS, S/MIME, DKIM), na segurança do sistema DNS (ex.: DNS-SEC) e nos mais diversos serviços de software para computação ou de armazenamento de dados em nuvem. A utilização de CAs e PKIs tem ainda repercussão no uso de dispositivos Bluetooth ou NFC, na utilização confiável de automóveis, equipamentos hospitalares, serviços de *smart-cities*, aplicações IoT, *smart-cards*, cartões bancários, cartões de cidadão ou passaportes, entre muitos outros casos e sistemas críticos.

Dada a sua relevância, existem hoje preocupações crescentes acerca dos modelos de concepção de PKIs e respectivos modelos de confiança centralizados, em que a raiz de confiança é detida por entidades centrais atuando com cariz autoritário ou oligárquico. Estas são susceptíveis de operação incorreta, podem constituir pontos centrais de falha ou ataque e operam na prática sem escrutínio das condições de operação por parte dos principais para os quais os certificados são emitidos. Perante estas preocupações, o redesenho de soluções PKI com base em modelos decentralizados pode apresentar vantagens interessantes.

Várias soluções para descentralização da base de confiança foram propostas e algumas tornaram-se historicamente relevantes e bem conhecidas, no contexto de aplicações específicas, como por exemplo o sistema PGP (*Pretty Good Privacy*) orientado para gestão de certificados para Email ou para troca de ficheiros autenticados, num modelo de confiança P2P do tipo *Web-of-Trust*. Mais recentemente, a ideia de descentralizar a confiança numa solução PKI para uso genérico está associada à noção de DPKI (*Decentralized* PKI). Numa DPKI pretende-se descentralizar as funções de uma PKI sem que prevaleçam entidades com autoridade central. As funções e operações de uma PKI podem ser realizadas de forma cooperativa, envolvendo várias entidades distribuídas.

A tecnologia Blockchain potencia a abordagem de soluções DPKI, por permitir endereçar modelos descentralizados de notarização de operações com controlo de integridade do estado dessas operações, bem como de replicação desse estado em todas as entidades prticipantes, com garantias de consistência e com base em interações P2P. Numa DPKI suportada em plataformas Blockchain pode perspectivar-se a emissão colaborativa de certificados X509v3 com base em regras de emissão, validação ou de revogação, executadas por todas as entidades com critérios de consistência. É ainda interessante suportar critérios de reação rápida na verificação de históricos partilhados e rastreáveis relativos ao estado dos certificados. As anteriores operações podem ser processadas por Smart Contracts, programáveis para serem executados de forma consistente e autónoma pelas entidades que cooperam na DPKI.

Esta dissertação objetiva a concepção, implementação e avaliação experimental de uma solução DPKI baseada numa plataforma Blockchain. A solução baseia-se numa *framework* que endereça um modelo de consórcio colaborativo, alavancada por planos de serviços suportados na plataforma Blockchain Hyperledger Fabric (HLF). Os certificados X509v3 são emitidos e geridos respeitando invariantes de segurança e confiabilidade, com regras de processamento definidas e executadas pelos nós da Blockchain expressas em Smart Contracts (ou Chaincodes). Os certificados são emitidos de forma cooperativa, sendo autenticados por assinaturas digitais orientadas a grupos ou assinaturas de limiar, com propriedades de tolerância a falhas ou ataques bizantinos. Os Smart Contracts expressam os critérios e condições de emissão, assinatura, atestação, validação e revogação, sendo executados autonomamente e consistentemente por todas as entidades na Blockchain. O histórico de estados dos certificados são guardados em blocos de operações com garantias de integridade e imutabilidade, agregados numa estrutura de árvore Meckle mantida e replicada na Blockchain. As operações de transição de estado são processadas com garantias de serialização e ordenação suportadas por protocolos de consenso tolerante a falhas bizantinas (ou consenso BFT).

Palavras-chave: PKI, DPKI, Blockchains, Notarização Distribuída, Blockchains de Consórcio, Plataforma Hyperledger Fabric (HLF), Tolerância a Falhas Bizantinas (BFT), Consenso com BFT, Assinaturas Cooperativas Orientadas a Grupo (COSigs), Assinaturas Digitais de Limiar (Threshold Signatures.

Contents

List of Figures xvii

List of Tables xix

Listings xxi

Acronyms xxiii

1 Introduction 1
- 1.1 Context and Motivation 1
- 1.2 Problem Statement 6
- 1.3 Objective and Contributions 7
- 1.4 Document Structure 8

2 Related Work 9
- 2.1 Background 9
- 2.2 Public Key Infrastructure 9
  - 2.2.1 PKI and PKIx 10
  - 2.2.2 PKI Implementations 12
- 2.3 Web-of-Trust Models 14
- 2.4 Blockchain 16
  - 2.4.1 Blockchain Characteristics and Foundations 17
  - 2.4.2 Blockchain Trends 20
  - 2.4.3 Blockchain Platforms 20
  - 2.4.4 Decentralized Ledgering with Resilient Group Signatures 23
- 2.5 Blockchain-Enabled PKI Approaches 24
  - 2.5.1 Background 24
  - 2.5.2 PB-PKI 25
  - 2.5.3 Blockchain-Based PKI Management Framework 26
  - 2.5.4 IKP 27
  - 2.5.5 CertCoin 28
  - 2.5.6 PomCor: Backing Rich Credentials with a Blockchain PKI 29
  - 2.5.7 Blockchain-Based Certificate and Revocation Transparency 29

2.5.8 SCPKI 30
2.5.9 Other Approaches 30
2.6 Discussion 31

3 System Model and Architecture **35**
3.1 Application Scenario 35
3.2 System Model 36
3.2.1 Entities 37
3.2.2 Interactions 39
3.2.3 Requirements 43
3.3 Reference Architecture 43
3.4 Software Architecture Components 45
3.4.1 Base Hyperledger Fabric 45
3.4.2 Extended Hyperledger Fabric 47
3.4.3 Blockchain-Enabled DPKI Proxy 53
3.5 Adversary Model Considerations 55
3.6 Summary 56

4 System Implementation **57**
4.1 Prototype Overview and Technologies 57
4.2 Prototype Implementation 58
4.2.1 Blockchain Network 58
4.2.2 DPKI Proxy 59
4.2.3 DPKI Chaincode 65
4.2.4 DPKI Signatures 67
4.2.5 Bootstrap Signatures 67
4.2.6 Internal Clients 68
4.2.7 X509v3 Certificates and CRL Extensions 69
4.3 Summary 69

5 Experimental Evaluation and Analysis **71**
5.1 Evaluation Environment 72
5.2 X509v3 Certificate Issuing for External Clients 72
5.2.1 Cryptographic operations and their impact on the system 73
5.2.2 Latency variation in the issuing process 74
5.2.3 Impact in the size of issued X509v3 certificates 75
5.2.4 Impact of the number of endorsers 75
5.2.5 Impact of the number of normal peers 76
5.2.6 Comparison with the Issuing Process in a Conventional PKI Solution 77
5.3 Revocation of certificates and CRL Issuing 78
5.4 OCSP Requests 79
5.5 Summary 79

6 Conclusion and Final Remarks **81**
6.1 Conclusion 81
6.2 Future Work 82

Bibliography **85**

I Blockchain-Enabled DPKI Chaincode **91**

List of Figures

2.1 Hierarchy of a PKI certificate chain (left) and Architectural model of a PKIx (right) 10
2.2 Process of a simple PKI using OpenSSL 12
2.3 Threshold signature construction 23
2.4 Example of a PKI Blockchain structure 25
2.5 Certificates in Blockchain-Based PKI Management Framework 26
2.6 IKP schema 28

3.1 Application Scenario 37
3.2 Standard Model Sequence Diagram 39
3.3 Self-Signed Certificate Model Sequence Diagram 41
3.4 CSR Model Sequence Diagram 42
3.5 Architectural View 44
3.6 Hyperledger Fabric Architecture 46
3.7 Hyperledger Fabric Transaction Flow 47
3.8 BFT Consensus service for Hyperledger Fabric 48
3.9 Certificate Validation and Signature 49
3.10 Blockchain Public-Key Certificate State (BCS) 50
3.11 Example of Certificate Trust Level 52
3.12 Blockchain-Enabled DPKI X509v3 certificate 55

4.1 DPKI Signature Flow 68
4.2 A certificate issued by the Blockchain-Enabled DPKI 69

5.1 Variation in the latency in issuing X509v3 extended certificates. 74
5.2 Number of Endorsers and size of issued certificates. 75
5.3 Latency in issuing X509v3 extended certificates with a variable number of Endorsers. 76
5.4 Latency in issuing X509v3 extended certificates with a variable number of Regular Peers. 76
5.5 Performance of Threshold Signatures and Multi-Signatures in the revocation of certificates. 78

5.6 Performance of OCSP processing when using Threshold Signatures and Multi-Signatures. 79

List of Tables

2.1 Comparison of Blockchain Platforms. 21

4.1 Proxy REST API Endpoints 60
4.2 Proxy HLF SDK Client Interface 64
4.3 Excerpt of Chaincode Functions 67

5.1 Testbench Environment 72
5.2 Analysis of different Cryptographic Algorithms and Key Sizes 73
5.3 Comparison with the issuing process in a conventional PKI 77

Listings

3.1 Example of an Extended Smart Contract 52
I.1 Annex: Excerpt of the PKI Chaincode Properties and Functions 91

Acronyms

API	Application Programming Interface.
ARP	Address Resolution Protocol.
BaaS	Blockchain-as-a-service.
BCS	Blockchain Public-Key Certificate State.
BFT	Byzantine Fault Tolerance.
CA	Certificate Authority.
CoSigs	Cooperative Signatures (or Group Oriented Cooperative Multi-signatures).
CPS	Certificate Practice Statement.
CRL	Certificate Revocation List.
CRR	Certificate Revocation Request.
CSR	Certificate Signing Request.
DDoS	Distributed Denial of Service.
DHT	Distributed Hash Table.
DNS	Domain Name System.
DoS	Denial of Service.
DPKI	Decentralized Public Key Infrastructure.
ESCC	Endorsement System Chaincode.
EVM	Ethereum Virtual Machine.
HLF	Hyperledger Fabric.
HSM	Hardware Security Module.
HTTP	Hypertext Transfer Protocol.
HTTPS	Hypertext Transfer Protocol Secure.

IETF	Internet Engineering Task Force.
IKP	Instant Karma PKI.
IoT	Internet-of-Things.
IP	Internet Protocol.
IPFS	InterPlanetary File System.
IPSec	IP Security.
JSON	JavaScript Object Notation.
LAN	Local Area Network.
MAN	Metropolitan Area Network.
MITB	Man-in-the-Browser.
MITM	Man-in-the-Middle.
NFC	Near Field Communication.
OCSP	Online Certificate Status Protocol.
OID	Object Identifier.
P2P	Peer-to-Peer.
PAN	Personal Area Network.
PB-PKI	Privacy-Aware Blockchain-based PKI.
PBFT	Practical Byzantine Fault Tolerance.
PEM	Privacy-Enhanced Mail - Format.
PGP	Pretty-Good-Privacy.
PKCS	Public Key Cryptography Standards.
PKI	Public Key Infrastructure.
PKIx	PKI based on the industry standard X.509 model.
PoET	Proof of Elapsed Time.
PoS	Proof of Stake.
PoW	Proof of Work.
RA	Registration Authority.
REST	Representational State Transfer.
RFC	Request for comments.
RTT	Round-Trip Time.

SCPKI Smart Contract-based PKI.
SDK Software Development Kit.
SSH Secure Shell.

TCP Transmission Control Protocol.
TLS Transport Layer Security.

UI User Interface.

VA Validation Authority.
VAN Vehicular Area Network (including VANETs Vehicular AdHoc Networks).
VPN Virtual Private Network.

WAN Wide Area Network.
WLAN Wireless Local Area Network.

X509v3 X509 Certificate with extension fields.
XSPP Extended Signing Policies Provider.

Chapter 1

Introduction

1.1 Context and Motivation

Public Key Infrastructures. The IETF RFC 4949 (Internet Security Glossary) [55] defines public-key infrastructure (PKI) as the set of hardware, software, entities, policies, and operational procedures needed to create, manage, store, distribute, and revoke digital certificates for the use of public-key cryptography, authentication and key-distribution protocols and for standardized digital signatures. PKIs enable the secure, convenient, trust and efficient acquisition of public keys related to principals, with <Subject IDs to Public Keys> mappings established by the certificate attributes and their policy enforcements. The IETF-PKI working group [52] has been the driving force behind setting up a formal (and generic) standardized model based on X.509 PKI frameworks for X509 certificates [29], suitable for deploying a certificate-based architecture on the Internet. The PKIX model [59] is generically behind the standardization of the entity model and functions in PKI solutions in different available platforms. It is also the base framework of the PKI technology used by Internet Certification Authorities (CAs) on the Internet. In the PKI model, the certification structure is based on the use of X509 certificates, issued and managed in certification chains used as hierarchies of certification, with an arbitrary number of certification levels. In such hierarchies, each level L corresponds to an entity certifying the certificate for an entity in Level L-1. In practice, these levels are usually levels of the certification of the same CA. On top of each certification chain, there is a root-of-trust certificate, corresponding to the top certification level of the CA, accepted as a centralized trust authority [59]. In the validation of those certification chains, Public Keys of CAs (or Root Public Keys), are pre-installed in local key stores, provided as pre-configured objects in software systems (e.g. devices, computers, operating systems, software solutions, as provided by their manufacturers).

Based on the assumption that a root-of-chain (or a root in a certification chain) is considered trustable, certification chains can be processed and validated, by verifying each certificate in the chain, according to the respective standard attributes. The process is in general conducted in a direct way (considering that each certificate is issued and signed by an entity with an upper-layer certificate public-key). However, is also possible to adopt a reverse way (when a public key of a previously validated and trusted certificate was used to certify upper-layer certificate). Cross-certification is also possible, mixing direct and reverse certification chains. By possible peering agreements between two different CAs in different hierarchies, it is possible to address a model based on an oligarchy of independent CAs. A set of CAs can constitute such close oligarchies, as a variant extension of the centralized model, allowing for better scale-out conditions, but always following a centralized authority model.

PKI Limitations and Drawbacks. Despite its various advantages, the PKI trust model assumptions has several drawbacks [16]. CAs, as centralized roots of trust, and consequently central points of failures, can be compromised to issue fake certificates that are still valid due to bad practices. There is also a lack of transparency concerning CA behaviors. Information about certificate revocation is also slowly spread and verification of the certificates revocation status is prone to errors.

PKIs are cornerstones in the trust computing bases of most devices, systems, applications, services or protocols. The certification and validation of X509 certificates (and included public-keys for asymmetric cryptographic methods) are in fact behind the security guarantees of all the most used security protocols in the Internet from Cloud-Based Web-Enabled Security Protocols and Web/HTTPS environments to other TLS-supported secure channels, DNS-Security, SSH-Services, IPSec or Secure VPNs, as well as for accessing Wireless Local Area Networks. Many of the problems are induced by the centralized trust model, as addressed in PKI frameworks, and the related issues have been known and extensively discussed for many years by many authors.

At the same time, current implementations have suffered from several critical issues. One issue is the reliance on the user to make an informed decision when there is a problem in verifying a certificate. Unfortunately, it is clear that most users do not understand what a certificate is and why this might be a problem. This issue is also applicable to many programmers, due to the complexity involved in extended attributes and policies that are today involved in an X509 certificate and how these attributes must be completely managed in the verification processes in the context of specific applications. Hence they choose to accept a certificate, or not, for reasons that have little to do with their security policies, which may result in the compromise of their systems.

Another critical problem is the assumption that all of the CAs represented in the "trust store" of end-users are equally trusted and managed, and apply correct and equivalent management policies. The reality shows that this is not true. Those policies are really unknown by the users. This was dramatically illustrated by compromises from

well-known CAs [23], that resulted in many fraudulent certificates issued with public-keys mapped in well-known organization names and private keys controlled by others. Other concern is that different software implementations use different trust stores and different validation policies, hence presenting different security views under different trust assumptions. In this way, there are differences in the verification assumptions from different software even if executed in the same computer.

Summarizing, the centralized PKI model and the role of CAs can have problems and limitations, generally because it relies on central trusted parties acting as a possible central failure or attack points, and so, paradoxically, as possible untrustable parties. Trust centralization is also a big problem in terms of transparency. It leaves the management of outsourced trust without scrutiny from the certificated entities for which the certificates are issued. This is a door open for attackers to conduct man-in-the-middle (MITM) attacks, intrusion attacks against the CAs, or misbehaviors and bad operation of CAs, without any awareness control from the users. Currently, there are thousands of claimed trusted CAs around the world that have been appearing as attractive targets for cyber-criminals. These thousands of CAs have the ability to create alternative identities for one same entity, with different issuing practices in generating certain attributes in certificates. This is a problem in the control and correct validation of such identities and certificates in multilevel certification chains, a well-known vulnerability with serious security incidents and repercussions. Even considering the standardization effort behind the X509v3 framework, the fact is that different CAs also issue certificates with different attribute values and meanings, as well as, with different attributes considered critical for the validation under the sole responsibility of users.

Attacks against certificates issued on centralized roots of trust have been also conducted with a mixed of MITM attacks (combining different techniques such as: ARP spoofing, IP spoofing, DNS spoofing, HTTPS spoofing) and man-in-the-browser (MITB) attacks, as well as, other intrusions in computers where the certificates are used, with numerous incidents that have already shown to increase the risk of MITM attacks and new intrusions, when we place too much trust in one or a reduced number of CAs. In practice, the registration process of identities for certification requests is also a problem requiring special attention of registration processes conducted by CAs, or Registration Authorities (RAs) working as delegates of CAs. Attackers can trick the CAs and RAs into thinking they are someone else, or they can go so far as to compromise the CA, and get it to issue a rogue certificate for a fake or for a stolen digital identity. These attacks occurred frequently in the past. This was the case of the DigiNotar incident that happened in 2011 [14] when fraudulent certificates from the Dutch CA company were issued as a result of an attack.

More recently, in an incident that happened in 2017, hackers took control of Brazilian banks DNS server and tricked a CA into issuing a valid certificate to them [24].

Many observers state today that the *out-of-date* PKI design poses high-security risks

and single point of failures, in breaking encrypted online communication, as well as, in breaking the authenticity of communicating entities. This is a risk for all the security protocols and systems used today on the Internet (in different levels of the TCP/IP security stack and services). In the research community, there is also a growing opinion arguing that centralized PKI systems are struggling to keep up with the evolving and future requirements in the digital landscape and the modern world of scalability challenges, requiring a better designed and decentralized approach to PKIs.

It is interesting to note that the Internet Engineering Task Force (IETF), responsible for Web-enabled PKI standardization itself, has created a draft memo describing current issues and concerns on PKIs [28], sharing also the above concerns.

Independently, groups of researchers, for example, around the "Rebooting the Web of Trust" movement [54]), also assessed current PKI weaknesses in their publications, agreeing that the current implementation of centralized and Web-accessible PKI solutions have serious problems that should not be ignored.

DPKIs – Decentralized Public Key Infrastructures.

Decentralized public key infrastructures (DPKIs) are a possible alternative approach and are regarded as a promising way to return the control of online identities, and related public keys, to the entities they belong to. By doing so, DPKI solutions are envisaged as a way to address many usability and security challenges that plague traditional PKIs [59].

The goal of DPKIs is to ensure that, unlike the conventional PKIX framework design model, no single third-party can compromise the integrity and security of any system as a whole. In a DPKI the trust management must be decentralized by design, through the use of technologies that make it possible for geographically and politically disparate entities to reach consensus on the state of identities in a shared database. The DPKI approach, as approached in [2] focuses primarily on a decentralized key-value datastores and operations not depending on single points of failures or attack targets. More recently, the possible use of Blockchains is advanced as an enabling technology for that purpose, despite that other technologies can also provide similar or superior security properties for consistently distributed trust assumptions and distributed and consistently maintained shared databases.

Blockchains. In 2008 Satoshi Nakamoto caught the attention of many people and organizations by publishing an article regarding a P2P cryptocurrency system called Bitcoin [42]. This system uses a distributed ledger technology in a consistently replicated database, known as Blockchain: an immutable and public chain of blocks aggregating transactions ordered and validated under a model known as Proof of Work (PoW). Today, Blockchains are in the core of more than 2073 cryptocurrency systems [10], but this technology is now explored in many other applications such as IoT applications, supply chains, health records, and other application domains (e.g. [36, 41, 51]). Also, other consensus algorithms started to be used for different types of Blockchains, mainly due to the

fact that PoW, as a safe eventual-consistency model, is computationally very expensive, being a source of bad performance metrics for latency conditions and transaction throughputs, and harmful in terms of energy efficiency. Consensus planes for Blockchains can be also addressed by other solutions, including Byzantine Fault Tolerance (BFT) protocols, looking carefully for scalability vs. performance tradeoffs [13]. Relevant examples today use Practical Byzantine Fault Tolerant Protocols (PBFT) in different variants [8, 9, 40] in the core of different consensus planes in different types of Blockchains [7, 22, 26, 53].

Blockchain-enabled DPKIs. Several Blockchain-enabled DPKI solutions have been approached in recent proposals, each with different purposes (e.g. [1, 4, 38, 66]). Differently, models inspired by Web-of-Trust [59, 65], particularly used for managing public-key certificates, already addressed in the past the context of distributed-systems [48, 58]. Considering Blockchain service planes for the implementation of decentralized ledgers and replicated consistent logs of immutable and ordered operations, this technology can potentially help to overcome some identified problems of conventional centralized PKI systems, avoiding their root-of-trust model assumption. Blockchain-enabled DPKIs can be designed with a decentralized ledgering model, data-replication consistency guarantees, and distributed trust management properties. Therefore, PKI functions for issuing and managing X509v3 certificates can be based on a web-of-trust model enabled by the Blockchain services.

The research motivation and the hypothesis in our dissertation are to address Blockchain technology as a possible enabling technology for DPKI solutions, helping in resolving the identified issues, limitations and security drawbacks in conventional PKI systems and their centralized design models.

The decentralized nature of Blockchain-enabled DPKI management frameworks seems to be interesting to tackle the problems of conventional PKIs, providing the typical required functions. At the same time, the idea is to use the potential of Blockchains to eliminate single points of failures, and reacting fast to misuses of the managed certificates in their lifecycles, as well as, revocation operations.

Summarizing, some points emerge as initial advantages:

- A Blockchain can make the process of certificates' issuing and revocation states stored in a transparent, reliable, and immutable shared ledger. This can prevent attackers from breaking in or inducing in integrity breaks, thus effectively avoiding the MITM attacks against PKI and CA targets or intrusions on single nodes;

- Trust management can be established consistently by consensus protocols. All peers participating in the DPKI solution have to follow the rules of the consensus protocol, so the state of certificates will be regarded consistently, with all the operations that can change the replicated state observed by each peer with total order guarantees;

- A DPKI can benefit from the use of different types of Blockchain service planes,

> that make it possible for geographically and politically disparate entities, or peers, using neutral data storage solutions and flexible support for arbitrary data types. This allows the management of certificates and their life-cycle states in different representation formats, as well as different forms of binding information between identifiers and related certificates, that can be globally accessed and readable.

Furthermore, some researchers argued that the logic of key management and functions can be implemented on Smart Contracts [1], provided as a key element that can be extended with the necessary expressiveness requirements to execute the processing conditions on peer-operations, in many current Blockchain platforms.

Nevertheless, the use of Blockchains to design a DPKI solution always imposes some attention, for example, in requiring a device of a peer to synchronize a full copy of all the consensus data or different types of synchronization modes. The scalability considerations for the DPKI requirements must be also carefully analyzed in terms of induced workloads, as well as the tradeoff on the anonymization of Blockchain peer-identifiers, particularly in permissionless platforms, and DPKI participants and identifiers binding to the managed certificates. The support for BFT consensus primitives beyond the Blockchain-enabled DPKI operations is also another issue that must be investigated. To address these issues, some initial design considerations must be addressed, related to the adversarial model conditions, the management of peer identifiers and namespaces used for certificates' subjects, the registration process of such identifiers, or the way smartcards can be used to support the typical PKI functions. All the above dimensions are related to the objectives of the present dissertation in the definition of the system model and architecture for the proposed Blockchain-enabled DPKI solution, as well as, to drive the choice of the base Blockchain platform and its service planes.

1.2 Problem Statement

There are proposals to enhance PKIs [33] by promoting PKI transparency and responsibility using public logs. Recent proposals require that more than one entity confirm a certificate or allow domains to express security policies for their certificates, and detect wrong behaviors through shared logs. Although these proposals are a step forward, they did not have much impact in practice, mainly due to the difficulty and costs of implementation, performance penalties and others. Due to its characteristics, the Blockchain technology can help build safer and stronger PKIs. An immutable public ledger where we may store certificates and other information offers transparency and can be managed cooperatively. Data can be quickly disseminated, which is suitable for revocation data and it may be possible to specify rules related to certificates or CAs with the use of Smart Contracts, by also automatizing reactions such as reports or rewards [38].

In the last years, several Blockchain-based solutions started to come to life [1, 4, 20, 34, 38, 64, 66], despite that, many of them are based on cryptocurrency-oriented

applications. Blockchains are also very different in their services planes, permission vs. permissionless models, under open vs. consortium-based P2P distributed environments.

For the context of the dissertation we defined our problem in the following way:
Can we design a Blockchain-enabled DPKI framework addressing its required functions, in a cooperative P2P environment with a distributed ledger and decentralized trust model? Can we provide the necessary services allowing different entities to cooperate, in order to overcome the identified problems of PKI design assumptions based on a root-of-trust model?

We defined the objectives of our dissertation by addressing the above questions.

1.3 Objective and Contributions

In this dissertation we propose, design, prototype and evaluate, a Blockchain-Enabled DPKI framework model, based on service planes leveraged by the Hyperledger Fabric (HLF) platform [26], extended by some required services present in [21]. In our proposed PKI framework, X509v3 certificates are issued and managed following security invariants, processing rules, and trust and consistency control, expressed and executed by Blockchain Peers using Smart Contracts. Certificates are issued cooperatively, using group-oriented digital multi-signatures (COSigs), as aggregated multi-signatures or, optionally, by BFT threshold-signatures [61]. The Smart Contracts dictate how Blockchain peers participate in issuing, signing, attestation, validation and revocation processes, in a consistent way. Any peer can validate certificates by verifying its consistent state, stabilized in searchable blocks in a Merkle tree structure, maintained in the Blockchain. State-transition operations are managed with serializability and total ordering guarantees, provided by BFT consensus primitives, in a consensus plane designed for the HLF Platform.

Summarizing, our contributions are:

- Design of the framework, in a consortium BFT-enabled Blockchain model, defining its entities, roles, components, protocols, and operations, to enable the required PKI services. With the purpose of cooperative issuing of certificates, with certificates authenticated by cooperative Multi-signatures or BFT Threshold Signatures, representing a cooperative CA. Furthermore, with the use of Smart Contracts to define the issuing, certification, validation and revocation rules and invariants.

- Implementation of the framework in a prototype developed[1] on the HLF Blockchain in a Cloud-Blockchain PKI as a Service solution.

- Experimental evaluation of the developed prototype to validate the solution, including the observation of performance metrics with different parameterizations and

[1]Central repository of the Blockchain-Enabled DPKI prototype: https://github.com/miguelreisa/thesis-prototype-general

comparative analysis of such indicators faced to functions supported in a conventional and centralized PKI solution. The evaluations cover the following observations:

- X509v3 certificate Issuing, including:
  * Evaluation of cryptographic operations
  * Latency of the certificate issuing process
  * Size of extended X509v3 certificates
  * Variability in the number of peers in the issuing process
  * Comparison with certificate issuing in a conventional PKI
- Scale impact in the number of participant nodes and their roles in the DPKI
- Evaluation of processes for certificate revocation and issuing of Certificate Revocation List (CRL)
- Evaluation of certificate validation requests supported by the provided Online Certificate Status Protocol (OCSP) implementation

1.4 Document Structure

The remaining chapters of this report are organized as follows: in Chapter 2 we survey and discuss related work references to address our objectives and expected contributions, presenting and summarizing the identified inspirations, differences, limitations or drawbacks, considering our specific goals; Chapter 3 addresses the system model, design principles and architectural foundations of our proposed Blockchain-Enabled DPKI solution; Chapter 4 is focused on describing the implementation of a prototype of our designed solution; Chapter 5 is dedicated to the experimental evaluation effort conducted to analyze and to validate the proposed solution; finally, Chapter 6 concludes the dissertation report, presenting the main conclusions and final remarks, as well as, proposing future research work directions.

Chapter 2

Related Work

2.1 Background

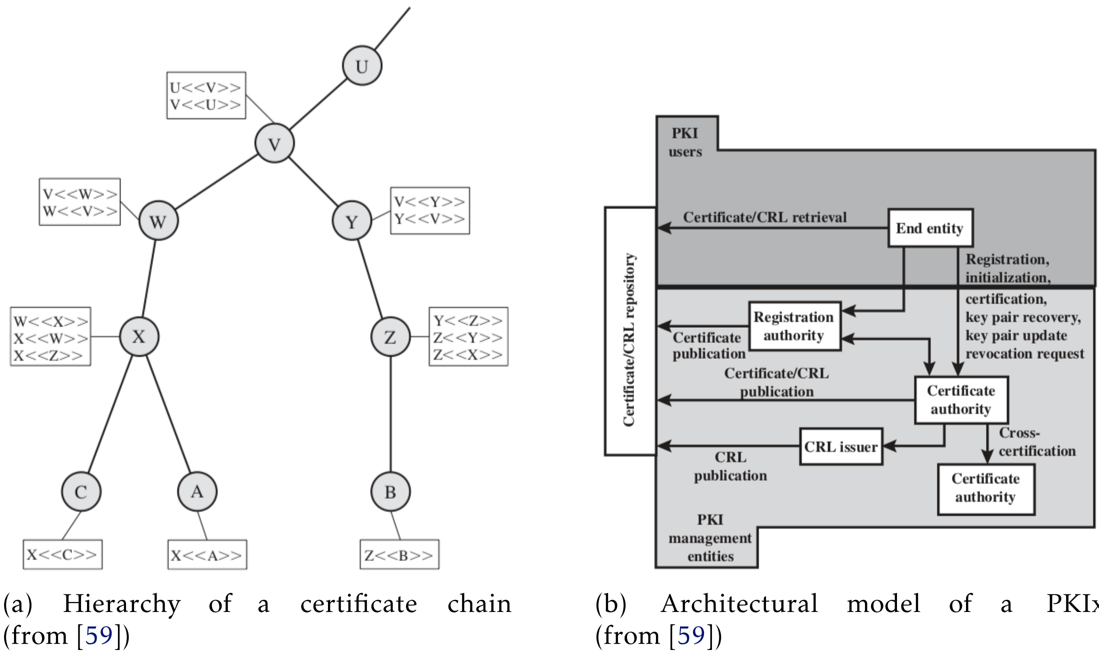
The background aspects for the objectives of the dissertation involve (i) the study of PKI (Public-Key Infrastructures) and the standardized frameworks behind the current implementations, (ii) the understanding of principles of trust decentralized models (namely Web-Of-Trust models), particularly addressed for the distributed management of public-keys and related X509 certificates, (iii) a study on different Blockchain platforms, to understand their typologies and relevant differences in the services' planes provide, and (iv) the study of recent proposals in the literature to address Blockchain-enabled PKI frameworks and solutions. We summarize those related work strands in the following sections, concluding with a critical analysis of related work issues focusing on the defined objectives and expected contributions.

2.2 Public Key Infrastructure

A Public-Key Infrastructure (PKI) is a set of software, hardware, policies, roles and procedures needed to create, distribute, store and revoke digital certificates based on asymmetric encryption. The main purpose of developing a PKI is to allow safe, convenient and efficient acquisition of public keys which allows us to encrypt and sign data [59].

The PKI has several tasks, such as:

- Define key pair creation policies.
- Define policies regarding the issuing and revocation of certificates of public keys.
- Define certification chains. Figure 2.1a shows an example of a chain of certificates. For example, W may be a CA that issued the certificate of another CA X, and X



(a) Hierarchy of a certificate chain (from [59])

(b) Architectural model of a PKIx (from [59])

Figure 2.1: Hierarchy of a PKI certificate chain (left) and Architectural model of a PKIx (right)

issued the certificate for the subject C.

- Generate key pairs if needed.
- Distribute, publicly, public key certificates, as well as the public key revocation certificates.

2.2.1 PKI and PKIx

PKIx is a PKI based on the industry-standard X.509 model that is appropriate for developing and deploying a certificate-based architecture on the Internet [59].

The main elements of PKIx are the following:

- **End entity**: a term used to describe the end-users, devices or other entity that can be identified on the subject field of a certificate. They normally use the PKI services.
- **Certification Authority (CA)**: issuer of certificates and Certificate Revocation Lists (CRLs). It can also support a wide variety of administrative functions, even though these functions are normally delegated to one or more Registration Authorities (RAs).
- **Registration Authority (RA)**: an additional component that may assume a variety of administrative functions from the CA. Associated to the final process of the end entities registration, but can also assist in other areas.
- **CRL Issuer**: an additional component that a CA may delegate to publish CRLs.

- **Repository**: any method used for storing certificates and CRLs, so the end entities are able to retrieve them.

Figure 2.1b shows the architectural model of a PKIx. Besides the main elements, PKIx identifies several management functions that must be supported by the management protocols, these functions are:

- **Registration**: the process in which a user makes himself known for the first time to the CA (directly or through an RA). Occurs before the CA issues a certificate to the user. This initiates the process of registration to a PKI. The end entity receives one or more secret keys to share for future authentication.
- **Initialization**: A client system must have the necessary key materials related to the keys stored elsewhere in the infrastructure before it can operate with security (e.g. the client needs to be initialized with the public key and other information regarding the trusted CAs to be used in certificate paths validation).
- **Certification**: Process in which a CA issues a certificate related to a user's public key. The certificate is returned to the user's client system and/or posts that certificate in a trusted repository.
- **Cross Certification**: A cross-certificate is a certificate issued by one CA to another CA, it contains a CA signature key used for issuing certificates. In order to establish this certificate, two CAs must exchange information.
- **Key Pair Recovery**: When using a key pair, it is important to provide a mechanism to recover the necessary decryption keys when access to the keying material is not possible. Without this mechanism, it will not be possible to recover the encrypted data. This allows entities to restore their key pair from an authorized key backup facility (normally it is the CA that issued the end entity's certificate).
- **Key Pair Update**: Key pairs need to be updated regularly and new certificates issued. This is required when the certificate lifetime expires or it is revoked.
- **Revocation Request**: A trusted person advises a CA of an abnormal situation that requires a certificate revocation. Reasons for this include private key compromised, affiliation change, name change, and others.

Various PKI enhancements were proposed in the last years in order to improve the weak aspects of the TLS PKI. One predominant concern is the dissemination of revoked certificates which is made by a CA. The dissemination should be fast to keep all the clients and servers connections secure and the revocation status needs to be authentic, that is, clients can verify if the revocation was created by a trusted CA [62]. Another concern is the fact that there is no effective way to monitor SSL/TLS certificates in real-time. Fake certificates can be issued by compromised CAs or simply due to mistakes.

Examples of improvements are [33]:

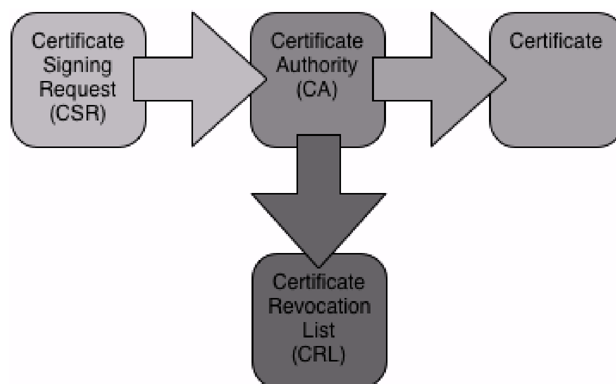


Figure 2.2: Process of a simple PKI using OpenSSL

- Approaches based on public logs that are constantly verified to monitor CAs [32].
- Recent propositions require that multiple entities must confirm the validity of a certificate.
- Revocation systems that spread information of non-expired certificates that were invalidated (e.g. private keys leaked).
- Allow domains to express security policies regarding their TLS certificates and detect incorrect behaviors by publishing policies in public logs.

Despite the appearance of new proposals, they didn't have a practical impact. CRLs are still the best-established approach to disseminate information of revocation, although it is not activated in mobile browsers and most desktop browsers.

2.2.2 PKI Implementations

Public Key Infrastructures (PKIs) are needed since most applications need to be secured with certificates and respective keys but they are difficult and expensive to implement since the implementation of a flexible trust center software is expensive and prone to mistakes.

OpenSSL [49] is Apache-licensed and it is the most basic form of CA and tool for PKI and is included in the main Linux distributions. It is a toolkit for the SSL/TLS protocols that allows creating a PKI with the basic components CA, RA, certificate, Certificate Signing Request (CSR), CRL and Certificate Practice Statement (CPS), where the last one is a document that describes the structure and processes of a CA [50]. The OpenSSL command line allows to easily create CAs, certificates, and others.

A process of a simple PKI is shown in Figure 2.2 (from [50]). The requestor generates a CSR and submits it to the CA, the CA then issues the certificate based on the CSR and returns it to the requestor. Later, if the certificate is revoked, the CA adds it to the CRL.

Root CAs issue certificates for CAs, intermediate CAs also issue certificates to CAs but are bellow the root, and signing CAs issue certificates for users.

OpenCA [46] is an open-source PKI implementation based on three main components: web interface to perform operations, OpenSSL backend for the cryptographic operations and a database that store the needed information, such as CSRs, certificates, Certificate Revocation Requests (CRRs), and CRLs.

OpenCA has many elements, such as a public interface, RA interface, CA interface, flexible certificate extensions, CRL issuing, warnings for soon to expire certificates and many others. It is designed for a distributed infrastructure in order to support maximum flexibility for big organizations.

EJBCA [15] is a PKI CA software built with the Java language. It is based on components and independent of the platform. Besides that, it guarantees flexibility and scalability and can be used to build a complete PKI infrastructure for any organization.

It can have multiple CAs and. an unlimited number of SubCAs and RootCAs. X.509 is one of the standards and supports certificates of long or short expiration. An RA Web UI is available for self-registry and issuing by the administrators. It also possesses revocations and CRLs.

Big scale installations can use multiple EJBCA instances running on a cluster, a distributed database in a separated cluster and a third cluster with HSMs (Hardware Security Module that offers additional security) that stores the keys of different CAs.

It supports multiple PKI architectures, such as:

- Single CA/RA: single instance that acts as CA and RA.
- CA with distributed RAs: having multiple RA instances allows to register a diverse set of users and devices. CAs have role-based access control in order to decide what each RA is allowed to do.
- External RA with only outgoing connections: an external RA (or several) stores the certificate requests and revocations in a separated database. The CA can, periodically, pull these requests from the database (only outcoming traffic is allowed from the CA). This is a way to isolate the CA for security reasons, allowing flexibility in registering entities at the same time.
- Validation Authority (VA): EJBCA has a VA incorporated, therefore the Single CA/RA setup has validation of certificates. However, if we want to build a bigger PKI it is better to separate the VA from the CA for performance reasons or, if we want to use a VA for multiple PKIs, for example.
- Cluster and High Availability: both CA and VA can be clustered in order to achieve more availability and performance. This means that there are more servers involved, but the architecture doesn't change whether it is clustered or non-clustered.

2.3 Web-of-Trust Models

Web-of-Trust [59, 65] is a concept introduced in the design of security services for distributed systems. This model, targeted for decentralized trust management functions, was considered in the past as a support component in the design of Pretty-Good-Privacy (PGP), GnuPG, and other OpenPGP-Email systems [48, 67] to establish authentication proofs of binding between public keys and principals, as defined in the IEEE RFC 822 standard [60], where unique identifiers are related to email addresses. The Web-of-Trust model was defined as an opposite approach for centralized trust models (e.g. PKIs). Based on this model, is also possible to address multiple and independent Webs of Trust, where any peer, through their identity certificate, can be a part of multiple webs, acting as a CA for their own certificates, managed in P2P interaction models.

For the decentralized management of public-keys and certificates, each peer maintains locally a ring of its private keys, maintained securely and possibly encrypted with passphrase-based encryption methods, and a ring of public keys that includes sets of self public keys and public keys of other peers. The main idea is not to include any specification for establishing certifying authorities or centralized trust. Instead, the model offers autonomous and convenient means of deciding trust, associating trust-information metrics with the public keys stored in the public-key ring structure.

The trust information on the public keys is included as additional data in each public-key entry of the public key ring. Trust data is managed and computed as a result of the dissemination of certificates in P2P interactions, where new, unknown or already known, certificates are received in announcements, originally sent and disseminated by different peers.

In a basic model, the structure for a public-key ring organization is as follows:

- Each entry is a public-key that corresponds to a certificate. Associated with each entry there is a key legitimacy field that indicates the extent to which the peer node managing the ring will trust that public key which belongs to another peer node; the higher the level of trust, the stronger is the binding of that peer with the stored public key.
- Also associated with a public key entry there are zero or more digital signatures (collection of signatures certifying the public key). These signatures are collected from messages arrived from other peers in the dissemination process of certificates.
- Each signature in the collection has an associated signature trust field that indicates the degree to which the principal trusts the signer to certify other public keys.
- The key legitimacy field for each public key certificate is derived from the collection of signature trust fields in each entry.
- Finally, each entry defines a public key associated with a particular owner, with an owner trust field included. This field indicates in each moment the degree to which

this public key is trusted to sign other public-key certificates.

In the management of the trust-related information of certificates stored in the public key ring, the levels of trust are assigned form initial parameterizations and computations, defined and executed in an autonomous way, by each peer (in isolation). In the model, signature trust fields are in practice cached copies of the owner trust field from another entry. Locally to each peer, the management process for public-key rings can be conducted by using the following rules:

- When the peer inserts a new public key on the managed public-key ring, it must assign a value to the trust associated with the owner of this public key. If the owner is the peer itself, and therefore this public key also appears in the private-key ring, then a value of ultimate trust can be automatically assigned to the trust field. Otherwise, it will be necessary to decide some trust value to be assigned to the owner of this key. For this decision, it can be specified that the owner is unknown or that a certain level of trust will be decided, according to possible local peer parameterizations for this purpose.
- When the new public key is entered, one or more signatures may be attached to it (depending on the received messages signing that public key). As new signatures can arrive in the future, more signatures may be added later to the public-key entry. When a signature is inserted into the entry it is necessary to search the public-key ring to find if the signature is among the known public-key owners. If so, the owner trust field value for this owner is assigned to the trust field for this signature. If not, an unknown user value is assigned (or some other default metrics).
- The value of the key legitimacy field is then computed considering the signature trust fields present in the entry. If at least one signature has a signature trust value of ultimate, then the key legitimacy value can be considered as complete. Otherwise, it is computed a weighted sum of the trust values. For example, a weight of 1/X can be given to signatures that are always trusted and 1/Y to signatures that are usually trusted. X and Y are configurable parameters in each peer node. When the total of weights of the introducers of a Key/PeerID combination reaches 1, the binding can be considered to be trustworthy, and the key legitimacy value is then set to complete. Thus, in the absence of ultimate trust, the peer must collect X signatures that are always trusted, Y signatures that are usually trusted, or some combination to achieve the complete trust.

The management process of public key rings continues as a dynamic process, being orthogonal to the use of public keys by different applications running in the peer node. Any peer may wish to revoke its current public keys. This can happen due to the compromise of the respective private keys or is simply to renew the validity of the public keys. Note that the compromise would require that an adversary, somehow, had obtained the

involved private keys or that the opponent had obtained the private key from the private key ring, as a consequence of an intrusion attack.

The convention for revoking a public key is for the owner to issue and to sign a key revocation certificate, sending it to other peers. Each peer can also reroute an observed revocation certificate to other peers. A revocation certificate can be similar to a regular certificate, only including a special attribute mentioning that the purpose of the certificate is to revoke the respective public key. The corresponding private key must be used to sign the revocation certificate that revokes the corresponding public key. The owner should then attempt to disseminate this certificate as widely and as quickly as possible, to enable potential correspondents to update their public-key rings as soon as possible. Note that an adversary who has compromised the private key of an owner can also issue a revocation certificate. However, this would deny the adversary, as well as the legitimate owner in the future use of the public key. Therefore, it seems a much less likely threat than the malicious use of a stolen private key.

There are two main considerations we must mention about the presented Web-of-Trust model. The first consideration is that the model follows an asynchronous process of convergence, with no prior consistency guarantees on the state of the distributed public-key rings that are stored and managed differently by the different peers. Furthermore, the collection of signatures for each disseminated public key corresponds to different aggregations of individual signatures and is not signed cooperatively and consistently by the same group of peers. The last observation is relevant for the approach of the solution as we must address in the dissertation.

2.4 Blockchain

The Blockchain is a paradigm that can reinvent the way we deal with distributed databases, contracts, transactions for shared logging, or notarization systems under properly distributed system model assumptions. It can enable the creation of transparent global networks, allowing for direct relations between agents without the intervention of central authorities.

In the more essential engineering vision, a Blockchain is a distributed database, used as a decentralized and immutable ledger, with the purpose of managing distributed transactions and to store the state of such transactions in a replicated and consistent environment. The participating nodes execute consensus mechanisms and protocols in order to validate transactions and to store consistent states in the public shared ledger.

The Blockchain design principles introduce an innovative way to establish trusted relationships in P2P interaction models, avoiding the need for intermediation entities to act as central trust authorities. Centralized intermediation, besides being a slowness factor, is vulnerable to failures due to the fact that if it fails or operates incorrectly, all the trustability assumptions of dependent principals will be compromised, and they will be strongly affected. With Blockchain-enabled services, it is no longer necessary to

place trust in intermediary entities. Trust assumptions are based on verifiable consensus mechanisms, running in the services planes and structural internet working support for P2P interactions [37]. Apart from many incorrect uses, Blockchains are used today in business areas that can take advantage of the technology, including Finance, Energy, Mobility, Transports, Smart Cities, Smart Cars, IoT systems, with lots of industrial projects on-going, and a few interesting successfully cases showing innovation and competitive factors for new business-models or organizational-processes [13].

2.4.1 Blockchain Characteristics and Foundations

Backgound. A Blockchain system is supported by different planes of services, operating together to provide the required properties. In such planes there are in general three main essential components: (i) the immutable public ledger containing logged-transactions, (ii) the consensus mechanisms and related algorithms to validate and order transactions, (iii) and well-established processing rules to validate and to maintain the Blockchain state in the expected way, with the required consistency criteria [37].

Optionally, depending on the nature, the characteristics or application purposes of different Blockchains may be designed with different flavors, and other complementary services planes can be added: authorization (or permission) service planes (for permissioned-Blockchains), membership control planes to regulate the participation and roles of peers (in consortium Blockchains), alternative or configurable consensus planes, as well as security and privacy services planes or trusted execution environment components (for security requirements).

A relevant component in a Blockchain is the mechanism targeted to address incentive models, a relevant feature related to the sustainability conditions of the Blockchain in order to incentivize the participation and engagement of nodes. This is particularly relevant to permissionless Blockchains. Incentive models can have strong repercussions on the support for the verification of transactions, as well as on the engagement of nodes in the consensus plane mechanisms. For example, the mining process in Bitcoin addresses exactly the above dimensions, allowing participants to obtain rewards from their proof of engagement, using Proof of Work (PoW) as a consensus-style mechanism [42].

Currently, another important element of Blockchain platforms is the establishment of processing rules and algorithms using conventionally named Smart Contracts [57]. In a more pragmatic vision, these contracts are essentially computer programs that can be expressed in a programming language, expressing invariants, conditions and other processing rules. Smart Contracts are stored and executed by the Blockchain nodes and can be used to automate the unstoppable transfer of crypto-tokens between users, according to agreed-upon conditions. In other application-domains, Smart Contracts are also used to deliver things, such as weather data, currency exchange rates, airline flight information, and sports statistics.

Blockchain ledger and consistency. The Blockchain ledger, as a chain of blocks, is a

data-structure (such as a Merkle-Tree) containing blocks of transactions, with each block having proof of integrity with a cryptographic hash of the previous block. In Blockchains that use PoW consensus mechanisms, the blocks contain a nonce.

The chain establishes the total order of transactions and, therefore, avoids double-spending attacks, since the adversary can't change the block data without the other nodes noticing due to the hashes present in the chain, and because there is a mechanism to select the right chain (based on the number of blocks computed). This means that it is only possible to append or read data from the ledger, after the validation of the blocks.

Types of Blockchain platforms There are three main types of Blockchains, regarding membership management and permission model: permissionless, permissioned and consortium [45]:

- **Permissionless/Public:** the ledgers are stored in P2P networks with decentralization and anonymity. PoW is the most used consensus protocol and it has the objective to limit the rate which new blocks are committed. Anyone can join the network but consensus needs a big amount of energy and time. Bitcoin and Ethereum use this type of membership.

- **Permissioned/Private:** also known as distributed ledger technology. Normally, a single organization has control of the Blockchain in order to manage the participation membership. Parties have verifiable IDs and need access control to participate. This allows the secure interactions between identity groups that possess a common objective but don't fully trust each other. It is possible that in such cases, different roles for participants can be defined. It may exist a closed group of nodes with the task of creating new blocks by executing BFT algorithms to determine the order of the blocks stored in the ledger. Not all roles will be involved in such operations, given the scalability and performance problems that can be related to such algorithms. Unlike permissionless architectures, permission Blockchains do not spend a big amount of energy and time in the consensus algorithm. Multichain [22] for example uses this type of membership control.

- **Consortium Blockchains:** semi-decentralized and permissioned Blockchains, but instead of a single organization having the control, a number of organizations have the control of the Blockchain where, for example, each one may control a node of the network. The administrators restrict reading rights to certain users as they wish or can assign specific roles to other users. Data can be public or private. In concrete implementations, sometimes we verify that not all functions are based on a full P2P model due to the presence of different roles. For example, Hyperledger Fabric (HLF) [26] uses this type of membership management control.

Blockchain consensus plans and consensus proofs. Depending on the design assumptions for consensus guarantees and possibly related inventive model, different types

of consensus mechanisms can be involved in consensus planes. The most well-known are the following [11]:

- **Proof of Work (PoW)**: as introduced above, is particularly used in permissionless Blockchains and for cryptocurrency ecosystems. This algorithm consists of the computation of a strong hash-function, having a random nonce and the hash of the block as input, to obtain a previously defined result. Peers repeat this computation process until they achieve the result. When a peer solves this cryptographic puzzle, he sends the block in question to the rest of the network. Peers receiving the block verify its contents and the solution of the puzzle, appending the block to the ledger if the block and solution are valid. The peer that solved the puzzle receives a monetary reward for the energy and effort spent. The downsides of PoW approaches are the energy consumption and the fact that decentralization may be lost if a miner has more computation power than the rest of the network, resulting in the renaissance of oligarchies in practice (a situation that is happening in the current crypto-currency systems).

- **Proof of Elapsed Time (PoET)**: an approach that tries to remove the computational cost of PoW maintaining at the same time its advantages. For each block, miners must wait an agreed set time, and the first to finish waiting is selected to validate the block. A platform is used to ensure that the miners, in fact, waited the defined time (under certain trustability assumptions). The downside is that the platform seller must be trusted and proofs of elapsed time must be obtained and exhibited with required trusted verification.

- **Proof of Stake (PoS)**: in this case, the creator and validator of a new block is selected in a deterministic way, depending on its wealth/stake. The more wealth a miner has, the more mining power he has. Miners are incentivized simply due to the fact that if they do not have correct behavior, they lose their wealth. The drawback is that wealthy miners acquire more easily more cryptocurrencies, where new miners have difficulties to do so, causing the possibility of centralized wealthy peers with more power.

- **Consensus Proofs with Practical Byzantine Fault Tolerance (PBFT)**: interesting for consortium Blockchains, is one of the consensus algorithms used in HLF that supports pluggable consensus components. A leader does the ordering of the transactions and consensus is made through these steps: leader broadcasts a request, each validator signs and broadcasts a prepare message, and if enough prepare messages are received, the validators broadcast committed messages. The request is accepted if enough commits are received. This algorithm has a considerably better performance when compared to consensus algorithms used in permissionless Blockchains, but it does not scale well when the number of participating nodes in the consensus

rises. The optimization of PBFT consensus in scalable Blockchains has been an interesting research area, addressing scalability vs. performance tradeoffs.

2.4.2 Blockchain Trends

Cryptocurrencies are among the many applications that can be implemented using the Blockchain. Its distributed and decentralized characteristics make it possible for the creation of many different types of applications.

In the last year, we saw the use of the Blockchain technology by companies to explore new ways to approach business processes, IoT, content streaming [35], security (mentioned in this paper) and many others.

In the future, we will probably see an increase in the use of this technology in the use of Smart Contracts in traditional business systems, security innovations, Blockchain-as-a-service (BaaS) [3], and others. The food industry can suffer major changes regarding transparency and traceability in its flow. Even the art industry can take advantage of a Blockchain ledger to fight thieves and art forgers.

2.4.3 Blockchain Platforms

There are many platforms, each with its focus points and characteristics. Table 2.1, based on a table from [21], compares several platforms regarding their main characteristics. The platforms are briefly explained:

- **Ethereum [17]** is used by many applications, some of them approach PKIs and are mentioned in this report. It is a decentralized platform that runs Smart Contracts in a virtual machine (Ethereum Virtual Machine - EVM) that requires payments via the cryptocurrency 'gas'. It is designed so it can be used for any type of application. The consensus mechanism is PoW but it is expected that, in the future, a PoS based algorithm named Casper [18] will be used.
- **Quorum [53]** is an enterprise and permissioned version of Ethereum. The consensus mechanism is based on a voting process that uses Smart Contracts to assign voting rights, offering better performance when compared to Bitcoin and Ethereum. There are open and private transactions: open transactions are similar to Ethereum's transactions and in private transactions data is not publicly shown.
- **Hydrachain [25]** is a permissioned extension of the Ethereum platform. It allows the setup of private or consortium chains. Ethereum's Smart Contracts are fully compatible with this extension. Instead of using PoW, like Ethereum, it uses a BFT consensus protocol named HC Protocol, where a quorum of validators signs the blocks. One of the upcoming features is the possibility of multi-chain setups.
- **Hyperledger Fabric [26]** is a permissioned Blockchain that allows the insertion and remotion of components regarding consensus and membership services. It

						Consensus plane			Storage plane	View plane			
	Type	**Comm.**	**Decentr.**	**Mult. chains**	**Smart Contracts**	**Mechanism**	**BFT**	**Throughput scalability**	**Ledger replication**	**View computation**	**Tx. privacy**	**Peer anonym.**	**Sign. Validation[7]**
Ethereum	Permissionless	P2P	Full	✗	✓	PoW	51%	Low (< 100 tps)	Global	Eventually consistent, causally ordered SMR	✗	✗	Traditional
Quorum	Permissioned	Hybrid	Partial	✗	✓	Consortium (Pluggable)	3f+1[1]	Presumably high (> 1000 tps)	Partial (private Smart Contracts are only replicated between authorized peers)	Strongly consistent, totally ordered SMR	✓	✓	Traditional
Hydrachain	Permissioned	Hybrid	Partial	✗	✓	Consortium	3f+1	Presumably high (> 1000 tps)	Global	Strongly consistent, totally ordered SMR	✗	✗	Traditional
Hyperledger Fabric V1	Permissioned	Hybrid	Partial	✓	✓	Off-chain service[2] (Pluggable)	3f+1[2]	High (> 1000 tps)	Partial (each *channel* holds a ledger between a subset of nodes)	Strongly consistent, totally ordered SMR	✓	✗	Multi-signatures
Hyperledger Fabric with BFT and Group Signatures[6]	Permissioned	Hybrid	Partial	✓	✓	Off-chain service[2] (Pluggable)	3f+1[2]	High (> 1000 tps)	Partial (each *channel* holds a ledger between a subset of nodes)	Strongly consistent, totally ordered SMR	✓	✗	Multi and Threshold Signatures
Hyperledger Sawtooth	Permissioned	P2P	Full[3]	✗	✓	PoET / random leader election[2]	51%[4]	Presumably high (> 1000 tps)	Global	Eventually consistent, causally ordered SMR	✗	✗	Traditional
Hyperledger Burrow	Permissioned	Hybrid	Partial	✓-	✓	PoS (Tendermint)	3f+1	High (> 1000 tps)	Partial (each chain holds a ledger between a subset of nodes)	Strongly consistent, totally ordered SMR	✓-	✗	Traditional
Hyperledger Iroha	Permissioned	Hybrid	Partial	✗	✗	Consortium	3f+1	Presumably high (> 1000 tps)	Global	Strongly consistent, totally ordered SMR	✗	✗	Multi-signatures
Counterparty	Permissionless	P2P	Full	✗	✓	PoW	51%	Low (< 100 tps)	Global	Eventually consistent, causally ordered SMR	✗	✗	Multi-signatures
Multichain	Permissioned	Hybrid	Partial	✓	✓	Consortium	Not BFT	High (> 1000 tps)	Partial (each chain holds a ledger between a subset of nodes)	Eventually consistent, totally ordered SMR	✓	✓	Multi-signatures
Openchain	Permissioned	C/S	Partial	✓	✓	Partitioned Consensus[5] (Pluggable)	n/2 + 1	High	Global	Strongly consistent, totally ordered SMR	✗	✓	Multi-signatures
Tezos	Permissionless	P2P	Full	✗	✓	PoS	51%	Presumably Low (< 80 tps)	Global	Eventually consistent, causally ordered SMR	✗	✗	Multi-signatures

[1] With Istanbul BFT; [2] With the unofficial BFT-SMaRt ordering service presented in [7]; [3] Assuming every node is equipped with Intel's SGX technology. Otherwise decentralization is partial; [4] If using PoET; [5] Transactions are validated by different authorities depending on the assets being exchanged. Each organization controls their own instance and each instance has only one authority validating transactions; [6] Version implemented in [21]; [7] All Blockchains support the normal traditional signature validation.

PoW = Proof-of-work
PoET = Proof-of-elapsed-time
PoS = Proof-of-stake

Table 2.1: Comparison of Blockchain Platforms.

does not require that Smart Contracts are written in a specific language or based in cryptocurrencies. Uses a notion of membership that may be integrated with industry-standard identity management. Nodes can participate in multiple chains (named channels) concurrently. This platform separates the transaction flow in three steps that may be performed in different entities in the system:

1. Execute the transaction and verify its correctness, endorsing it;
2. Ordering through a consensus protocol;
3. Validation of the transaction through application-specific trust assumptions that also prevent race conditions caused by competition.

- **Hyperledger Sawtooth** [44] is a permissioned Blockchain. The consensus mechanism is pluggable and its default is PoET, mentioned above in this report. Ethereum Smart Contracts are supported by the Hyperledger Burrow EVM and can be written in a wide variety of languages, such as Python, JavaScript, and Java. When possible, the transactions are executed in parallel in order to achieve better performance.
- **Hyperledger Burrow** [31] is a permissioned Ethereum smart-contract Blockchain. The consensus mechanism is the PoS Tendermint [8] engine. Entities have permissions and either contain Smart Contract code or correspond to a public-private key pair. Unlike Fabric, nodes cannot participate in multiple chains concurrently.
- **Hyperledger Iroha** [27] is a permissioned Blockchain-based on HLF. It can be used to manage custom digital assets, such as currencies, serial numbers, and patents. It uses a PBFT-based consensus algorithm, called Sumeragi, which has a peer reputation protocol in order to choose the order of nodes that process transactions. Sumeragi ensures high performance and low latency transactions.
- **Counterparty** [12] is a permissionless Blockchain build on top of Bitcoin. Users can trade any type of digital token and anyone may write Smart Contracts. It has an additional currency, XCP, in order to be possible to pay for the execution of Smart Contracts. There are wallets that, for example, allow the creation and management of tokens and multi-signature addresses. Since it is based on Bitcoin, the consensus mechanism is PoW, which has the drawbacks mentioned in this report.
- **Openchain** [47] is a permissioned Blockchain suitable for organizations that wish to manage digital assets in a secure way. The consensus mechanism uses Partitioned Consensus, where each Openchain instance only has one authority that validates transactions. Different transactions are validated by different authorities depending on the assets involved. There are validators, that validate and store transactions, and observers, that receive a read-only copy from the ledger and perform their own validations. Unlike most Blockchains, it has a client-server architecture. Each chain is governed by the organization that deployed that chain.

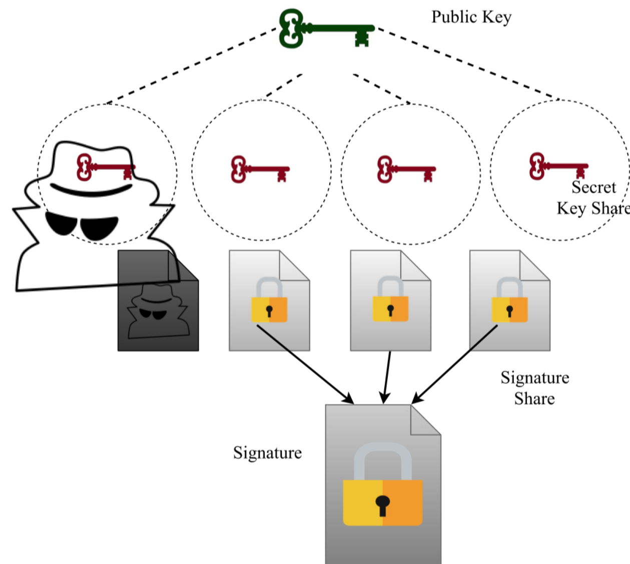


Figure 2.3: Threshold signature construction

- **Multichain** [22] is a permissioned Blockchain that supports multiple chains running in parallel. The consensus mechanism is a variant of PBFT, where instead of having several validators, there is only one that is selected in a round-robin type policy. Transactions occurring in a chain can trigger transactions in others. It does not support Smart Contracts.
- **Tezos** [63] is a permissionless Blockchain that supports Smart Contracts. The consensus mechanism is based on PoS and any stakeholder can participate in the consensus process. Smart Contracts are written in OCaml language and possess formal mathematical verifications in order to improve security and decrease bugs. The ledger is self-amending and stakeholders vote on proposals to modify the rules of the network.

2.4.4 Decentralized Ledgering with Resilient Group Signatures

Threshold signatures improve the resilience and robustness of a system, such as a Blockchain. In threshold signatures, a group of participants generates one fault-tolerant signature. It was invented over 20 years ago but it is still not commonly used.

The work in [61] implements threshold signatures schemes for an asynchronous Blockchain system. There is a group of n parties P = {P1, P2, ..., Pn} in which t may be corrupt, and for the reconstruction of a signature, it is necessary at least k valid signature shares and n, k and t must follow a specific requirement. This means it is not necessary to have all parties involved in the construction of the signature and a number of participants may fail to deliver their signature share and still being possible to reconstruct the signature. After the request to sign a message, each signing party computes its share of the signature and the client constructs the final signature by combining all the

parts. A simplification of this process is shown in Figure 2.3 (from [61]). The resultant signature is the same regardless of the combination of valid signature shares. To generate the secret shares (shares of the key), it is used the Shamir Secret Sharing [56], where a trusted dealer that does not belong to the group of n parties generates the secret shares and sends them secretly to each party member. The identity of all signers that collaborate in the construction of a signature must not be known and this is possible and desirable in a Blockchain. This scheme offers improvements in signature verification by using less storage and verifications when compared with the multi-signature scheme since, instead of multiple signatures being stored and verified, the validator only validates one signature.

2.5 Blockchain-Enabled PKI Approaches

2.5.1 Background

Conventional PKIs have several problems: CAs are central points of failure and there is too much trust placed on them; there is a lack of transparency regarding certificates and their revocation; and many others. Information about the revocation of certificates takes some time to disperse, and a client that is receiving a server's certificate may see the certificate as valid when in reality it was revoked due to some security concerns.

It is hard to update multiple entities and not have performance penalties due to communication and implementation costs. For example, backing up signature keys in a CA opens the possibility to the CA to act as the owner of the keys. This would require to backup the decryption keys on the CA but not the signature keys. In this case, each user would have to have two pairs of keys, one for encryption and other for signatures.

The Blockchain has characteristics that can improve PKIs, such as:

- Decentralization eliminates single-points-of-failure in CAs;
- Ledger, a reliable transaction record of PKI events, is published and it is trustable as long as the majority of the Blockchain contributors are honest;
- Certificate transparency of certificates and their revocation since the ledger can be a public log;
- Transactions in the Blockchain make the dissemination of revocation information faster than usual.

Figure 2.4 (from [4]) shows what a PKI Blockchain structure can look like. Such as the structure of a traditional Blockchain, each block has the hash of previous blocks in order to have a reliable transaction record. The registration, updates, and revocations of keys are made by publishing the identity, public key, and action (e.g. register) as a transaction. The Merkle root contains the hash of transactions per each block and is used to verify transactions without downloading the entire Blockchain.

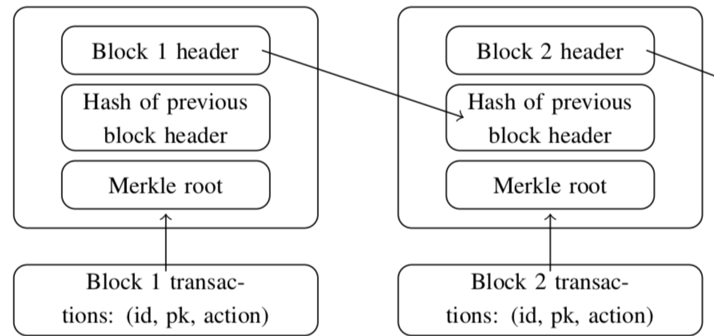


Figure 2.4: Example of a PKI Blockchain structure

2.5.2 PB-PKI

Despite existing Blockchain-based PKIs that offer security properties, they link the entities to their respective public keys. This may not be a problem in many cases, but some applications require some level of privacy (e.g. smart cards, IoT devices, smart vehicles).

In the Privacy-Aware Blockchain-Based PKI (PB-PKI) [4], L. Anox and M. Goldsmith adapt the CertCoin, a Blockchain-based PKI that is also mentioned in this report, to be privacy-aware. CertCoin links public keys to their identity in the registry and update of keys. Privacy-aware means that the public key is not connected to its identity except if the owner wishes to do so or there is a consensus from the majority of the network.

They established different notions of privacy:

- Total privacy: entities cannot link a public key to its identity and messages can be sent to non-specific members or via broadcast, without having knowledge of the public key identity.
- Neighbour group anonymity: an identity is identified within a group of neighbours. The entity of a group gives the other members information that allows them to link its public key to its identity when making a key update. This way the group of neighbours can attest to the rest of the network the correctness of the update.
- User-controlled disclosure: the users choose when to show the link between their public key and their identity.

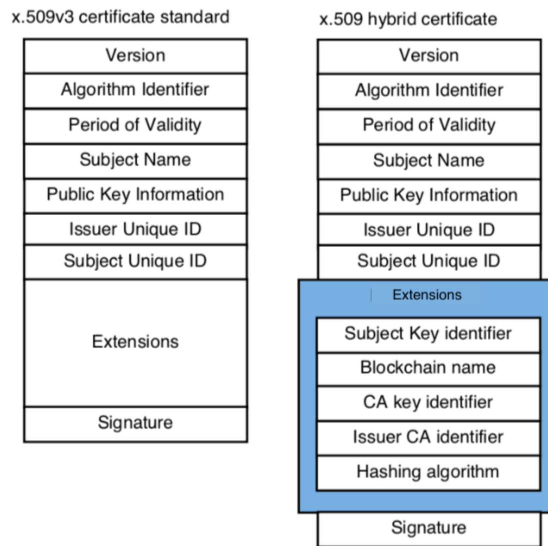
Each entity has a public key-pair and an offline key-pair. The offline key-pair is randomly generated at each update and the new public key is calculated by using the previous public key and offline secret key as input. By doing this, it is not possible to link the new key to the previous one. An entity can use its offline key pair to prove the ownership of the public key and has also a master offline key pair that can be used to recover the offline keys in case they are stolen or lost.

In order to prevent the link a new key to the old one by looking at a key update and seeing that a key changed, there are two possible approaches: the use of a time delay where both keys are valid so it is hard to know when the new key was created; or, when

using neighbour groups, all the members of the group must update their keys whenever one member wishes to do so.

When an identity must be tracked, the authorities can ask the identity to show the link between the public key and identity. If it does not follow this command, it is possible for the majority of the network to use key shares from previous updates and reconstruct the keys that belong to the target entity. This reconstruction can be achieved since the entity that updates its keys must share the key secrets of the new key with the rest of the network, in order to make this consensus possible.

2.5.3 Blockchain-Based PKI Management Framework



(a) Certificate extensions related to the Blockchain

Cert.	Issued By	Issued To	CA Contract ID	Issuer CA ID
RootCA	RootCA	RootCA	0x1234xxxx	0x00000000
SubCA	RootCA	SubCA	0x5631xxxx	0x1234xxxx
EndUser	SubCA	End user	-	0x5631xxxx

(b) Type of certificates stored in the Blockchain

Figure 2.5: Certificates in Blockchain-Based PKI Management Framework

This framework [66] focuses on the revocation of certificates, one of the main problems of traditional PKIs. As many others Blockchain-based PKIs, the Blockchain acts as a public append-only log that offers certificate transparency since the certificates and associated public keys are stored in that log. It allows revocation of certificates, and since it is not possible to remove information from the Blockchain ledger, a CA can only mark a certificate as revoked. Any behavior that is not correct from a CA can be traced by other entities. The Blockchain contains a white-list that has valid certificates and a black-list that contains revoked certificates. When revoking a certificate, an entity simply adds the hash of the target certificate to the black-list, marking it as revoked.

This work extends the X.509 certificate to be compatible with the Blockchain approach by adding extension fields to set metadata information related to the Blockchain. The extensions are shown in Figure 2.5a (from [66]).

Each CA has an associated Ethereum Smart Contract that possesses an array of hashes of the issued certificates, additional info (i.e. expiration date) and a mapping of the revocation data referenced by the certification hash. A certificate contains the address of the Smart Contract belonging to the parent CA to allow the validation of the chain of trust between a leaf and the root certificate. The types of certificates are shown in Figure 2.5b (from [66]).

Due to the fast synchronization of the ledger in each node, revocation data is quickly disseminated through the network, something that does not occur in traditional PKIs.

Since this system makes use of the Ethereum Blockchain, there are gas costs involved, and due to the volatility of cryptocurrencies, there is no certainty in the costs to upload and update certificates.

2.5.4 IKP

In traditional PKIs, it usually takes some time to notice and verify that a certificate is invalid or revoked. This is mainly due to the fact that there are no automatized processes to report certificates. Besides, there are no incentives to report non-authorized certificates.

Instant Karma PKI (IKP) [38] is an automized platform used to define and report wrong behaviors from CAs, to incentivize them to correctly issue certificates and the detectors to report non-authorized certificates as fast as possible. Participant HTTPs domains are able to publish domain certificate policies which specify the criteria the certificates must follow, and any violation of those policies constitute wrong behaviors from CAs. CAs can sell reaction policies, that refers to financial transactions to domains that must occur when a non-authorized certificate is reported. The domains affected by the certificate, the detector and the CA receive payment from those transactions so that CAs lose money when issuing expired or invalid certificates and the detectors receive money by reporting them. Figure 2.6 (from [38]) explains briefly how this scheme works.

The system is implemented in Ethereum, where the Smart Contract ecosystem offers a public mechanism in order to automatize the handling of reports from detectors and execute financial transactions, assuring fast responses to wrong behaviors from the CAs.

IKP is an extension of the standard TLS architecture and introduces two new entities:

- IKP authority: stores the information of CAs, public keys for authentication and financial account information to receive payments. It also stores domain certificate policies, given by the domains, and reaction policies, that automatically react when a report is received. The authority is responsible for execution those reactions and may be instantiated as a Smart Contract in Ethereum;

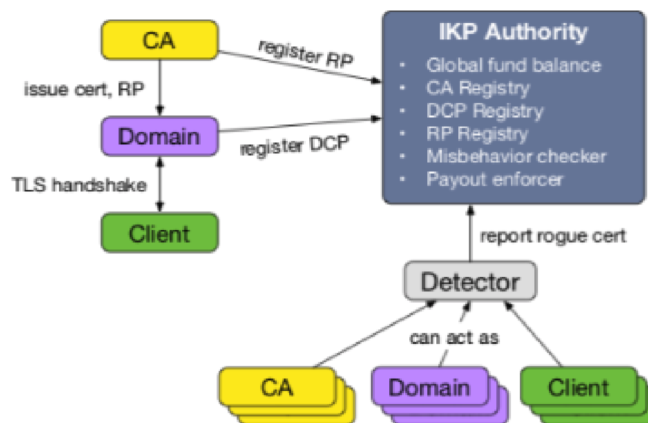


Figure 2.6: IKP schema

- Detectors: report suspect certificates to the IKP authority. Any entity can be a detector.

2.5.5 CertCoin

CertCoin [20] is based on the cryptocurrency Namecoin [43], which is based on Bitcoin and is developed to act as a decentralized DNS, and supports register and update of domains. Registration of a domain has a cost but its update is free. This works as an incentive for the works to mine blocks with CertCoin information. It takes advantage of the merged mining protocol of Namecoin where the miner is allowed to mine more than one block at the same time, increasing the security of all transactions.

The transaction regarding a domain registration contains two public keys that are associated with the bought domain, a public online key and an offline public key. The online private key is used to sign messages and verify their authenticity from the server that is hosting the website and the website itself. The offline private key must be stored in a secure offline storage in order to not be vulnerable to threats and is used to sign or revoke keys when in the presence of some type of compromise. When creating a new key for the domain, it will be signed with the old key, allowing the owner to prove ownership of the key or to trace the previous keys and signatures.

CertCoin uses cryptographic accumulators [19] to store tuples in the form of (d, pk, exp), for example, where d is the domain, pk the public key associated, and exp the expiration date of the key. Accumulators are the representation of a set of elements with constant size. After the addition of a new element, a witness is generated so it can prove that the element was in fact added. This avoids the overhead of storage that exists when each user stores all the Blockchain information and, instead, to solve that problem, each user stores only the current state of registered domains and keys.

Blockchains do not support key-value gets, which is important in PKIs to make queries

regarding public keys from specific domains. CertCoin proposes the use of an authenticated Distributed Hash Table (DHT), the Kademlia unauthenticated DHT [39] in particular. Although its use in CertCoin contains a few changes: use of digital signatures for authentication and integrity, assignment of unique Node IDs to resist Sybil attacks where an adversary can create several nodes with the objective to gain reputation, and a modified key recovery process to incentivize all nodes to participate and support the DHT. To contribute, each domain allocates a node to the DHT.

This approach removes the trust deposited in CAs, that can suffer from problems and attacks, and creates a domain repository where the miners are rewarded when mining blocks containing domain purchases and transfers.

2.5.6 PomCor: Backing Rich Credentials with a Blockchain PKI

The objective of the PomCor project is to identify five remote identity proofing solutions to use as an alternative to knowledge-based verification. In one of those solutions, described in [34], a rich credential is supported by a Blockchain PKI. A rich credential allows a subject to identify himself to a remote verifier without any prior relationship by using verification steps, such as private key, password, and biometric features.

As in others Blockchain-based PKIs, the hash of each certificate is stored in the ledger. The difference here is: hashes of valid certificates are stored in a ledger and hashed of revoked certificates are stored in a different ledger. In order to authenticate an entity, if its certificate is in the valid certificates ledger and not in the revocations ledger, then it is valid. This does not require a separate process that may be demanding in terms of processing and storage, making clients prone to skip this verification step (e.g. CRLs).

2.5.7 Blockchain-Based Certificate and Revocation Transparency

CAs can suffer attacks that make them issue incorrect certificates and, due to the lack of transparency in traditional PKIs, they are only noticed after some time, in which it is possible that SSL/TLS negotiations have already taken place involving those certificates.

Blockchain-Based Certificate and Revocation Transparency [64] takes advantage of the Blockchain characteristics where it acts as a public append-only logger to store certificates e revocation operations. Certificates signed by a CA and their information regarding the state of revocation are published by the subject/web-server as a transaction. Each subject has a publishing key pair to sign certificate transactions in order to manage its certificates while cooperating with CAs. This removes the total authority that usually CAs have in traditional PKIs by giving some control to the servers.

The system architecture uses some aspects of the bitcoin system. Miners append the transactions to the Blockchain after verifying the transaction and mining the block. Block in the Blockchain is similar to the one in bitcoin regarding the header and organization of transactions (Merkle hash tree). It is also possible to use an incentive mechanism where

the miners receive coins and those coins are purchased and consumed by the servers when they publish certificate transactions.

To validate certificates, a browser first communicates with a P2P network of servers and miners to download the block headers from the Blockchain, then it verifies if the headers are correctly chained and if each one has a valid PoW nonce and if everything is valid updates its local copy if a longer chain of certificates was received. Since the certificate chain was firstly verified by the miners, the browser only has to verify if it trusts on the trusted root CA associated to the certificate involved in the SSL/TLS negotiation, every other verification is made by the miners.

Although this offers an interesting way to achieve certificate transparency, introduces overheads regarding storage, certificate validation, and incentive costs.

2.5.8 SCPKI

In traditional PKIs, CAs sign certificates as a whole and sometimes it can be useful if single attributes can be signed instead, making possible to an entity vouch for another entity's attribute, such as its public key, name, address or other [1].

SCPKI [1] is a system that makes use of the web-of-trust model, where entities vouch for other entities instead of having a CA that issues certificates, and the Ethereum Smart Contracts so entities can create, manage and sign attributes.

Since Ethereum involves gas costs, SCPKI is designed so it is possible for the users to store big amounts of data outside the Blockchain (e.g. IPFS [5]) in order to have fewer costs. The hashes of data are stored in the Blockchain to keep their authenticity.

SCPKI is hosted on the Ethereum Blockchain and has two main components:

- Smart Contract: acts as an interface to the Blockchain to manage identities and attributes. It has functions to add attributes, used by an entity to add an attribute to its identity, sign attributes, so entities can vouch for other entities' attributes, and revoke signatures, in order to entities revoke their own signatures if necessary;
- Client: interacts with the Smart Contract and systems like IPFS so the users can search and filter attributes.

This solution has its drawbacks. The system is designed so that all the entities referenced by the system must be using it, that is, if an entity wants to vouch for another entities' attribute, the target entity must be in the system and have created that attribute. Regarding privacy, the system is not suited to publish more private attributes, since, in order to entities sign attributes, they must be made public.

2.5.9 Other Approaches

Alternatives, such as Certificate Transparency from Google, have a problem where if an attacker can obtain a valid but fake certificate and control the Certificate Logs, he can

develop a man-in-the-middle (MITM) attack by giving the client a fake view of the log that contains the fake certificate (split world attack) [30]. CertLedger [30] offers certificate and revocation state transparency by handling that problem. It uses a Blockchain-based public log to validate, store and revocate TLS certificates. The system is resistant to the split world attack due to the use of Blockchain. There is a decentralized public log mechanism with consensus between the clients that makes impossible for the attacker to develop that attack since the log is distributed and it is not possible to reverse its state, therefore he can't show a fake view of the log without the client noticing. Another improvement is the fact that the revocation process can be done by the domain, which removes some power from CAs because this process is normally only done by CAs. During the TLS handshake, the ClientLedger Client and a domain agree on the last block number and the domain sends the certificate with the proof that is generated from the State Merkle Tree of the agreed block. The client can verify if the certificate was issued for the domain and the proof through the state tree hash that is in the agreed block header. Clients do not need to make more validations because, when appending a certificate to the Blockchain, the validation is already made. The revocation status is verified through that state tree hash. Because of this, clients do not need CAs to verify the revocation state of a certificate and may verify it by using the state tree hash from the downloaded block header.

2.6 Discussion

PKIs are essential to the secure use of digital public-key certificates. Nevertheless, traditional PKIs have several problems, such as centralized management infrastructures and, therefore, trust centralization established by root-of-trusts and related oligarchies. Several potential problems emerge from the inherent trust-centralized model.

Blockchains, as disruptive technologies, can change the way PKIs can be designed, with decentralized and cooperatively managed trust assumptions, in the context of a new approach for Blockchain-enabled PKI solutions. Some approaches shown in previous sections use the Blockchain technology to implement PKIs to address different issues: to remove the centralization of trust and to avoid single points of failure, to propose management function using immutable decentralized ledgers, to offer transparency in the management processes, and to support revocation information, in order to be quickly disseminated by all the interested peers. Although discussed implementations address many problems and generic interests shared in our approach, some studied solutions have drawbacks or do not provide what we are looking for. The main identified drawbacks are the following:

- Many Blockchain platforms used by studied solutions, and their base service planes, are targeted for cryptocurrencies ([1, 4, 20, 34, 38, 66]), which limits the implementations due to their specific focus, volatility, and costs for the use of PKIs for

different purposes and different applications. Moreover, the service planes implement functions not particularly interesting for our approaches, such as the base consistency criteria for replication and related consensus primitives, the high-costs on energy needs, scalability assumptions and anonymous participation with no membership control.

- Several Solutions, such as [1, 34, 38, 66], use Ethereum, which has gas costs in inherently design considerations for permissionless platforms, not addressing the case for consortium-Blockchains that are in the more specific target of our approach.
- Some proposals use financial transactions ([4, 38]) as an incentive to guarantee correct behavior of CAs. We are looking for a solution that does not need monetary incentives, addressing a consortium model of participating entities interested in cooperating as members of a decentralized PKI solution, providing the same functionalities found in current PKIs, following, for example, the functions and roles defined in the PKIX standard framework.
- In PomCor [34] it is possible for a group of miners to create a fork where a certain certificate is not in the ledger of revoked certificates, wherein the *real* ledger is, and this may take some time to be noticed. In the meanwhile, the certificate could already have been used by clients (incurring in similar problems with asynchronous revocation processes in centralized PKIs purely supported by the periodic dissemination of CRLs).
- Some implementations are based on permissionless platforms, with the consensus mechanisms based on the PoW consensus style. However, there are serious performance issues regarding the validation of blocks (in the validation management of certificated during their operational life-cycle). The base energy problem of the Blockchain services is also an issue that will promote PKI solutions not energy-aware, and not efficient in dealing with energy vs. effectiveness vs. performance tradeoffs.
- We found in the studied solutions, that the addressed processes related to certificates and signatures of the Blockchains and PKI functions use certificates are in general bootstrapped under centralized assumptions, even targeting applications on top in a decentralized way. This seems to be paradoxical. In other words, the issuing process is out-of-cope of the promoted decentralized solutions, with the emphasis targeted only to the validation and revocation operations, with centralized bootstrap processes. We must also notice that in many current Blockchain platforms, the management of certificates is an orthogonal issue that must be solved externally, currently by using the current centralized PKI solutions. This is something that we can understand in a “separation-of-concerns"approach. However, from another viewpoint, it is an unmatched solution for the purpose of Blockchains in establishing decentralized trust-management conditions.

It would be ideal to follow design assumptions, where CAs or external centralized PKI services and components have a lower or no presence at all, where peers simply create and issue their own certificates in a cooperative way and gather trust from other peers. We also would like to separate the context of PKI from the application, where the application could be cryptocurrency-related or not. As far as we studied from the related work, no solution addresses the requirements that we want to address in our dissertation. As a summary, we emphasize the following characteristics not present in the state-of-art solutions:

- Cooperative issuing of certificates, with certificates authenticated by both cooperative Multi-Signatures or BFT Threshold Signatures, representing a cooperative CA.
- Smart Contracts with expressiveness to define the issuing, certification, validation and revocation rules and invariants, and required trust metrics consistently defined in the Blockchain
- PKI management functions supported in a consortium BFT-enabled Blockchain, where transactions are proposed to a subset of endorsement nodes (acting as CAs), but with membership control that also provide high transaction throughputs for the certificate issuing process.

We base our design and implementation criteria on the above analysis of limitations and drawbacks in the studied related work, designing our solution leveraged by an extended plane of services built for the HLF Consortium Blockchain Platform. In these extensions, we include BFT consensus primitives for the consistency criteria, Threshold Group Signatures, and the support of extended Smart Contracts. The extensions are provided in a prototype of the extended platform, initially developed in NOVA LINCS [21]. This is explained in the following chapter where we will address the system model considerations and architectural concerns, in the design of our proposed DPKI Blockchain enabled solution.

Chapter 3

System Model and Architecture

In this chapter, we propose our system model for a Blockchain-Enabled DPKI Solution, based on a permissioned-oriented collaborative consortium model. This solution aims to address the main problems of traditional PKIs, such as their centralized trust model. Clients will be able to issue and manage X509v3 certificates through transactions and by following security invariants and processing rules. These invariants and rules will be defined in Smart Contracts, which in the Hyperledger Fabric (HLF) Blockchain, are referenced as chaincode, and verified by the peers when receiving transactions proposals. The immutable public ledger allows any peer to verify the validity of all transactions and reproduce the current state of the certificates through them. We start by discussing an application scenario, followed by an overview of the system model design principles, entities, interactions, and requirements. We discuss the reference architecture and its components, and the adversary model.

3.1 Application Scenario

PKIs allow the use of digital signatures, asymmetric encryption, and other security technologies [59]. Through certificates issued by entities of a PKI, it is possible for many applications to interact with authentication and security. The main entity of a PKI is the Certificate Authority (CA), which issues digital certificates. A client sends a certificate request to a CA in order to obtain a certificate signed by the CA, being then able to use the certificate for various applications. The CA possesses a certificate signed by another CA in a chain where the root CA self-signs its certificate, where normally these root CAs certificates are stored in our devices to authenticate other certificates, and their signatures, from that chain. Entities may then verify if a certificate is valid by verifying its attributes and signature. This traditional model is the standard and several business models are

dependent on it.

The traditional model is complex and possesses several problems. If the keys of several certificates are compromised, the notification regarding their revocation it is not spread fast enough. The main problem is the fact that the system has a single point of failure. If a CA is compromised that may trigger a threat chain, where the certificates signed by that CA can be compromised and, besides, the CA may be unreachable. Another undesirable aspect is the lack of transparency regarding the way that certificates are issued and validated in CAs.

To address those problems, we propose a Blockchain-Enabled DPKI where we mitigate the problems related to centralization and transparency. The application scenario for our proposition is illustrated in Figure 3.1, as well as the different entities and their interactions, which are explained in detail further below. External clients may request functions of certificate issuing, revocation, Online Certificate Status Protocol (OCSP) requests, and others, through the system proxies. The proxies have direct communication with the Blockchain peers in order to send transaction requests regarding DPKI functions. We can see in the figure that an external client contacts a Proxy, for example for the issuing of an X509v3 certificate, and the Proxy contacts the endorsers, which are responsible to verify, through Smart Contract functions, and sign transactions and DPKI content. Internal clients are inside the Blockchain network but in other channels. One example of the participation of internal clients is the existence of a Supply Chain channel, where all the participants of the channel use certificates issued by the DPKI channel. The ordering service peers order all received transactions so the peers storing the transactions in their local ledger have the same order of events and results. Various organizations can participate in the system and agree upon the rules used in the issuing and revocation of certificates. A valid scenario is the participation of existing PKI-owning organizations to offer a DPKI, where each organization has an endorser in the network, to sign certificates and other DPKI content, and one Proxy to enable the communication with external clients.

3.2 System Model

In our system model, distinct entities work cooperatively in a Blockchain in order to support a DPKI. Instead of having a single and centralized CA, several nodes in the Blockchain network act as if each one was a single CA (Endorser Nodes). Each CA node signs a certificate received in an issuing request, where the issued X509v3 certificate ends up with multiple signatures instead of one, as in traditional X509v3 certificates. Issued certificates are then stored in the persistent and immutable ledger of the Blockchain to offer transparency.

To facilitate the external client's process to issue and manage certificates, there are Proxies that act as intermediaries, and clients send the same data to a Proxy that would send to a traditional CA. The Proxy handles the gathering of signatures and storage of

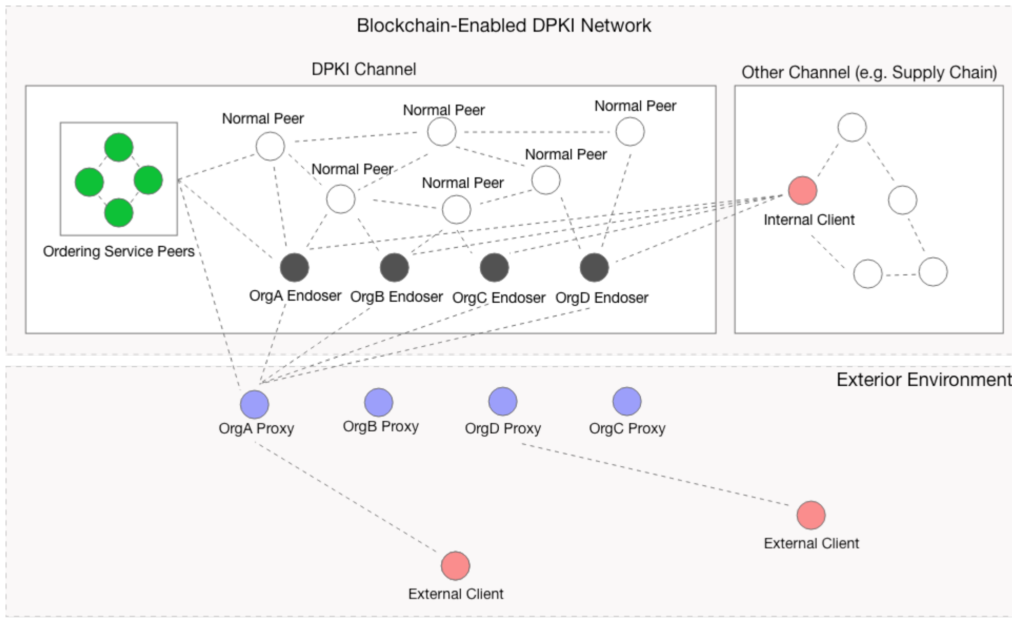


Figure 3.1: Application Scenario

the certificate in the Blockchain, where its behavior can be validated by contacting another Proxy or even the Blockchain directly. There may be internal clients, present in the Blockchain, that contact the CA nodes directly. HLF has the notion of channels, and our Blockchain-Enabled DPKI can be functioning in a specific channel, and in other channels can exist other types of applications such as Supply Chains, where the integrating elements request certificate related operations to the DPKI channel.

Different entities or organizations participate in the consortium Blockchain, each one possessing an Endorser Node that signs certificates and other DPKI-related material, and a Proxy. Functions in Smart Contracts validate certificates through specific validations, and the trust level of a certificate can also be computed through Smart Contract's functions when requested by an internal or external entity. Inspired by the Pretty-Good-Privacy (PGP) model, entities can sign issued certificates in order to increase their trust level.

3.2.1 Entities

DPKI Proxy. Facilitates the contact between the external client and the Blockchain-Enabled DPKI network. The external client contacts the Proxy to request operations of the DPKI (e.g. issue, revoke). There may, and should, exist more than one Proxy. If four organizations are cooperating in the consortium-based DPKI, each one has, for example, one endorser signing certificate issuing requests and, optionally, one Proxy. This removes single points of failure and a client can contact only one Proxy or several. When a Proxy receives a certificate request from a client, it is responsible to send the certificate signing request as a transaction to the Blockchain, collecting the signatures from the endorsers to insert in the certificate and sending it to the Blockchain for storage and to the client.

The system model supports two types of clients:

- **External Client**. The external client is the main client in the Blockchain-Enabled DPKI. This type of client is not aware of the entities inside the Blockchain, only knowing the outside Proxies information. In a traditional PKI, a client contacts a CA to request the issuing of a certificate by sending the correspondent Certificate Signing Request (CSR). In our system model, the client requests the certificate issuing in the same way by contacting the Proxy as if it was contacting a traditional CA, sending the CSR. The Proxy handles the communication with the Blockchain in the name of the client, returning the resulting issued certificate. The client may also request other types of operations such as Certificate Revocation List (CRL), the revocation status of a certificate through OCSP, and revocation requests. Information received from the Proxy that was obtained from the Blockchain is signed by the Blockchain entities that handled the request. This way clients can confirm that the received data was not tampered or created by the Proxy. In case of doubt, clients can go further and contact the Blockchain nodes directly.

- **Internal Client**. Entities inside the Blockchain may also request DPKI operations. This makes sense mainly in Blockchains with the notion of channels. The Blockchain-Enabled DPKI can be running in a channel and an entity from another channel (e.g. Cryptocurrency related) can request a certificate to use it in its channel.

The Blockchain supports the model and has several components and entities, offering services and different service plains. We detail this later in this chapter. The Blockchain entities are the following:

- **Endorser**. In our system model, endorsers not only validate and sign transaction requests but also sign certificates, CRLs and OCSP responses that are validated through Smart Contract functions.

- **Orderer**. In order to guarantee consistency and order regarding certificate operations (e.g. revocation), all types of transactions including certificate related, are ordered for storage in the Blockchain ledger by the peers.

- **Normal Peer**. While transactions and other content are signed by the endorsers, normal peers have the purpose of receiving, validating and storing the transactions changes in their local ledger, offering integrity and transparency. The transactions are delivered in blocks by the ordering peers.

3.2.2 Interactions

The main interaction in our solution is regarding the issuing of X509v3 certificates. The system model offers three types of interaction with the client, that are detailed further below. We will focus on the external client, which we foresee to be the main one. The internal client is expected to use the Self-Signed Certificate Model, where he acts as its own proxy, managing the received signatures.

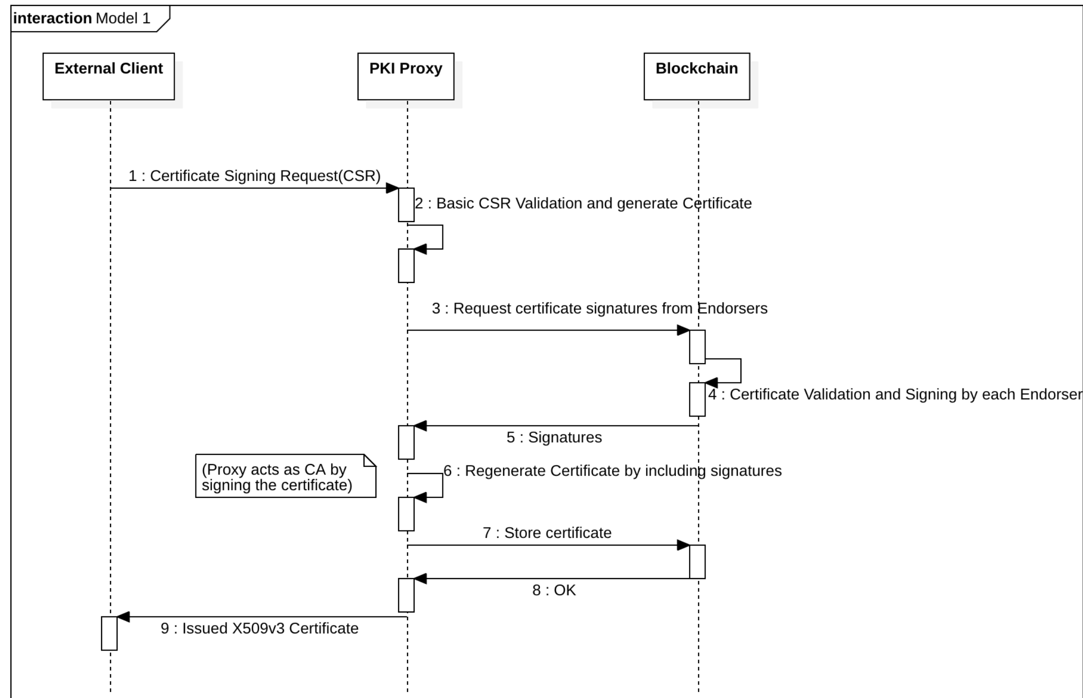


Figure 3.2: Standard Model Sequence Diagram

Standard Model. This interaction model is viewed as the main one, where the external DPKI client contacts the Blockchain-Enabled DPKI as if it was contacting a traditional CA. When requesting the issuing of a certificate, the client sends the same data that would send a CA, a CSR. This model is illustrated in Figure 3.2. The client contacts the DPKI Proxy, which acts as an intermediary, to request DPKI operations. The DPKI Proxy has a component configured to communicate with the Blockchain peers, having information such as their certificates stored.

1. The client sends a CSR to the DPKI Proxy, requesting an X509v3 certificate that is validated by a smart contract and signed by the endorsing peers.

2. The DPKI Proxy does some initial validation to check if the CSR has a valid signature and attributes. If everything is valid, it generates an X509v3 certificate from the client's CSR and signs it.

3. Having this initial certificate, the Proxy sends a transaction proposal request including the certificate to the Blockchain with the purpose of having the Endorsers validate and sign this certificate.

4. Each endorser validates the received certificate through a Smart Contract's function. The output of the validation function must be the same on every endorser. After this validation, each endorser signs the certificate and sends the signature to the Proxy.

5-6. The Proxy receives the signatures from the endorsers and validates them. It creates a new X509v3 certificate from the client's CSR again but now including the endorsers' signatures. The Proxy itself signs the created certificate, acting as a CA. This way the certificate has one standard signature (in the signature field) besides having the endorsers' signatures. Furthermore, we also obtain transparency, knowing which Proxy acted as an intermediary in this process.

7. For persistence, future validation and transparency, the Proxy sends a transaction request to store the issued X509v3 certificate in the Blockchain's persistent ledger. The endorsers validating the transaction validate all signatures created by the other endorsers regarding the certificate.

8-9 Upon receiving the confirmation from the Blockchain that the certificate was validated and stored, the Proxy sends the final X509v3 certificate to the client, finalizing the certificate issuing.

Self-Signed Certificate Model. If obtaining an X509v3 certificate issued by the Blockchain-Enabled DPKI is not desired by the client, he may send a self-signed certificate instead of a CSR to the Proxy, requesting validation and signatures from the Endorsers regarding its certificate. This model is illustrated in figure 3.3

1. The client generates a self-signed X509v3 certificate with the desired attributes and extensions and sends it to the DPKI Proxy as a request to obtain the validation and signature from the endorsers.

2-3. The DPKI Proxy validates the client's certificate and associated signature and sends a transaction proposal request including the certificate to the Blockchain in order to obtain the endorsers' validation and signatures.

4. As in Standard Model, each endorser validates the received certificate through the Smart Contract function. After this validation, each endorser signs the certificate and sends the signature to the Proxy.

5-6. The DPKI Proxy receives the signatures generated by the endorsers in the proposal request, validates and sends them to the client.

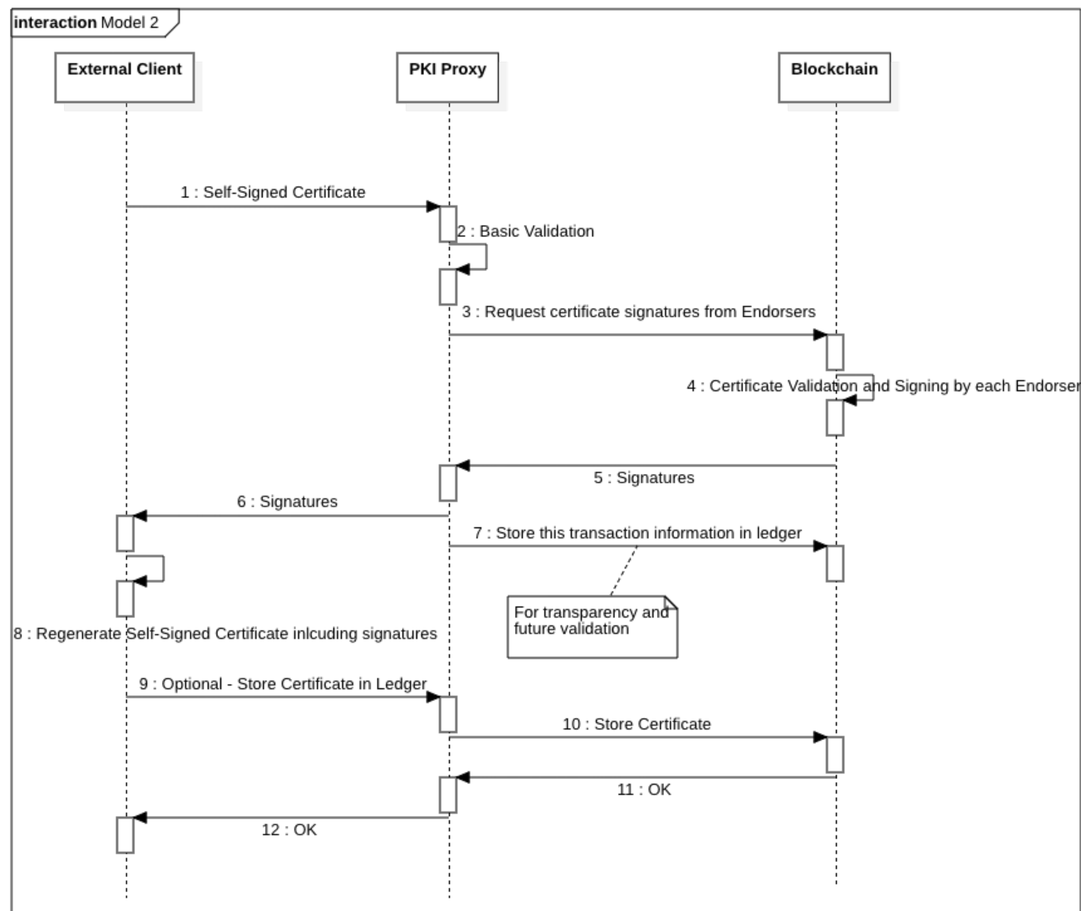


Figure 3.3: Self-Signed Certificate Model Sequence Diagram

7. The DPKI Proxy finalizes the transaction proposal by assembling the endorser's responses in a transaction and sends it to the Blockchain for final validation and storage. This is not strictly necessary to the issuing of the certificate but offers transparency by storing in the Blockchain the information related to this process.

8. Meanwhile, the client re-creates the self-signed X509v3 certificate by including the signatures of the endorsers.

9-12. Optionally, the client may specify in the request sent to the DPKI Proxy to store the final certificate in the Blockchain. This facilitates the process of other external entities to validate the client's certificate by contacting the Blockchain, which now has the certificate and other information related to the issuing process. The DPKI Proxy sends a transaction proposal to the Blockchain in order to store the certificate, sending a response to the client about whether the certificate was stored successfully or not.

CSR Model. A client may wish to have a certificate signed by a traditional CA that

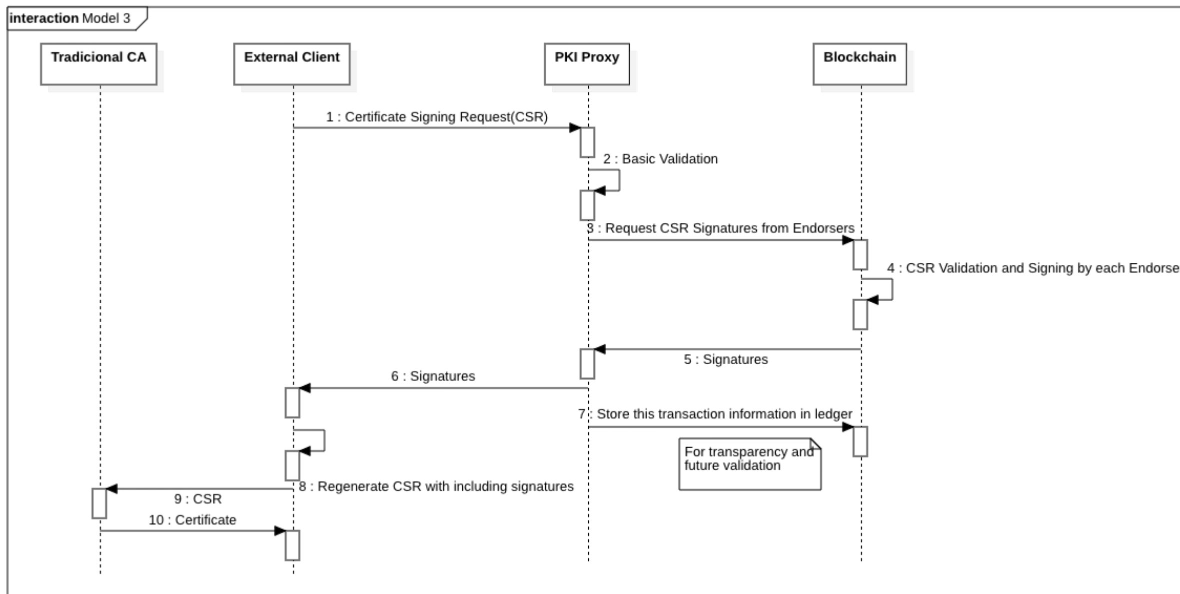


Figure 3.4: CSR Model Sequence Diagram

is already trusted in the outside environment. This model makes possible that a client possesses a certificate that was signed by the Blockchain-Enabled DPKI, through its endorsers, and by a traditional CA.

1. The client creates a CSR and sends it to the DPKI Proxy, requesting signatures from the Blockchain-Enabled DPKI.

2. The DPKI Proxy executes basic validation regarding the CSR's signature and attributes.

3. If the validation is successful, the Proxy sends a proposal request including the client's CSR in order for the endorsers to validate and sign it.

4. Each endorser validates the CSR attributes and signature, signing it if the validations are successful.

5-7. The DPKI Proxy gathers and verifies the signatures from the endorsers, sending them to the client. It also finalizes the transaction proposal sending the CSR and its signatures for storage, for future validation and transparency.

8-9. The client re-creates the CSR by including the signatures of the endorsers. The final CSR is sent to a traditional CA with the standard approach.

10. The traditional CA, whichever is chosen by the client, executes its process to validate the client's CSR and generates an X509v3 certificate. Although the CA must accept the field that includes the endorser's signatures.

3.2.3 Requirements

Our system model must hold the following requirements:

RQ1 BFT-Consensus and persistence of data regarding the read and write operations related to the DPKI.

RQ2 DPKI Decentralization with cooperative issuing of certificates, with certificates authenticated by cooperative Multi-Signatures or BFT Threshold Signatures, representing a cooperative CA

RQ3 Identity of the DPKI network participant entities and assure non-repudiation of operations.

RQ4 Auditability and transparency of any DPKI related operation executed in the past by any participant entity.

RQ5 DPKI should be accessible to internal and external entities of the Blockchain.

RQ6 A collaboration of organizations in the Blockchain in order to create Smart Contracts that dictate rules regarding the certificate issuing process.

RQ7 Issuing and revocation requests should have a similar processing work in the client such as in traditional PKI operations.

RQ8 Quick and easy obtention of revocation information by OCSP or CRLs.

RQ9 The trust level of certificates can change by having different entities signing specific certificates if they desire.

RQ10 Processing and obtention of the trust level of certificates, measured by Smart Contract functions.

RQ11 Smart Contracts with expressiveness to define the issuing, certification, validation and revocation rules and invariants, and required trust metrics consistently defined in the Blockchain

3.3 Reference Architecture

Our system aims to implement a decentralized PKI (DPKI) with auditability, transparency, and non-repudiation of operations. To achieve this, the base of the system is the Blockchain Technology. In order to have all the requirements in the Blockchain layer, we have an extension layer of the Blockchain services. For the service to be more accessible to the exterior, there is an API layer to facilitate the interaction and processing needed. All these layers together form the architecture illustrated in Figure 3.5. The architecture supports the DPKI, as also the necessary communications for interested entities and clients.

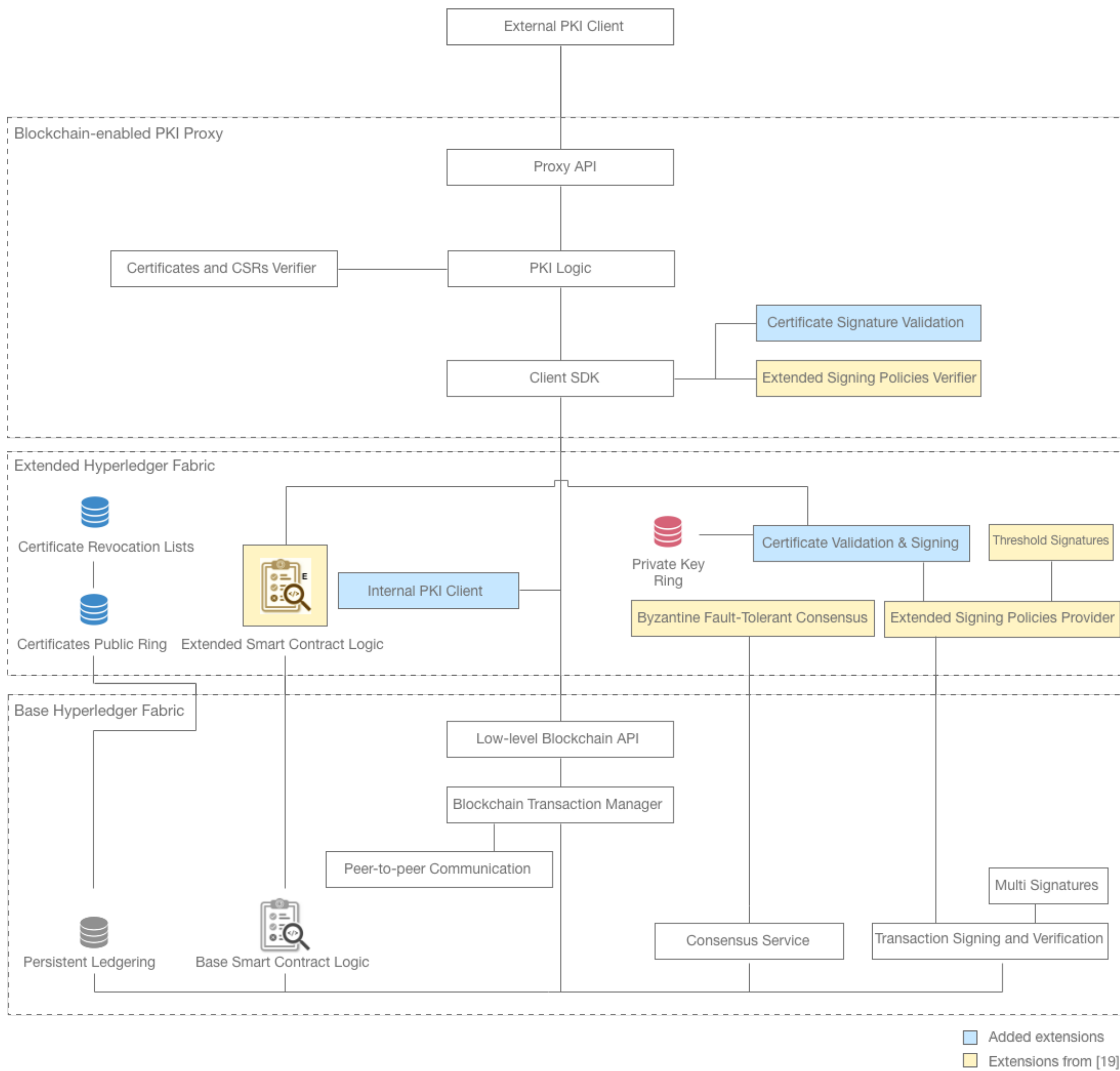


Figure 3.5: Architectural View

Support libraries for BFT-Consensus, BFT Threshold Signatures, and extended Smart Contract logic, some of the desired requirements for this system, are implemented in the work of [21], which extends the HLF, a consortium-based Blockchain, to have these requirements. Since we also will use HLF, because of its permissioned-oriented collaborative consortium model and the notion of channels, we will take the work from [21] and develop our solution having its extensions as the base, making the desired changes. Smart Contracts in the HLF Blockchain are referenced as chaincode, therefore, we will also use this term from here.

Base Hyperledger Fabric. HLF [26] offers some of the desired characteristics mentioned previously. Its consortium-like model enables the management of the different type of peers that coexist in the Blockchain network. However, it does not possess the Threshold Signature scheme that we desire and, although it is possible to plug consensus mechanisms, it does not have a BFT consensus mechanism as one of its defaults.

Extended Hyperledger Fabric. Fortunately, work from [21] extends HLF by adding to its base an extended Blockchain service layer that adds a fully decentralized threshold group-oriented signature verification process of transactions, and extended Smart Contracts that allow contracts not only to specify the properties of an application running on a Blockchain system but also the properties of the system itself and how transaction flows should occur. This extended layer also includes a consensus service [7] based on the BFT-SMaRt [6] consensus mechanism, a PBFT-like mechanism that was modeled to work with HLF. Besides those extensions, we added in the HLF's ledger two virtual components: a public ring with trust metrics of certificates and a private key ring. The public ring serves to store the Blockchain Public-Key Certificate State (BCS), explained further below, in which each peer has the same copy in its ledger, while the private key ring, stores the private keys (or signature shares when a certificate uses threshold signatures) of a peer, therefore being specific to each peer. Changes to the HLF's Extended Signing Policies Provider (XSPP) were also made in order to the endorsers sign certificates, with multi-signatures or threshold signatures. Through these extensions, internal entities in the Blockchain will be able to request certificate issuing.

Blockchain-Enabled DPKI Proxy. To facilitate the process of DPKI operations when involving external clients (requirement RQ5), a Proxy layer was implemented. This is important in order to external clients have the same process (e.g. same content format in the requests) with the Blockchain-Enabled DPKI that they would have with a traditional centralized process, following the requirement RQ7. The Proxy is able to communicate with the Blockchain network through the client SDK, sending DPKI related transaction proposals and receiving responses, also verifying transactions and certificates signatures. It executes basic validations before sending the client's requests to the Blockchain. An API is offered to request DPKI operations regarding issuing, verification, and management of certificates.

These layers form our Blockchain-Enabled DPKI solution, where clients do not have more burden requesting DPKI operations and there is transparency regarding transactions and certificates.

3.4 Software Architecture Components

3.4.1 Base Hyperledger Fabric

This is the layer in which our solution depends on. It could be used any permissioned-oriented collaborative consortium Blockchain but, as mentioned previously, we chose HLF due to its channels concept, being open-source, and having the work from [21] that extends the HLF Blockchain with functionalities that we desire, such as BFT consensus and BFT threshold signatures. It has Smart Contracts technology, named chaincode, to define rules and functions that, in our system, will be used to define the issuing,

revocation, and management of certificates.

Each network node has a copy of the ledger where it is stored information from transactions that were fully ordered by the consensus protocol. This Blockchain has a high performance of transactions throughput. Being permissioned, the participants are known among themselves and not anonymous. They may not fully trust each other but the network can be operated under a model with legal agreements or frameworks to manage disputes.

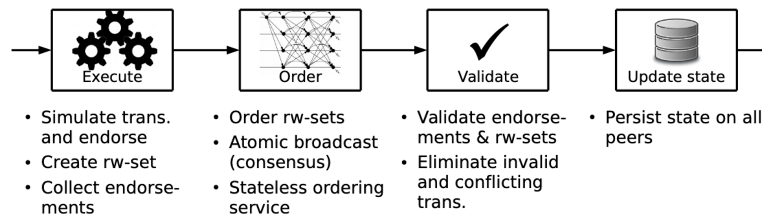


Figure 3.6: Hyperledger Fabric Architecture

It is possible to insert consensus protocols. This is focused in more detail in the Extended HLF layer components. It has also a new architecture for transactions, execute-order-validate, shown in Figure 3.6 (from [26]), where:

1. Execute: Endorser executes the transaction and verifies its correctness, endorsing it if the result is valid.

2. Ordering of transactions through the pluggable consensus protocol.

3. Validate: validate transactions through endorsement policies before committing them into the ledger.

This allows parallel executions by eliminating non-determinism since the inconsistent results in the execute phase can be rejected before the ordering phase. Performance and scalability are improved because of this architecture.

The transaction flow is shown in Figure 3.7 (inspired by [26]) and is briefly the following:

1. The client creates a transaction proposal to invoke a chaincode function with input parameters and sends the proposal to the endorsing peers.

2. The endorsing peers, nodes responsible for the endorsement of transactions, verify the transaction (e.g. clients' signature and the result by simulating the chaincode function). In this step, there are no updates to the ledger. The simulation generates read and write sets that, along with the endorser signature, are sent to the client.

3. The client verifies the endorsers' signatures and checks if the results from each response are identical and, if so desires, submits the transaction to the ordering service.

4. The ordering service does not verify the transaction content, it just receives and orders transactions in blocks, to send them to the Blockchain peers.

5. Peers receive and validate transaction in blocks through the endorsement policy, and ensure that no changes were made in the read variables since the transaction was endorsed. Transactions are marked as valid or invalid. Each peer stores the block with transactions in the ledger and commits the write sets. Invalid transactions are also registered in the ledger.

6. The client is notified that the transaction was appended in the ledger. It is also informed if the transaction was validated or not.

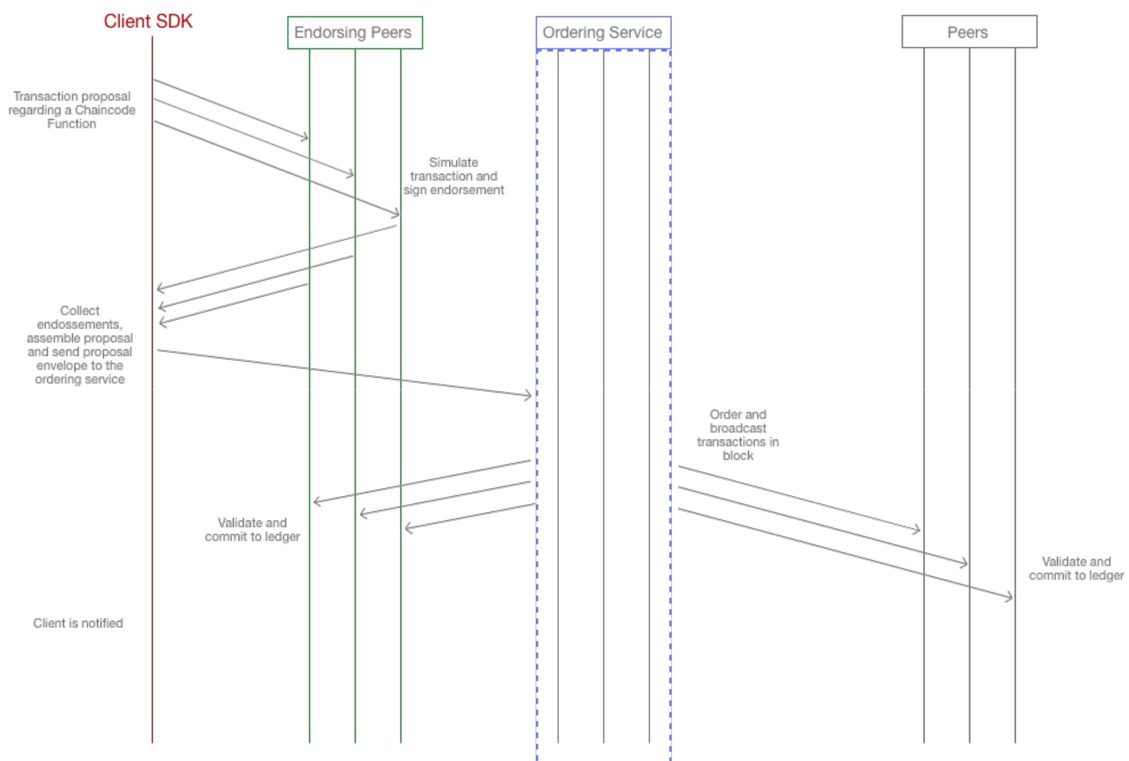


Figure 3.7: Hyperledger Fabric Transaction Flow

3.4.2 Extended Hyperledger Fabric

BFT Consensus. The base Hyperledge Fabric, although allowing the insertion of several consensus algorithms, does not bring any BFT algorithm incorporated, only crash-fault. We wish to have support for BFT consensus in our security requirements. We are not interested in the consensus algorithms used in cryptocurrency-based Blockchains due to their performance and attack vectors. The HLF extension on which we are building our solution embedded a BFT ordering service [7] that is designed and implemented

to incorporate in the HLF's consensus mechanism. This service is built on top of BFT-SMaRt [6], which is based on a PBFT algorithm and receives transactions that are waiting to be stored in the Blockchain ledger, to group them in blocks, distributing those blocks through the Blockchain network nodes in order to validate and store them in the ledger. The BFT-SMaRt algorithm assumes the standard BFT-SMR model in which there are *3f + 1* replicas to tolerate f byzantine faults. It is open-source and was the first BFT SMR library to support reconfigurations of sets of replicas and to offer transparent and efficient support for durable services. In this consensus service, depicted in figure 3.8 (from [7]), built for HLF, the consensus nodes execute a BFT consensus protocol. There is a set of *3f + 1* ordering nodes and an arbitrary number of frontend nodes. The ordering nodes execute a distributed consensus protocol to establish the total order of transactions. The frontend nodes execute the relay of transactions sent by clients to the consensus protocol. The authors from this extended HLF [21] made some changes to this consensus service to make adaptations to allow, for example, the submission of transactions to the consensus replicas by multiple organization memberships, since in this extension multiple organizations have more than one signature per transaction.

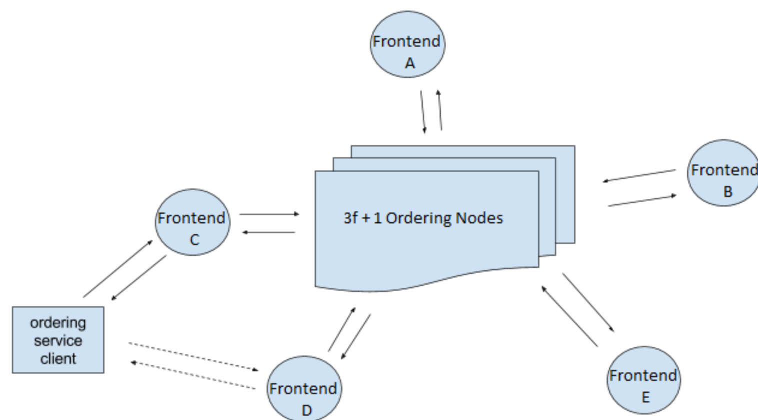


Figure 3.8: BFT Consensus service for Hyperledger Fabric

XSPP. Network nodes sign transactions to endorse them. Generally, the scheme used in this type of Blockchains is multi-signatures, which is present in the base HLF. Each endorsing peer signs a transaction and the envelope that contains endorsers' signatures, built by the client with the received signatures, has a signature for each endorser, using the standards RSA or ECDSA. In order to improve the fault-tolerance of the system, the work from [21] incorporated the extension XSPP, which contains signing policies based on threshold signatures, that are explained in section 2.4.4. This signature scheme improves the signature verification by using less storage and verifications when compared to the multi-signature scheme since only one signature is validated. This scheme also removes single points of failure, wherein multi-signatures, if the verification or creation of a signature fails, the transaction involved may become invalid. This extension adds

signing and verification functions to the transaction flow when using threshold signatures. The signature scheme to use in the endorsement of transactions is specified in a Smart Contract.

For our solution, we also made changes to this component in order to have endorsing peers sign certificates with the specified signature scheme, which works as the transaction signing process.

Threshold Signatures

Threshold signatures are implemented in a cryptographic service (implemented in [21]) as a service plane of the Extended HLF. The signature models correspond to the model from [61], which were previously discussed.

Certificate Issuing

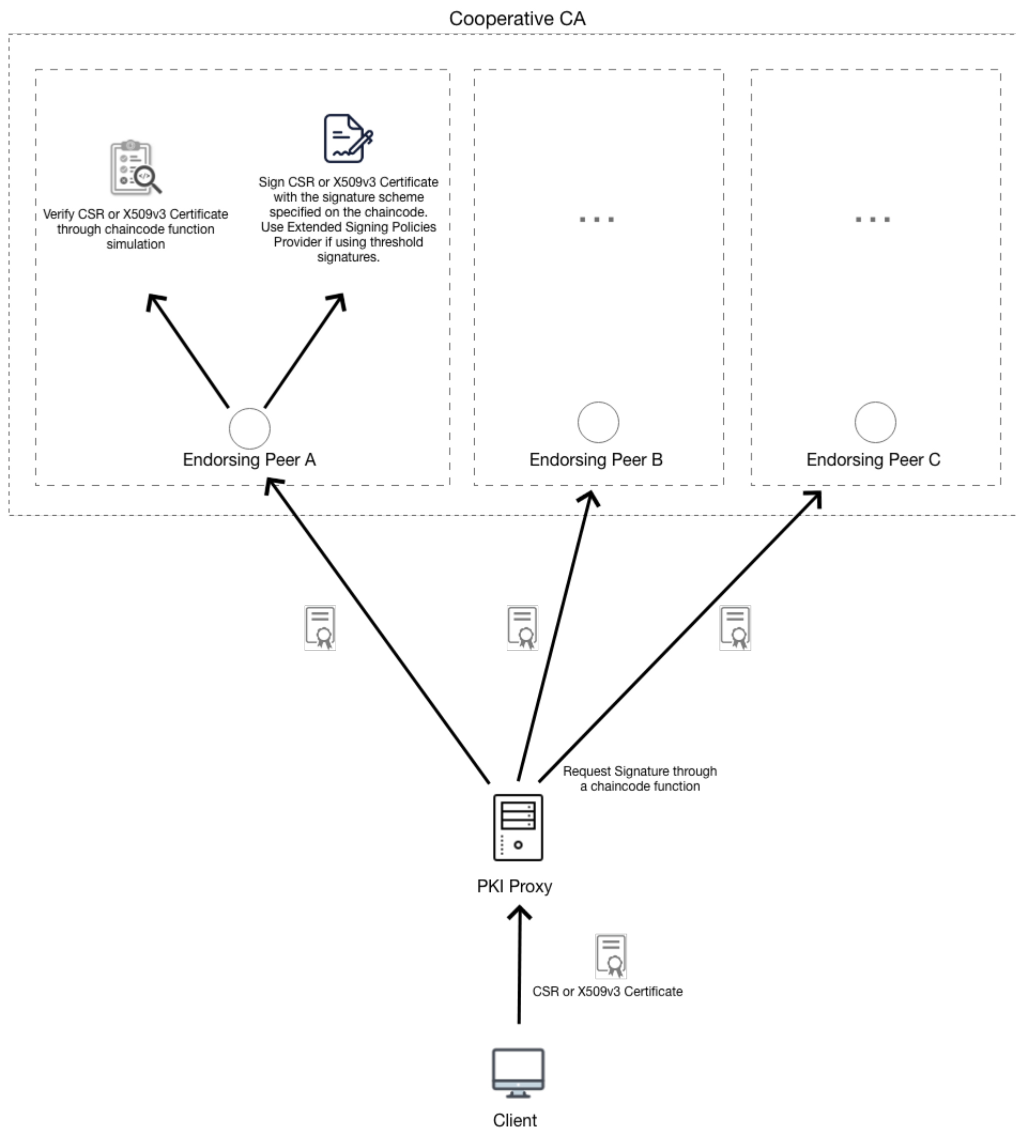


Figure 3.9: Certificate Validation and Signature

In this system it is possible to request signatures from the endorsing peers, that are seen as a cooperative CA, through the invocation of a chaincode function, giving a CSR or X509v3 certificate as input. A simplified version of this process is shown in figure 3.9. Each endorsing peer receives the CSR or X509v3 certificate to sign and verifies it through the simulation of the function invoked by the client to request the signatures. This function has verifications that may vary from function to function. If the verification is successful, the endorsing peer makes a request to the component responsible for signing transactions and certificates, Endorsement System Chaincode (ESCC), to create a signature from the CSR or certificate. This component queries the chaincode to obtain the signature scheme to use. If multi-signature is the scheme specified, it uses the default mechanism. If it is threshold signature, uses the XSPP to create a threshold signature with its private share. At the end of this process, if successful, the endorsing peer responds to the transaction proposal by sending the built signature. The client takes all the received signatures, verifies them, and inserts them into the updated CSR or certificate. This process is similar to the transaction signing process, the main difference here is the CSR or certificate verification in the chaincode function.

Private Key Ring

Certificate signing keys are stored in each endorsing peer as each one has a private key ring, each private key belonging and stored in a single peer of course. Various types of keys may be stored for multi-signatures and threshold signatures. New keys or shares (threshold signatures) can be stored over time. The component responsible for signing certificates reads these keys to sign a CSR or X509v3 certificate.

Public Ring

Certificates and their related information can be stored in the Blockchain ledger, maintained by each participating peer. Every action related to the management of certificates, either successful or not, can be stored in the ledger for transparency and future validations. In order to facilitate the reading of certificates information, their information could be grouped and condensed in an easy to read structure. An example of a structure is the **Blockchain Public-Key Certificate State** (**BCS**) table, shown in figure 3.10.

Timestamp	UserID	PK X.509 Certificate	Key Legitimacy	Signatures
...	...	...	...	...
...	..	...	...	...
Time at which the certificate was considered valid	UID + Certificate Hash (UID || H(cert))	X.509 Certificate issued by a peer	Level of trust of the certificate in the network. Smart contract defines the parameters related to this level of trust	Signatures of peers that signed this certificate. Can be a single signature, multi-signatures, or a threshold signature, depending on the rule defined by the Smart Contract
...	...	...	...	...
...	...	...	...	...

Figure 3.10: Blockchain Public-Key Certificate State (BCS)

The BCS would be built by running through all transactions related to certificates in the public ledger and using their information, present in every peer, to construct the

table. These transactions include issuing or revocation of a certificate, signing of a peers certificate by any other peer, and others. For instance, the column related to the certificate signatures can be built by going through all the transactions that involve a peer signing that certificate. We should reinforce that this BCS is not initially visible as a table in the public ledger, the idea is that the data needed to build it is present in the ledger. After building it, the table may then be stored in the ledger. As such, if we wish to have a version of this table present in the ledger, it is needed that a peer builds it. We see two ways of doing this, either there are peers with a specific role designed to construct this table when needed or between time intervals, or any peer is allowed to do it and receives a monetary reward for putting in the computational time and effort.

Certificate Trust Level

Another aspect present in our solution is the possibility of having a level of trust in certificates, measured by Smart Contract functions. This is inspired by the PGP's Web-of-Trust scheme, where users have key rings in order to store the public keys of other users and assign levels of trust to each entry since public keys collect signatures throughout time. In our system model certificates may have different levels of trust, depending on the signatures that certified them and computations from Smart Contract functions. The fact that the trust level is derived through defined Smart Contract functions and not by each individual user, addresses the asynchronous process of convergence issue in the Web-of-Trust model, where there are different management and storage of keys by each user. Certificates are signed by the same group of endorsing peers acting as a cooperative CA, offering an amount of consistency regarding the signatures on each certificate. The level of trust starts to deviate when different entities sign different certificates and Smart Contract functions compute the level of trust of certificates based on the number of signatures and which entities signed them. As expected, this level of trust will probably mean nothing to entities outside the Blockchain environment, since they are not aware of the system logic of certificates issued in the Blockchain-Enabled DPKI. This is mainly useful for applications running in different Blockchain channels, aware of the system logic, or external applications that somehow follow this logic. A client requesting the issuing of a certificate through the Blockchain-Enabled DPKI may not want any part of this level of trust model, choosing not to participate in this logic. Figure 3.11 shows an example of the level of trust of an X509v3 certificate, different chaincodes may generate different levels of trust for certificates.

Extended Smart Contracts

Another extension added in the work from [21], that is also useful for our solution, is the Extended Smart Contract Logic. Succinctly, it is a structure that consists of having three types of property sections in Smart Contracts or chaincodes:

- Base: existing properties in the base chaincodes. Contains properties such as the chaincode ID.

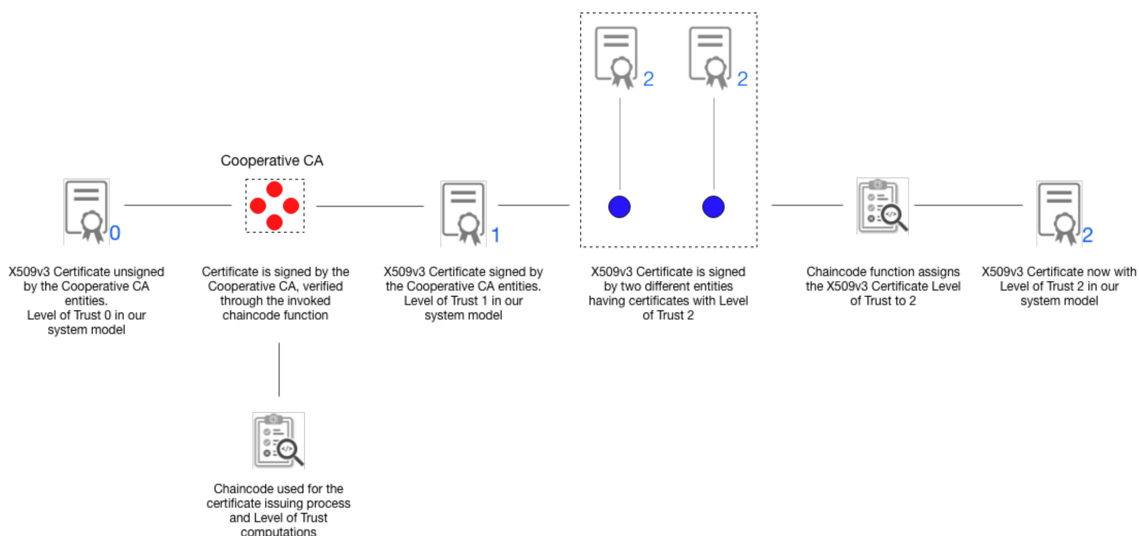


Figure 3.11: Example of Certificate Trust Level

- Extended: properties related to the extended layer. Examples of useful properties to incorporate are the signature scheme to use in transaction endorsement or the number of nodes necessary to sign transactions related to the chaincode.

- Application: any application-related property.

Different properties can be incorporated into different chaincodes as needed. For our solution, the needed properties are the signature mechanism to use when signing certificates, which can be different from the one used in transaction signing, nodes in the cooperative CA, and others. A structure example of extended properties is shown in Listing 3.1. We interpret DPKI-related properties as extended system properties and not application properties, mainly due to the fact that several of those properties will be read by internal Blockchain components when signing and managing transactions and certificates.

Listing 3.1: Example of an Extended Smart Contract

```
1  ExtendedSmartContractProperties DEFINITIONS ::= BEGIN
2
3      Transaction ::= SEQUENCE {
4          Id              OBJECT IDENTIFIER,
5          Payload         BIT STRING
6      }
7      LedgerData ::= SEQUENCE {
8          Key             OBJECT IDENTIFIER,
9          Value           BIT STRING
10     }
11     BasePlatformSystemProperties ::= SEQUENCE {
12         ContractId      OBJECT IDENTIFIER,
13         TransactionLog  SEQUENCE(SIZE(0..999)) OF Transaction,
14         LedgerRecords   SEQUENCE(SIZE(0..999)) OF LedgerData,
15         ...
```

```
16      }
17      ExtendedSystemProperties ::= SEQUENCE {
18          TransactionSignatureScheme      INTEGER(0..3),
19          TransactionWitnessNodes         SEQUENCE(SIZE(0..10)) OF IA5String,
20          CooperativeCASignatureScheme    INTEGER(0..3),
21          CooperativeCAWitnessNodes       SEQUENCE(SIZE(0..10)) OF IA5String,
22          CertificateIssuingRules         SEQUENCE(SIZE(0..999)) OF IA5String,
23          ValidFrom                       IA5String,
24          ExpiresOn                       IA5String,
25          AvailableDPKIFunctions           SEQUENCE(SIZE(0..20)) OF IA5String,
26          ...
27      }
28      ApplicationProperties ::= SEQUENCE {
29          ...
30      }
31  END
```

Internal DPKI Client

Entities within the Blockchain can also execute requests related to the management and issuing of certificates. This is mainly useful for nodes of other channels, where a node may participate for example in a supply chain channel and wishes to invoke a chain-code function from the Blockchain-Enabled DPKI channel to request signatures for its certificate. The certificates used in that channel could even operate fully through the management and signatures of the cooperative CA. Internal clients, although they can, will probably not have the necessity of using a Proxy to invoke DPKI-related operations, since they can be aware of the existing nodes that they need to contact to request certificate related operations. Even nodes inside the DPKI channel may invoke this type of operations in order to collect more signatures or revoke an owned certificate.

3.4.3 Blockchain-Enabled DPKI Proxy

This layer is responsible for the intermediation between the external client and the Blockchain.

Proxy API. This API allows external clients to contact the Proxy in order to make requests related to certificates regarding their verification and management. The requests that should be available are the following:

- issueCertificate(CSR)
- dpkiSignaturesRequest(X509v3SelfSignedCert or CSR)
- getCertificateRevocationList()
- getOcsp(certificateNumber, detailLevel)

- revokeCertificate(serialNumber, signature)

DPKI Logic. In receiving and sending requests related to certificates in several formats (e.g. PEM, PKCS#12), it is necessary to have the processing of those formats as their conversions as well. This component is also used to take the Blockchain endorsers signatures and insert them in new or existing certificates. We follow the standard model X509v3. Signatures of the Blockchain-Enabled DPKI are inserted as an extension in an X509v3 certificate. This is shown in Figure 3.12 where it specifies the type of signature used (e.g. multi-signatures or threshold signatures) and the collection of signatures (or signature shares if using threshold signatures).

Verification of CSRs and certificates. Despite the final and main verifications are made in chaincode functions, with previously established rules, Proxies may execute simple initial verifications. This allows, for example, to have some filtering of certificate requests. A possible idea is for the Proxies to have some sort of cache for common requests, in order to avoid to constantly contact the Blockchain to request revocation lists, state of certificates, and others.

Client SDK. This component is responsible for the communication with the Blockchain nodes. It has the necessary material to communicate with the nodes with authentication. Since the Proxy, normally, belongs to one of the entities participating in the Blockchain network, that material is previously installed and configured. It will have access to information regarding the available functions of the various chaincodes installed in the DPKI channel. If it does not have information of a certain chaincode, it is possible to obtain it through a query to the Blockchain. It builds the transaction proposals to invoke chaincode functions with certain parameters. Those functions will be mainly related to the issuing and management of certificates. Transactions are signed with the client SDK credentials. It also receives the endorsers' responses regarding transaction endorsements and certificate signatures.

Certificate Signature Validation. This component is in charge of validating signatures, mainly from endorsers, regarding certificates and their issuing process. In the construction of a Proxy, it is installed, safely, the cryptographic material (e.g. public keys of endorsers, and or threshold public key if threshold signatures are also being used) necessary for these validations. When the Proxy asks for the endorsers' signatures concerning a certificate, it verifies them before inserting them into the certificate. An external client may also execute these validations if it has access to the public keys.

XSPP. This component comes from the extension work in [21]. It is a modification of the HLF's standard client SDK in order to read and validate transactions signed with the threshold signature protocol. We also modified it to read the endorsers' signatures regarding certificates.

Version
Certificate Serial Number
Signature Algorithm Identifier
Issuer Name
Period of validity
Subject Name
Subject's Public Key Info
Issuer Unique Identifier
Subject Unique Identifier
Extensions
...
Signature Type (Multisignature / Threshold Signature)
Collection of Signatures
Certificate Signature

Figure 3.12: Blockchain-Enabled DPKI X509v3 certificate

3.5 Adversary Model Considerations

In order to have a suitable and practical solution for a Blockchain-Enabled DPKI, we need to define an adversary model, which is explained ahead.

The primary processing is in the Blockchain and its related services. We need to address that there may exist external threats with the objective of compromising the system processes, transactions, consensus, and communications. To maintain a secure and functional DPKI it is necessary to guarantee that the decisions and the storage in the ledger are not compromised. The BFT Consensus integrated in the extension layer by [21] brings those guarantees.

It is also correct to assume that it may exist internal threats. Nodes inside the Blockchain may show byzantine behavior when processing transactions and endorsements. Since we are implementing a DPKI, with sensible operations regarding issuing and revocation of certificates, we should have an elevated secure level tolerant to intrusions with BFT assumptions. BFT Consensus is also suitable for this type of threat. We assume that the minimum number of correct nodes is never compromised to maintain the conditions of liveness and safety of the BFT consensus protocol.

Communication channels may also be compromised by adversaries. Because of this, communications should be secured with TLS support, joined by multi-signatures or BFT threshold signatures in the endorsement and verification of transactions and certificates.

Proxies are controlled by entities that participate in the Blockchain network, we do not assume that more than a minimum (e.g. BFT consensus minimum) are compromised. If for example a single Proxy is compromised, the Proxy is removed and the external client contacts another Proxy

Denial of Service (DoS) and Distributed Denial of Service (DDoS) attacks are out of the scope of this dissertation and we did not consider them in the system model.

3.6 Summary

In this chapter, we presented the system model for the Blockchain-Enabled DPKI. The entities and their interactions with the system were briefly explained, as well as the main components that are required for deployment. We used HLF as the Blockchain component in the presentation model due to the characteristics previously mentioned. System requirements were also defined in order to build a suitable and secure model. Considerations were made regarding the adversary model and its impact on the construction of the presented model, which we also discussed. In the following chapter, we will present the prototype that was developed through the system model, detailing further the aspects of a possible implementation.

Chapter 4

System Implementation

In this chapter, we present the prototype developed by following the system model proposed in Chapter 3. We start by presenting the prototype overview and technologies, and then we present the prototype implementation. The prototype code regarding the Blockchain network, Proxy, and extensions of HLF is in the central repository[1].

4.1 Prototype Overview and Technologies

The main services and components of our prototype are the following:

- **Blockchain services**. For the Blockchain services, we used the Hyperledger Fabric (HLF) [26] platform, due to its characteristics, mentioned in previous chapters, with support for threshold signatures and BFT consensus through the extension made by [21]. We also made changes to the implementation to enable the signing of DPKI content given by specific Chaincode functions. The implementation code of HLF and its extensions is supported in Golang language, and the version used is Go 1.12.1. The consensus mechanisms and signature components made by [21] are made in Java language (Java 9) and use UNIX domain sockets for the communication between the Golang and Java processes. The exchange of information with HLF is made through gRPC[2] with Protobuf serialization.

- **Proxy component**. Acts as an intermediary to facilitate and standardize the relation of the client with the Blockchain network. Executes basic conversions and verifications before relaying the client requests to the Blockchain. It is also in charge of some DPKI operations such as inserting the endorsers' signatures in a new issued

[1] Central repository of the Blockchain-Enabled DPKI prototype: `https://github.com/miguelreisa/thesis-prototype-general`

[2] gRPC framework: https://grpc.io/

X509v3 certificate. For the proxy server, it was used Java language (Java 8) with the Spark Framework 2.8.0[3] to implement a REST API to provide endpoints so the client is able to communicate with the Proxy. The messages exchanged between the client and Proxy are in JSON format. Certificates, Certificate Signing Requests (CSRs) and other types of DPKI information are in Base64 when exchanged between the client, the Proxy, and the Blockchain peers. The Proxy interacts with the Blockchain by using an extended implementation of the HLF Java SDK 1.1, made by [21] to function with threshold signatures, and was further extended by our work to receive and interpret DPKI-related signatures. Bouncy Castle[4] version 1.59 was also used for the cryptographic functions.

- **DPKI Chaincode**. We address the HLF Chaincode created to execute DPKI-related operations as a component, due to its importance in this system. The Chaincode program chosen to run DPKI operations runs in a secured Docker container that isolated from the endorsing peer process. The result of the called function must be the same in all computations called by endorsers to be valid. If, for instance, the verifications of a certificate issuing process do not have the same result in all computations, that process is aborted. As the HLF code, the Chaincode is also in Golang with the version 1.12.1.

- **DPKI Content** For all the DPKI-related data, the RFC 5280 X509v3 [29] format was used. This standard is in all issued certificates, Certificate Revocation List (CRL), and Online Certificate Status Protocol (OCSP) responses.

- **Test tools and benchmarks** to validate and analyze the implementation and its difference to traditional PKIs.

Docker containers were used in order to be possible to run multiple Blockchain network nodes in a single machine. Each node has a dedicated container for its local ledger. This virtualization facilitated the implementation of the prototype, as its benchmarks. It was used version 18.09.03 of Docker CE[5] and version 1.23.2 of Docker Compose[6].

4.2 Prototype Implementation

4.2.1 Blockchain Network

As mentioned previously, HLF was used due to its characteristics and extensions from [21]. For the implementation of the Blockchain-Enabled DPKI, the main changes were made in the Endorsement System Chaincode (ESCC) component, for the signing of DPKI content, which explained further in Subsection 4.2.4. The Chaincode explained in Subsection 4.2.3

[3]Spark Framework: http://sparkjava.com/
[4]Bouncy Castle: https://www.bouncycastle.org/
[5]Docker CE: https://docs.docker.com/install/
[6]Docker Compose: https://docs.docker.com/compose/

is another important component for the management and validation of DPKI-related functions that produce DPKI content to be signed by the endorsers through the ESCC component.

4.2.2 DPKI Proxy

The proxies are the gateway between external clients and the Blockchain-Enabled DPKI network.

4.2.2.1 Proxy REST API

A REST API is exposed by the Proxy to enable clients the request of issuing and management of certificates. The endpoints are briefly explained in Table 4.1 and detailed further below.

- **Issue X509v3 Certificate:** the proxy receives a CSR in order to create a new X509v3 certificate created and signed by the Blockchain-Enabled DPKI system. The proxy that receives the request executes the following steps:
    1. Verifies the CSR content and signature (basic verifications since the Chaincode function is responsible for verifying if the specified rules are followed).
    2. Creates the associated X509v3 certificate from the CSR received, with pre-established start and expiration dates. The DPKI Chaincode then verifies if the dates are correct, accepting or not the certificate to be issued.
    3. The proxy acts as a traditional CA, signing the certificate and storing the signature in the associated fields. This also offers transparency regarding the Proxy that was responsible for the issuing process of this certificate.
    4. Sends a transaction proposal using the DPKI Chaincode function responsible for the issuing of certificates. This proposal is used to gather the signatures from all endorsing peers, which are endorsing this certificate. The Chaincode function process is explained further in this chapter. The only argument of this proposal is the generated X509v3 certificate in Base64 PEM format.
    5. After receiving the responses from the proposal, and confirm that they are coherent (e.g. the output of the simulation of the Chaincode function is the same in all endorser responses), for each response:
        - 5.1 Checks if the response contains a signature regarding the DPKI content of the transaction, which in this case is the signature of the endorser regarding the X509v3 certificate created by the proxy.
        - 5.2 Verifies the transaction signature. If using threshold signatures, uses the functions added by [21] in the SDK extensions.

Endpoint	Description
/getChaincodeDefinition	By using this endpoint, external clients can obtain information about the Chaincode and Blockchain-Enabled DPKI such as the cryptographic algorithms used and endorsers public keys. This way any external client can obtain enough information to verify X509v3 certificates, OCSP responses, and CRLs issued by the Blockchain-Enabled DPKI.
/issueX509v3Certificate	Receives a CSR in order to create a new X509v3 certificate, created and signed by the Blockchain-Enabled DPKI system. The newly issued X509v3 certificate is returned in Base64.
/signCSR	Receives a CSR to obtain signatures from the Blockchain-Enabled DPKI system in order to use it to issue certificates with another DPKI. The CSR, with the added signatures, is returned in Base64.
/signSelfSignedX509v3Certificate	Receives a Self-Signed certificate to obtain signatures from the Blockchain-Enabled DPKI system. The self-signed certificate with the added signatures is returned in Base64.
/revokeX509v3Certificate	Revoke a certificate and issue a new CRL with this certificate included. Receives the certificate serial number and a signature created by the client to ensure that the entity requesting the revocation is the owner of the certificate. The newly issued CRL is returned in Base64.
/getX509v3CertificateBySerialNumber	Search a certificate issued by the Blockchain-Enabled DPKI with the given serial number. Certificate is returned in Base64.
/crlRequest	Obtain the Blockchain-Enabled DPKI last updated CRL in Base64.
/ocspRequest/:serialNumber	Obtain the OCSP regarding the x509v3 certificate with the given serial number. The OCSP returned, contaning the status of the certificate (good, revoked or unknown), is signed by all the endorsers

Table 4.1: Proxy REST API Endpoints

5.3 Verifies the endorser signature regarding the certificate. We assume that each Proxy obtains the endorsers DPKI certificates before the Blockchain-Enabled DPKI system is initialized. If threshold signatures are being used, we use the functions added by [21], regarding an existing threshold signature library[7], in the SDK extension. When the Proxy server is starting, it queries the DPKI Chaincode to obtain the signature type and algorithm being used, storing them for future validations and signatures. It then adds that signature in a list to be later stored in a certificate extension.

6. Finalizes the transactional proposal by sending the transaction with all the endorsements. This is what makes the transaction to be committed in the ledger, offering transparency in the DPKI operations.
7. Inserts the signature type that is used by the DPKI in the X509v3 certificate as an extension, with OID 2.5.29.89.
8. Inserts the signatures list, added in 5.3, in the X509v3 certificate as an extension, with OID 2.5.29.90.
9. Optionally, inserts the original X509v3 certificate, without the Blockchain-Enabled DPKI related extensions, in Base64 format, as an extension with OID 2.5.29.88. This way, the verification of the signatures generated by the endorsers is easier. It should be taken into consideration that this makes the certificate size increase considerably.
10. Re-generate the X509v3 certificate, now with the endorsers' signatures. The proxy also acts as a traditional CA by signing the certificate.
11. A new transaction proposal is sent in order to store the newly issued X509v3 certificate in the Blockchain ledger, where it can be re-verified before the storage process. This proposal follows the HLF traditional flow, verifying the transaction responses, sending the response with the endorsements, etc..
12. Send a response to the client, with his X509v3 certificate in Base64 PEM format, concluding the issuing of the certificate.

In the other models, either signing a CSR or a Self-Signed certificate from the client, the steps are similar. The only difference is that the Proxy does not issue a new X509v3 Certificate, only obtaining and inserting the signatures from the endorsers regarding the CSR or Self-Signed Certificate.

- **Revoke X509v3 Certificate**: The proxy receives the client's certificate to be revoked in Base64 PEM format, a nonce created by the client, and a signature, also created by the client. The content of the signature is PEM of the certificate to be revoked, together with the given nonce. This is used to verify that the client requesting the revocation is, in fact, the owner of the certificate. The process related to sending

[7] https://github.com/sweis/threshsig

the transaction and verifying the endorsers' signatures regarding the DPKI content is the same as the issuing process. The arguments for the transaction proposal are the certificate to be revoked, the client's nonce and signature, and the date of the revocation, which is created by the Proxy at the moment that the revocation proposal is built. If the revocation date was created by the Chaincode function, which is run by each endorser's Chaincode simulator, the output date would be different in each simulation. Each proposal response contains the new CRL containing the client's certificate, which is the same in each response, otherwise, the process is aborted since the Chaincode function simulations did not produce the same result. Each response also contains the associated endorser signature regarding the new CRL. The proxy then adds the endorsers CRL signatures in the new CRL, inserting them in the CRL extensions, with the OID mentioned before. A different Chaincode DPKI function is used by the proxy to store the new CRL, containing the endorsers' signatures. This function may also verify that the endorsers' signatures inserted by the proxy are in fact valid.

- **Get X509v3 Certificate**: the proxy receives the serial number of the X509v3 certificate to search and returns the certificate if it is present in the Blockchain's ledger. The Chaincode function creates a query to search for the certificate. The fact that the response from the function simulation in each endorser needs to be the same, in order to the transaction be valid, offers data consistency.

- **CRL Request**: this endpoint is used to obtain the Blockchain-Enabled DPKI's certificate revocation list. The CRL that is returned is stored in the Blockchain's ledger and contains each endorser's signature and the signature type, both stored in the CRL extensions. As in searching for a certificate, the function simulation in each endorser needs to be equal, offering data consistency.

- **OCSP Request**: this endpoint is used to obtain the current status of the certificate with the serial number received as input. The Proxy creates a transaction proposal for the Chaincode function responsible to obtain the status of the certificate. The OCSP status is generated by the Chaincode by querying the Blockchain's ledger. The different possible status follow RFC 5280 [29] (good, revoked, or unknown). It would be possible executing optional logic that may attribute a certain level of trust level to the certificate. This trust level mechanism was introduced in Subsection 3.4.2. The OCSP response is then returned to the client, which also contains the endorsers' signatures so it is possible to verify that the certificate status is correct and endorsed by the endorsers.

The REST API uses the interface shown in Table 4.2 to operate the requests and relay the requests as transaction proposals to the Blockchain.

4.2.2.2 Proxy SDK Client

The proxy SDK client enables the communication with the Blockchain's peers in order to send transaction proposals regarding DPKI requests and content. The base functions of the SDK that enable this communication are from the official HLF 1.1[8], together with extensions[9] from [21] to enable the use of threshold signatures in transactions. We also added changes[10], to enable the signing of DPKI content.

The interface of the DPKI-related functions that communicate with the Blockchain peers via transaction proposal is represented in Table 4.2.

SDK Function	Description
getChaincodeDefinition	Makes a query to the Blockchain's Chaincode to obtain all the information regarding the Chaincode and Blockchain-Enabled DPKI properties such as cryptographic algorithms and endorsers public keys. This way the Proxy can give relevant information to external clients.
requestSignaturesForClientCertificate	Creates a X509v3 Certificate from a client's CSR (proxy acts as CA) and sends the certificate in a transaction proposal in order for the endorsers to sign the certificate. Inserts the signatures in the extensions of the X509v3 Certificate. Receives the client CSR as a parameter and returns the BLockchain-Enabled DPKI approved X509v3 Certificate containing the signatures generated by the endorsers.
requestSignaturesForClientCSR	Takes the client's CSR and sends it in a transaction proposal in order for the endorsers to sign it. Receives the client CSR as a parameter and returns the received CSR, together with the signatures generated by the endorsers.

[8]HLF SDK 1.1: https://github.com/hyperledger/fabric-sdk-java/tree/release-1.1

[9]Extended HLF SDK 1.1 for threshold signatures: https://github.com/fmiguelgodinho/fabric-sdk-java

[10]Extended HLF SDK 1.1 for threshold signatures and signed DPKI content: https://github.com/miguelreisa/thesis-prototype-general/tree/master/fabric-extended-java-sdk

requestSignaturesForClientSelfSignedCert	Takes the self-signed certificate of the client and sends it in a transaction proposal in order for the endorsers to sign it. Receives the self-signed certificate from the client and returns it, together with the signatures generated by the endorsers.
getX509v3CertificateBySerialNumber	Makes a query to the Blockchain's ledger using a transaction proposal regarding a X509v3 Certificate, with the specified serial number. Receives the serial number to search as a parameter and returns the corresponding X509v3 Certificate if it is present in the ledger.
ocspRequest	Sends an OCSP request as a transaction proposal to the Blockchain's Chaincode regarding the serial number specified by the client. Receives the serial number as a parameter and returns the OCSP response with the status of the certificate and the signatures generated by the endorsers regarding the OCSP response. Optionally, it may return the trust level of the certificate if the Chaincode provides it.
revokeCertificate	Asks the Blockchain-Enabled DPKI to revoke the client's certificate via transaction proposal. The Chaincode function is responsible to verify if the client that sent the certificate to be revoked is, in fact, its owner (via signature). Receives the serial number or base64 PEM of the certificate to be revoked, together with a nonce and a signature involving the base64 PEM and nonce. Returns the newly generated CRL, now containing the revoked certificate.
crlRequest	Asks for the Blockchain-Enabled DPKI's Certificate Revocation List using a transaction proposal for the Chaincode.

Table 4.2: Proxy HLF SDK Client Interface

4.2.3 DPKI Chaincode

One of the most important components in the prototype is the main DPKI Chaincode. This Chaincode contains the DPKI-related functions that need to be run in order to issue, revoke and manage certificates, as to obtain CRLs and OCSP responses. Table 4.3 describes the functions present in the system's Chaincode. It is worth mentioning that each function, if multi-signatures are being used, begins by verifying if all endorsers already signed each other's public keys in order to start the Blockchain-Enabled DPKI services. This is explained with more detail in Section 4.2.5. All functions are used by the Proxy except the ones mentioned otherwise. An example of a Chaincode (simplified) is in Annex I. The complete Chaincode function is present in the central repository of the prototype[11]. The idea is to exist liberty in the definition of properties and functions and the possibility to have different Chaincodes to manage DPKI content differently. The Chaincode functions to issue and manage certificates must contain certain validations in order for the endorsers to sign and endorse certificates, OCSP responses, and revocation lists.

Chaincode Function	Description
Init	Called when the Chaincode is instantiated by the Blockchain network. Stores the properties given in the Chaincode instantiation request.
getChaincodeDefinition	Obtain the Chaincode properties such as the DPKI content signature type and algorithm, or the endpoints of each endorser.
getEndorsementMethod	This function is called by the system ESCC to identify the signing method for transaction proposals.
getDPKISignatureMethod	This function is called by the system ESCC to identify the signing method for DPKI content.
checkEndorsersPubKeySignatures	Check if all endorsers have their public keys signed by all other endorsers. This is used to know if the endorsers accept and trust each other in order to start the Blockchain-Enabled DPKI services.

[11]Central repository of the Blockchain-Enabled DPKI prototype: https://github.com/miguelreisa/thesis-prototype-general

signEndorserPubKey	Used by the endorsers to sign the public key of the specified endorser. It is only used in multi-signatures since in threshold signatures there is only one public key shared by all endorsers, and therefore it is assumed that all endorsers trust that key.
signX509v3Certificate	Used by the Proxy to gather signatures from the endorsers regarding the newly issued X509v3 certificate. This function should execute validations before returning the X509v3 certificate so the endorsers may sign it.
signClientSelfSignedX509v3Certificate	Used to gather signatures from the endorsers of a client's self-signed certificate. Like the previous function, this should also execute validations.
signClientCSR	Used to gather signatures from the endorsers of a client's CSR. As the previous function, this should also execute validations.
ocspRequest	Obtains the certificate with the given serial number from the ledger, verifying it against the currently active CRL, also stored in the ledger. Besides returning the OCSP of the certificate, it may also return the trust level of the certificate, defined by a algorithm used in the function.
getCRL	Return the CRL stored in the Blockchain's ledger.
revokeX509v3Certificate	Receives the client's certificate in Base64 or only the serial number of the certificate to revoke, together with a signature from the client to ensure that he is the owner of the certificate. Generates and returns a new CRL containing the newly revoked certificate.
getX509v3CertificateBySerialNumber	Search for a certificate with the given serial number in the Blockchain's ledger.

storeX509v3Certificate	Store a newly issued X509v3 certificate, validated and signed by the Chaincode and endorsers, in the Blockchain's ledger for transparency, integrity, and the possibility to obtain the certificate in the future.
storeSignedCSR	Store a CSR validated by the Chaincode and signed by the endorsers for transparency, integrity, and the possibility to obtain the CSR in the future.
storeSelfSignedCert	Store a Self-Signed certificate validated by the Chaincode and signed by the endorsers for transparency, integrity, and the possibility to obtain the certificate in the future.

Table 4.3: Excerpt of Chaincode Functions

4.2.4 DPKI Signatures

The prototype supports choice between multi-signatures and threshold signatures. The creation of these signatures is executed in the same way and environment as the HLF transaction-related signatures. Figure 4.1 shows how this process works. When the component responsible for the signatures, either for transactions or DPKI content, recognizes that a Chaincode function output contains DPKI content, it generates the signature using the signature type specified in the Chaincode and inserts it in the response besides the transaction's body and signature.

We decided to separate the signing of DPKI content and transactions in terms of the signature method and key pairs of each endorser. This allows greater control, although it would be possible to use the same keys and signature method for the signing of transaction proposals and DPKI content. For example, for some reason, it would be possible to use multi-signatures in the signing of transaction proposals, and threshold signatures in the signing of DPKI content.

4.2.5 Bootstrap Signatures

Having all peers hold sets of multi and threshold signature keys at the beginning of the Blockchain network is not correct in terms of security if we blindly trust the signatures and the associated keys in DPKI-related transactions. To address this, we assume that

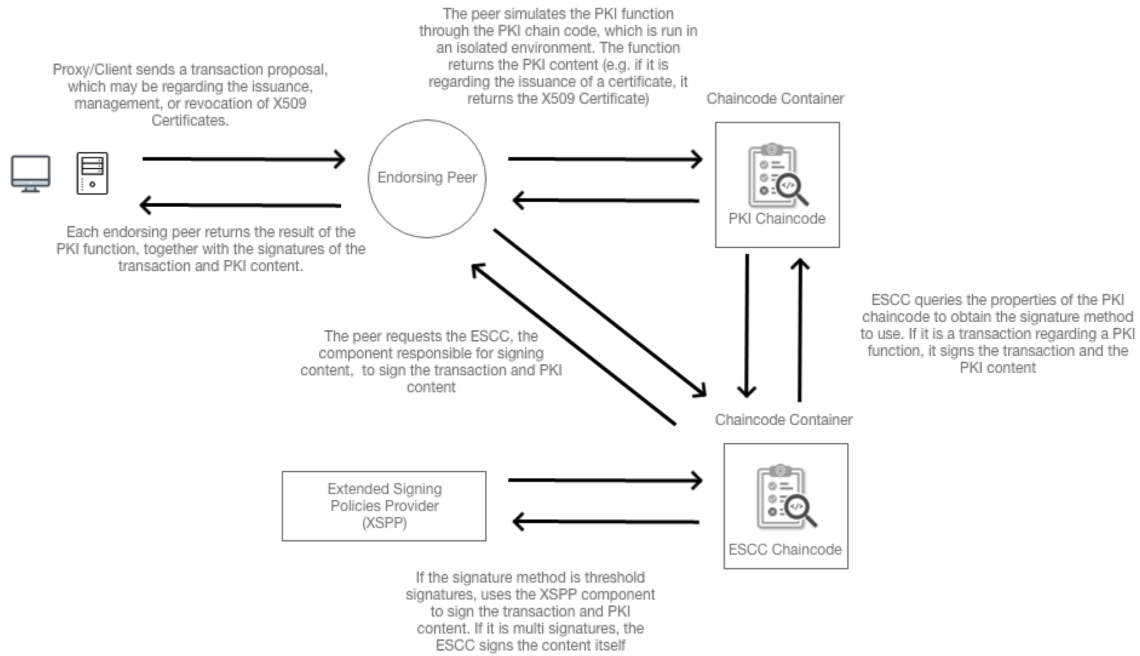


Figure 4.1: DPKI Signature Flow

endorsing peers have assigned keys before starting up the Blockchain network but, for the clients to be able to request DPKI operations, those keys must be trusted by all other endorsing peers. To achieve this, inspired by the PGP model, endorsing peers trust a selected public key from a fellow endorsing peer through a DPKI Chaincode function, which may contain extended verifications if desired. DPKI-related functions in the Chaincode can only be called if all the public keys are trusted by all endorsing peers, starting the DPKI services. We assume that the organizations that control the endorsing peers discussed the DPKI rules and conditions beforehand, but those signatures are the validation of the acceptance from all organizations.

4.2.6 Internal Clients

In the development and explanation of the prototype, our focus was the external clients, since we view them as the main clients of the Blockchain-Enabled DPKI. Although, it would be possible to have internal clients. These clients may be operating in different Blockchain channels (e.g. supply chain channel or a hospital-related channel) and want to use certificates issued by the Blockchain-Enabled DPKI. There are two ways of doing this: to have the same process flow as external clients, and contact a Proxy, or, since they have direct access to the Blockchain-Enabled DPKI peers, act as their own Proxy and send transaction proposals themselves in order to gather signatures from the endorsers regarding generated self-signed certificates, using the Self-Signed Certificate model presented in Chapter 3. To invoke a DPKI Chaincode function from another channel, that channel must have a Chaincode with a function that calls the DPKI Chaincode function. This way,

peers from various channels can use the Blockchain-Enabled DPKI without problems.

4.2.7 X509v3 Certificates and CRL Extensions

To store the Blockchain-Enabled DPKI signatures from the endorsers, we make use of the extension fields in both X509v3 certificates and CRLs. An extension with OID 2.5.29.89 was created to store the signature method used. To store the signatures an extension with OID 2.5.29.90 was created. The signatures generated by the endorsers are stored in Base64 strings. The OID 2.5.29 is the identifier for Version 3 certificate extensions. The numbers 89 and 90 were used since there are no extensions with those values. We added an optional extension with OID 2.5.29.88 that can be used to insert the original issued X509v3 certificate without the signatures. This is useful for easy verification of each endorser signature, but this increases significantly the size of the certificate. Without this extension, if we intend to verify the signatures, we must remove the extensions regarding the signature type and the signatures in order to successfully verify them against the originally issued certificate. Figure 4.2 shows an example of an issued X509v3 certificate with all the mentioned extensions.


```
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number: 1 (0x1)
  Signature Algorithm: sha512WithRSAEncryption
    Issuer: emailAddress = fakemail@mail.com, CN = Miguel Thesis, OU = DI, O = FCT UNL, L = Almada, ST = Setubal, C = PT
    Validity
      Not Before: May  1 21:47:16 2019 GMT
      Not After : Dec  1 22:47:16 2020 GMT
    Subject: C = PT, ST = Setubal, L = Caparica, O = FCT UNL, OU = DI, CN = Miguel Reis, emailAddress = mp.reis@campus.fct.unl.pt
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
        Public-Key: (2048 bit)
        Modulus:
          00:be:34:9f:50:04:e5:24:c2:a3:67:63:b9:a1:24:
          88:7e:41:....
        Exponent: 65537 (0x10001)
    X509v3 extensions:
      2.5.29.89:
        multisig
      2.5.29.90:
        .XapSkUBa7JTGXPINYDUi8kWJ4k+UdXCSnbf/+/TVbHj3IJHG1ktw2HxDUyvknC7fPDjV0tAnLlbwcpEm5IxVIdfQBP8jaicrTVoTZ5MCBwT2TK/
         9BPcUjhLkNj9ai6Uuy9nhlJ7XOKpTY+k76S7z7WiW/kDhpCNvZWgnOzA+cCyflcljK1eDJXx5rm1xZXT32KAbh/9maGUTrtxsnSVGBw/
         08K2czruESOFjDsAZSz+lYa6wdQ4MTEQ0gEVA2Z07z1vAEQZ1vW0Y5XIkgPaIkow5UoasEJdtPfYCCtUO7Y/
         ....................
         oSslt70VHIsOOpLGQCjg4pOmNyw3eRciFybqXNaR2oFj8KPatGl/D/EXaS3WWdJpYzzYqATovPuud2NU8gfxZaQD8MysJOJp0AjyA90A==
      2.5.29.88:
        -----BEGIN CERTIFICATE-----
        MIIEITCCAn2gAwIBAgIBATANBgkqhkiG9w0BAQ0FADCBiTEgMB4GCSqGSIb3DQEJ
        ARYRZmFrZW1haWxAbWFpbC5jb20xFjAUBgNVBAMMDU1pZ3VlbCBUaGVzaXMxCzAJ
        ..............................................................
        GoIxoMcTMc9vBBGWwhSRp4DmEYYKq0MajvWNaM0t9KR41e2X0cO1dP20BxsO9p/E
        mNebTmsvO0C8hclTHt/Nr25Uzr9TUGi2HIrqheyPA6mrG42UnhbyX/G9UExizdAZ
        mN8OUk9EQI2kvE2EhLddgaWqp8uyMBoDLQ==
        -----END CERTIFICATE-----

  Signature Algorithm: sha512WithRSAEncryption
     7b:77:39:b8:8f:9f:12:67:59:12:fe:58:52:89:25:4a:17:98:
     f7:7b:b5:....
```


Figure 4.2: A certificate issued by the Blockchain-Enabled DPKI

4.3 Summary

In this chapter, we presented the prototype developed by the system model from the previous chapter. We started by presenting a prototype overview and the technologies used in its implementation. We further detailed each component, mainly the DPKI signatures incorporated in the Blockchain, the DPKI Proxy, and the Chaincode. An overview

of the DPKI Proxy API exposed to the external clients was made, as well as the functions in the DPKI Chaincode that enable the management of X509v3 certificates in the Blockchain-Enabled DPKI. An explanation regarding the storage of signatures inside X509v3 certificates was also made, which enables the insertion of multi-signatures or threshold signatures in certificates without damaging the X509v3 standards.

Chapter 5

Experimental Evaluation and Analysis

In this chapter, we will present a set of experimental evaluations conducted with the developed prototype presented in Chapter 4. Succinctly, we intend to answer the following questions:

- Can we build a decentralized Blockchain-Enabled DPKI with a latency that does not prevent the proper use of its services by external clients?
- How do different cryptographic operations and key sizes affect the operations of the system?
- Does the latency per issuing of X509v3 certificates enable a viable system?
- Do functions such as certificate revocation, Certificate Revocation List (CRL) issuing, and Online Certificate Status Protocol (OCSP) requests have decent latency values that do not discourage external clients from following good practices and calling those services when needed?

For this purpose, we will first introduce the benchmark environment used for the conducted tests (Section 5.1), as well as the characteristics that all benchmarks followed. Then, we discuss in the following sections each experimental evaluation, explaining the intended observations, the conducted experiments, and the results observed. The DPKI functions that were executed in the experimental evaluations were X509v3 certificate issuing (Section 5.2), revocation of certificates and CRL Issuing (Section 5.3), and OCSP requests (Section 5.4), as these are the main functions of the Blockchain-Enabled DPKI developed. Finally, we conclude the chapter with the generic conclusions from all conducted evaluations (Section 5.5).

5.1 Evaluation Environment

A dedicated server was used to run the Blockchain Network and Proxy. The specifications of this server are shown in Table 5.1. Each peer of the Blockchain network is a docker container and has another dedicated container for its local ledger, therefore, for each peer, there are two docker containers. A single Proxy was used in the evaluations.

The external client used to call the Blockchain-Enabled DPKI functions is an average personal computer and has an RTT of 43ms with the dedicated server. The requests were made through the Proxy REST API with HTTP.

In all evaluations, four orderers were used in the BFT ordering service of the Blockchain network.

Regarding threshold signatures, the implementation used, developed in [21], is based on the optimistic version of the threshold signature verification algorithm [61], where the algorithm was able to verify the signatures on the first generated combination of k shares. Another aspect relevant in the usage of threshold signatures is the number of shares necessary to validate a signature. It is possible to only need for example two shares of four possible shares to validate a signature. In the evaluations, we required the total number of shares to validate signatures.

When multi-signatures were used, they were applied in the signing of DPKI related content, as well as in the signatures of transactions. The same applies to threshold signatures.

	Dedicated cloud server
CPU	AMD Epyc 7351P - 16 c / 32 t - 2.4 GHz / 2.9 GHz
RAM	128GB DDR4 ECC 2400MHz
OS	Ubuntu Server 18.04 Bionic Beaver LTS
Model	OVH Advance-4

Table 5.1: Testbench Environment

Each result in the evaluations consists of an average of twenty executions. Regarding endorsement policies (e.g. peers that need to sign transactions and DPKI content such as certificates to be issued), when we mention for example, six endorsers, this means that the endorsement policy requires that six peers, acting as endorsers, need to sign the DPKI content, either being certificates, OCSP responses, or newly issued CRLs.

5.2 X509v3 Certificate Issuing for External Clients

The most important function in our Blockchain-Enabled DPKI is the certificate issuing function, which is why most of the evaluations were made through this function. It is the process with more steps regarding signatures and their verifications, and certificate manipulation. All requests in these benchmarks follow the first model, represented in Figure 3.2, where the external client sends a Certificate Signing Request (CSR) to the Proxy in order to obtain an issued X509v3 certificate signed by all endorsers.

5.2.1 Cryptographic operations and their impact on the system

Signature	Issue Cert.	Endorsers sign.	Verify sigs	Store Cert.	Total
Multi-sig RSA 2048 bits	1345ms	0.002156ms	2ms	1491ms	3298ms
Multi-sig RSA 4096 bits	1187ms	0.002324ms	3,15ms	2135ms	3377ms
Multi-sig RSA/PSS 2048 bits	1075ms	0.002348ms	2,4ms	1493ms	3366ms
Multi-sig RSA/PSS 4096 bits	1175ms	0.002289ms	3,4ms	1489ms	3360ms
Multi-sig ECDSA 256 bits	1175ms	0.002343ms	4,7ms	1490ms	3452ms
RSA Threshold 2048 bits	1290ms	0.002766ms	6,5ms	1360ms	4438ms
RSA Threshold 4096 bits	4093ms	0.003020ms	12,25ms	3716ms	8478ms

Table 5.2: Analysis of different Cryptographic Algorithms and Key Sizes

In order to evaluate the performance of different cryptographic operations in the issuing of certificates, several benchmarks were run with different algorithms and key sizes. Table 5.2 shows the results of those benchmarks. For this evaluation, four peers, all endorsers, were used.

The second column refers to the transaction where the Proxy, after receiving the CSR from the client and generating a new X509v3 certificate, sends the certificate to the Blockchain network to be validated and gather signatures from all endorsers. The certificate is also being stored in the local ledger of each peer after the ordering service finishes ordering and dispersing the transaction blocks. It includes the verification of the CSR by the Proxy, the validations by the chaincode, and other minor steps needed in the issuing process. That is why it is understandable that this step represents a rather high percentage of the total time. The same happens in the fifth column, wherein this case the transaction is related to the storage of the newly issued X509v3 certificate.

We can see that the time needed for the endorsers to sign the DPKI content, in this case, the newly issued certificate, is small when compared to the total time. This shows that the use of different cryptographic algorithms when using multi-signatures does not affect considerably the time needed to issue an X509v3 certificate. The fourth column also confirms this, despite the verification of signatures taking much longer, when compared to the creation of the signatures, mainly due to the fact that the Proxy reads the public keys from files, an aspect that could be improved.

Multi-signatures with RSA have a similar time in the signing and verification when compared with RSA/PSS, where RSA/PSS has a slightly higher value due to the mask generation process. When comparing to ECDSA, as it is well known in cryptography, the verification step in ECDSA signatures is slower than RSA or RSA/PSS.

Although the times representing signature verification and generation only refer to the signing of DPKI content and not transactions, we believe that when signing transactions with threshold signatures, since the content is considerably bigger and there are several steps where there are verifications (e.g. when normal peers store the state in their local ledger, they validate the transaction and therefore the signatures), it escalates the total

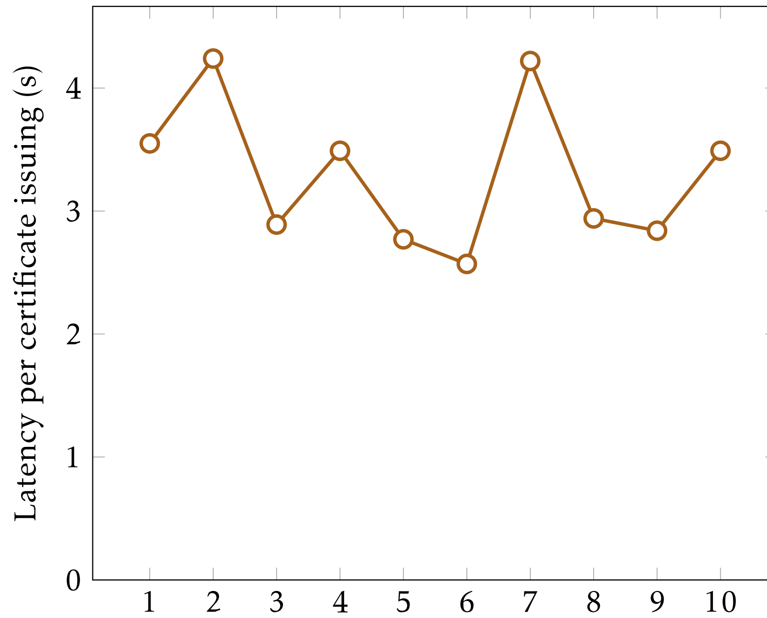


Figure 5.1: Variation in the latency in issuing X509v3 extended certificates.

time considerably. For that reason, we can see a difference of one second between multi-signatures with 2048 bits of key size and threshold signatures with 2048 bits of key size.

We can conclude that the latency suffers the most when we use threshold signatures, especially when we increase the key size. Threshold signatures increase latency due to the signature reconstruction process through the signature shares in the verification algorithm, which is more complex than the RSA, RSA/PSS or ECDSA processes.

5.2.2 Latency variation in the issuing process

One interesting aspect in all requests to the Blockchain-Enabled DPKI is the variation in the latency of request responses. Figure 5.1 represents an example of this variation. We can see a significant variation in the latency between different executions, where the minimum and maximum values differ in almost two seconds. We believe this is due to the validation and committing of transaction blocks in the Hyperledger Fabric (HLF) transaction flow, mainly in the ordering process by the ordering service. The ordering process used is BFT, which decreases the performance of the system, since it has to wait for a quorum of $3f + 1$ responses for each consensus round, and it is normal if different executions require a different number of messages, varying the final latency.

5.2.3 Impact in the size of issued X509v3 certificates

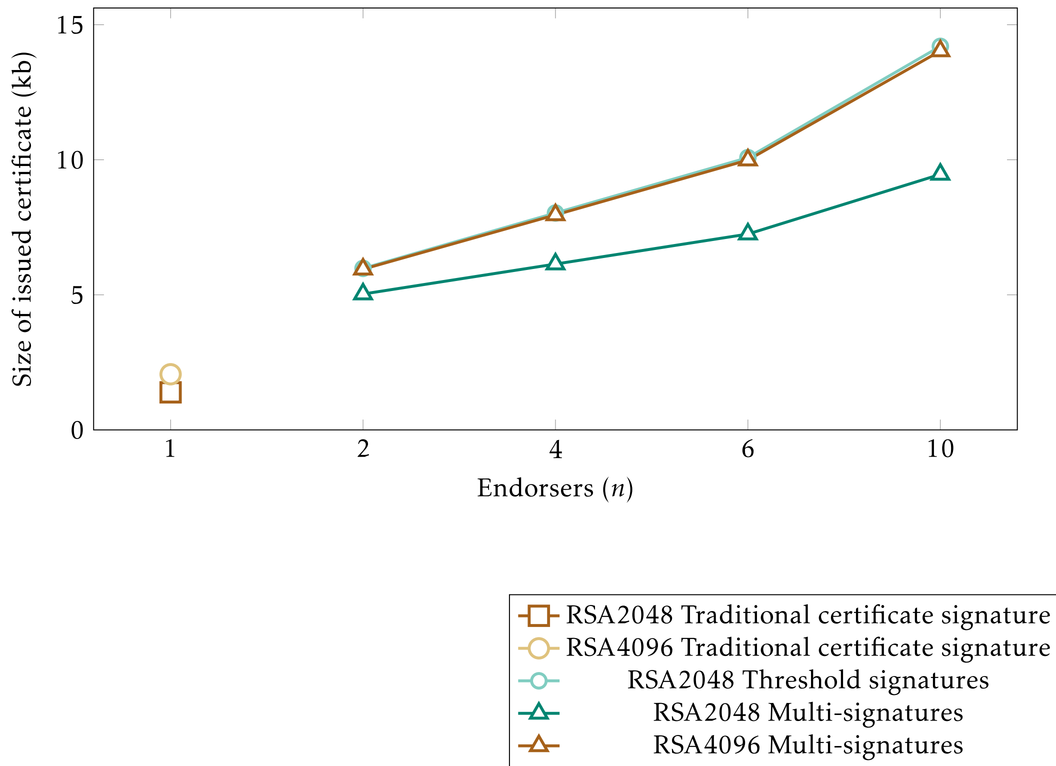


Figure 5.2: Number of Endorsers and size of issued certificates.

Figure 5.2 shows the relation between the size in KB of traditional certificates and Blockchain-Enabled DPKI certificates with multi-signatures or threshold signatures. Since we are working with multi-signatures and threshold signatures, issued X509v3 certificates contain all signatures in extensions. Therefore their size will increase considerably, mainly due to the signatures or signature shares (threshold signatures) and extra information. Besides the signatures being stored in the certificate, the Peer ID of the endorser that generated a specific signature is also stored in a <key, value> type of storage. Another important reason why the certificates have a bigger size is that we also stored the original X509v3 certificate generated by the Proxy before the addition of the signatures created by the endorsers. This facilitates the validation of the certificate and its signatures by other entities. The size increases considerably and it is optional.

5.2.4 Impact of the number of endorsers

As we mentioned in Section 5.2.1, threshold signatures add one second in the latency of certificate issuing when compared to the usage of multi-signatures, which we also can see in Figure 5.3. We can also verify that increasing the number of endorsers does not increase the latency as it would be expected. This shows that increasing the number of endorsers, which means increasing the number of signature generations and validations, and the

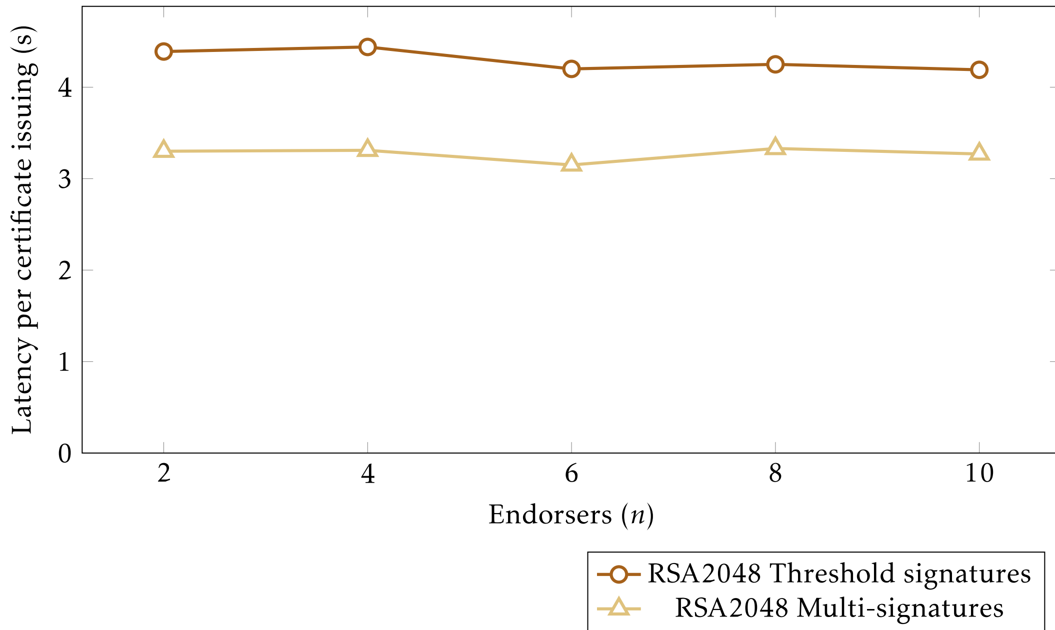


Figure 5.3: Latency in issuing X509v3 extended certificates with a variable number of Endorsers.

communication with the endorsers, does not affect the total latency like the ordering service and threshold signatures do.

5.2.5 Impact of the number of normal peers

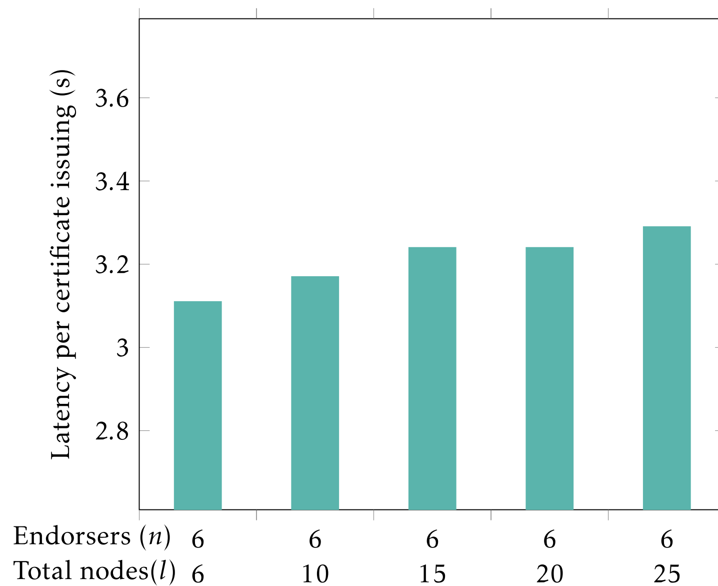


Figure 5.4: Latency in issuing X509v3 extended certificates with a variable number of Regular Peers.

In the previous evaluations, all the peers in the Blockchain network were endorsers, all signing DPKI content. To analyze how the latency changes with more peers, we executed tests where the number of endorsers was static while increasing the number of total peers, where non-endorsing peers just validate the final transaction blocks and store them in their local ledgers.

Figure 5.4 shows the results obtained with different numbers of normal peers while using multi-signatures RSA with a key size of 4096 bits. We can see that the number of normal peers does not increase considerably the total latency of the requests. This proves that most of the latency is due to the transaction and DPKI-content signatures, ordering service, and the endorsement steps in transactions, which are not affected by normal peers, that just receive the final transaction blocks at the end of each transaction.

5.2.6 Comparison with the Issuing Process in a Conventional PKI Solution

	Nº Signers/Endorsers	Latency (ms)
EJBCA PKI	1	348
Blockchain-Enabled DPKI Multi-Signatures	1	2756
Blockchain-Enabled DPKI Multi-Signatures	4	3298
Blockchain-Enabled DPKI Threshold-Signatures	4	4438

Table 5.3: Comparison with the issuing process in a conventional PKI

In order to have a comparison with the traditional issuing of X509 certificates, we run, on the same machine where the Blockchain network and Proxy were running, a docker container with an EJBCA PKI [15]. We used simple configurations, enough to enable requests regarding certificate issuing given a CSR. It is worth mentioning that we did not take in consideration the time to enroll an entity, which happens in real PKIs, since our solution also does not take that into consideration. We just analyzed the automated process, where an already registered entity sends a CSR in order to receive a newly issued X509 certificate.

Table 5.3 shows the obtained results, comparing the latency when issuing a certificate in the traditional EJBCA PKI and our Blockchain-Enabled DPKI. Even with only one endorser and, therefore, one signature, the issuing of certificates in the Blockchain-Enabled DPKI has more than two seconds of latency than the EJBCA PKI. These results were expected before developing the prototype, since the characteristics and improvements that our model provides have a tradeoff of performance due to the previously mentioned steps, such as the ordering process, communication steps between peers in transactions, and others.

5.3 Revocation of certificates and CRL Issuing

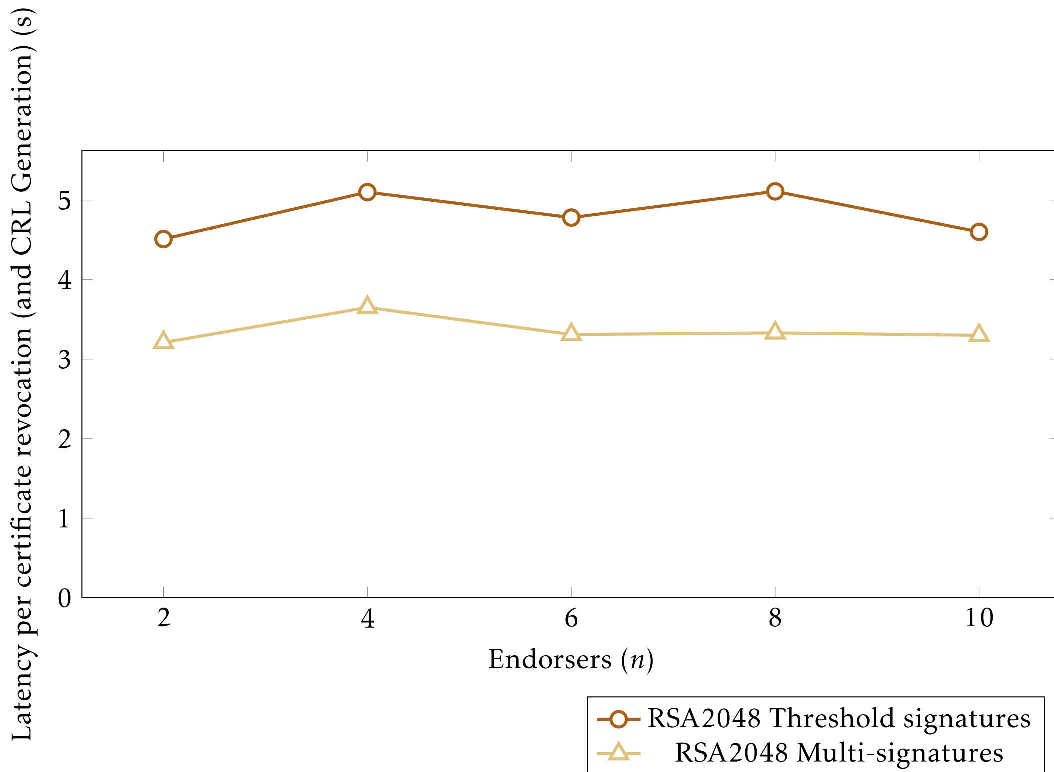


Figure 5.5: Performance of Threshold Signatures and Multi-Signatures in the revocation of certificates.

Certificate revocation and CRL generation is another important component of a PKI. Our Blockchain-Enabled DPKI implements a simple form of revocation of certificates and CRL Issuing. The process was previously explained in Chapter 4 and has a similar number of steps compared to the certificate issuing function. Figure 5.5 shows results with multi-signatures and threshold signatures, and different numbers of endorsers. The latency is similar to the latency in certificate issuing, shown in Figure 5.3, mainly when using RSA multi-signatures with a key size of 2048 bits. One aspect that makes this function have a bigger latency, is that the Proxy is running the REST API and logic in Java language, and when it obtains the new CRL and endorsers signatures, in order to insert them in the newly issued CRL, uses a Golang program to do so, this is because Golang enables a greater manipulation of already issued X509v3 certificates, CRLs, and other X509 content. Regarding the difference in latency between multi-signatures and threshold signatures, it is rather the same that the one shown in Figure 5.3, which was already commented in Section 5.2.4.

5.4 OCSP Requests

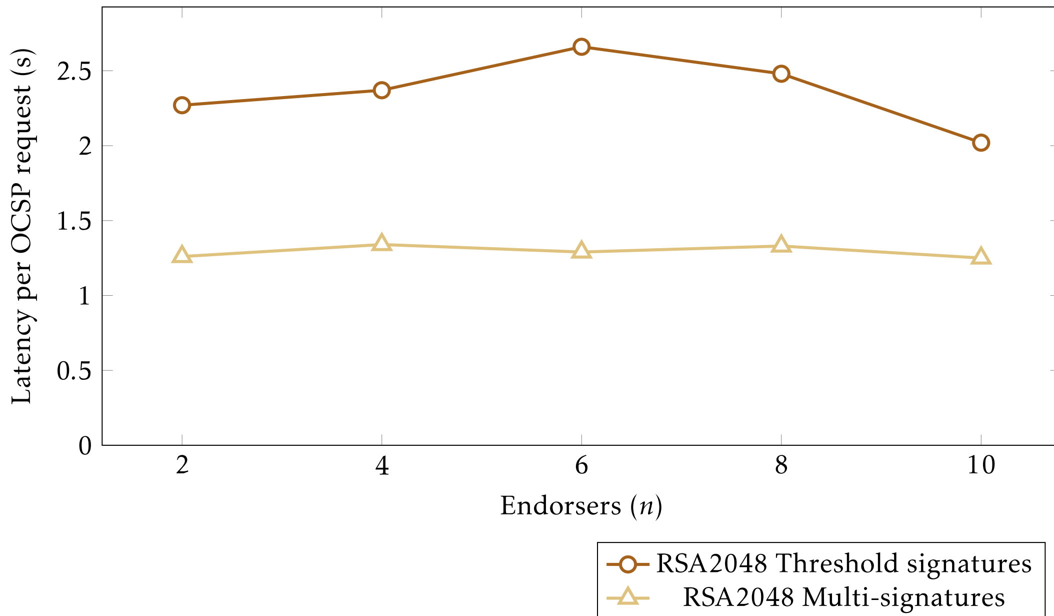


Figure 5.6: Performance of OCSP processing when using Threshold Signatures and Multi-Signatures.

Both of the previously evaluated functions in our Blockchain-Enabled DPKI write data to the ledger of the Blockchain. The system also provides OCSP requests, that do not write data into the ledger. Figure 5.6 shows the latency when calling OCSP requests. Since there are no write operations in the ledger, transactions are not submitted to the ordering service, decreasing the total latency considerably. The chaincode obtains and returns the current stored CRL so the endorsers can sign it and insert their signatures in the response of the OCSP. There is still a difference of approximately 1 second between multi-signatures and threshold signatures as it was seen in the other requests. With 10 endorsers and threshold signatures, we see a decrease in the latency, but we believe this is due to the variations mentioned previously in Section 5.2.2.

5.5 Summary

In this chapter, we described the evaluations executed over the presented and implemented prototype. We made requests through the most important functions of the prototype, through an external client, in order to obtain certificates, OCSP responses, and CRL issued by the Blockchain-Enabled DPKI.

We analyzed and commented on the following topics:

- Variations in the latency when switching between multi-signatures and threshold signatures, and cryptographic algorithms.

- Comparison against traditional X509v3 certificate issuing, when a single entity (CA) signs and issues the certificate.
- Variation in the latency of requests due to the Blockchain transaction flow.
- Impact in the size of issued X509v3 certificates due to the addition of multi-signatures or threshold signatures through extensions.
- Impact of the number of endorsers in the latency of certificate issuing and revocation, CRL issuing, and OCSP responses.
- Impact of the number of normal peers that just validate and store issued certificates and CRLs.
- General latency in CRL issuing and OCSP responses.

We can confirm that the time to issue certificates is mainly affected by the HLF transaction flow, in the communication required between peers and the ordering service BFT algorithm. Besides, when using threshold signatures the time has a greater increase when compared with multi-signatures due to the complexity in the threshold signatures algorithm. The latency observed in the executed evaluations was expected since we are working with a Blockchain network, which contains complex processes, a BFT ordering service, and multi-signatures or threshold signatures instead of the traditional one signature. We believe that the drawback of the observed latency does not overlap the benefits that this system offers. The same applies to the increased latency when using threshold signatures. We believe their benefits regarding reduced storage size and smaller transaction content overlap the latency added.

We may conclude that the developed prototype offers the mentioned improvements over the traditional PKIs while still having an acceptable latency.

Chapter 6

Conclusion and Final Remarks

6.1 Conclusion

The dissertation addressed the design, implementation, and evaluation of a Blockchain-Enabled DPKI (Decentralized PKI) solution, based on a permissioned-oriented collaborative consortium model. The proposal and its design options are leveraged by a set of service planes and their internal components, supported in the Hyperledger Fabric (HLF) Blockchain platform. In our proposed solution for the addressed PKI framework model, X509v3 certificates are issued and managed following security invariants and processing rules, with trust and consistency metrics, defined and consistently executed by Blockchain peers, under the scrutiny of smart contracts. In this way, X509v3 certificates are issued cooperatively, using multi-signatures or group-oriented threshold-based Byzantine fault-tolerant (BFT) signatures. The smart contracts executed by the DPKI peers dictate their participation in issuing, signing, attestation, validation and revocation processes. Any peer can validate certificates verifying its consistent state, stabilized in closed blocks in a Meckle-tree structure, maintained consistently with integrity guarantees in the Blockchain. All state-transition operations on managed certificates are also executed with ordering guarantees, provided by BFT consensus and communication primitives.

Externally, the proposed DPKI solution can be used by applications as a PKI service. It offers the conventional functions and operations found in the PKIX framework standard model for management of X509 certificates. In this we include the provided functions as we find in the PKIX architectural model, such as access and download of certificates' revocation status information, certificates and CRLs publication and retrieval, support for certificate signing requests (CSRs) and possibility for managing and obtaining the issued certificates in the standardized format Base 64 PEM encoding. However, our

solution also provides extensions in the X509v3 representation model, to support issuing with multi-signed public-key signatures, as well as, cooperative byzantine-fault-tolerant threshold digital signatures for trust-enforcement. Those extensions can be regulated by the expressiveness support of the provided smart-contracts.

A particular case for the use of our DPKI solution as a service is the provision of their functions for other Blockchain-enabled applications. In our approach and implementation, this is naturally supported for applications executing in different channels of the HLF Blockchain platform, running the DPKI in a specific service channel.

We have implemented the proposed solution by producing an available prototype and that can be used for research and experimental studies by interested researchers and developers. The prototype implements the design assumptions and all the components discussed in the dissertation, being ready to run as a Cloud-Blockchain enabled environment. The available prototype can also be seen as a base platform for future developments in providing full-fledged PKI management functions for a cooperative and decentralized certification authority model, following our design considerations.

We also conducted an extensive experimental evaluation over the developed platform. Our observations show that careful selection and the implementation of the components in the proposed solution is able to support the validation of our initial hypothesis and concerns. Moreover, the specific experimental evaluation and analysis on the performance of certificates' issuing, the cost of provided cryptographic services and the overall operation, reveals interesting results, comparing with conventional operations supported by centralized PKIs, showing the validity of the solution.

6.2 Future Work

We believe that we addressed the initially expected objective and related contributions to our dissertation. In the current implementation, some optimizations can be done to enhance the current prototype, considering some open-issues and future research work trends.

As open-issues we emphasize the following concerns:

- Deployment of the current implementation in a cloud-of-clouds environment, as a new testbench where the Blockchain peers can run in different virtual or dedicated machines from different service providers;
- Repetition of the initially conducted experimental evaluation and criteria, in this new testbench environment;
- Conduction of more experimental evaluations under more scalability requirements and assessment criteria, following the initial evaluation rational already addressed in the dissertation.

As future-work directions, we summarize the following ideas:

- Regarding threshold signatures, currently, there is no way of generating new signature shares when a new endorser joins the network. This an aspect that could be improved in order to have a secure way to re-generate signature shares.
- Another concern related to the generation of threshold signature shares is the fact that there is one dealer that generates and returns all the shares. This is a centralized approach that could be improved by having a procedure where several entities generate different shares in a byzantine fault tolerance fashion.
- Transaction signatures and DPKI content signatures use different key pairs and certificates. The certificates used in transactions are still generated by a utility of Hyperledger Fabric. A possible improvement is the possibility to use the same generated key pairs and certificates for both DPKI signatures and transaction signatures.
- Smart contracts are another aspect of the implementation that may be improved in the future, mainly regarding necessary reconfigurations throughout the system lifetime and improving expressiveness conditions for certificate issuing. The determination and usage of certificate trust levels by smart contracts is another aspect that should be explored in future work. As well as the generation of Blockchain Public-Key Certificate States (BCS) by peers inside the Blockchain network.
- The application scenario related to internal clients was not fully explored due to time restrictions. A direction for future work would be exploring the participation of internal clients and benchmarking when different channels, with different purposes, use the implemented Blockchain-Enabled DPKI.

Bibliography

[1] M. Al-Bassam. "SCPKI: A Smart Contract-based PKI and Identity System." In: *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts* (2017), pp. 35–40. doi: https://dl.acm.org/citation.cfm?doid=3055518.3055530.

[2] C. Allen, A. Brock, V. Buterin, J. Callas, D. Dorje, C. Lundkvist, P. Kravchenko, J. Nelson, D. Reed, M. Sabadello, G. Slepak, N. Thorp, and H. T. Wood. "Decentralized Public Key Infrastructure, Web-of-Trust Info." In: *Rebooting Web of Trust Design Workshop, San Francisco USA* (2015). url: https://github.com/WebOfTrustInfo/rwot1-sf.

[3] *Amazon Blockchain Service*. url: https://aws.amazon.com/pt/partners/Blockchain/.

[4] L. Axon and M. Goldsmith. "PB-PKI: A Privacy-aware Blockchain-based PKI." In: (2017). doi: https://www.researchgate.net/publication/318870515_PB-PKI_A_Privacy-aware_Blockchain-based_PKI.

[5] J. Benet. "IPFS - Content Addressed, Versioned, P2P File System." In: (). url: https://ipfs.io/ipfs/QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX/ipfs.draft3.pdf.

[6] A. Bessani, J. Sousa, and E. Alchieri. "State machine replication for the masses with BFT-SMART." In: *Proceedings of the International Conference on Dependable Systems and Networks* (June 2014), pp. 355–362. doi: 10.1109/DSN.2014.43.

[7] A. Bessani, J. Sousa, and M. Vukolić. "A Byzantine Fault-tolerant Ordering Service for the Hyperledger Fabric Blockchain Platform." In: *Proceedings of the 1st Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers*. SERIAL '17. ACM, 2017, 6:1–6:2. url: http://doi.acm.org/10.1145/3152824.3152830.

[8] E. Buchman. "Tendermint: Byzantine Fault Tolerance in the Age of Blockchains." In: (2016). url: https://allquantor.at/Blockchainbib/pdf/buchman2016tendermint.pdf.

[9] M. Castro and B. Liskov. "Practical Byzantine Fault Tolerance." In: *Proceedings of the Third Symposium on Operating Systems Design and Implementation* (1999). url: http://pmg.csail.mit.edu/papers/osdi99.pdf.

[10] *CoinMarketCap - All Cryptocurrencies*. URL: https://coinmarketcap.com/all/views/all/.

[11] *Consensus Algorithms: The Root Of The Blockchain Technology*. URL: https://101Blockchains.com/consensus-algorithms-Blockchain/.

[12] *Counterparty*. URL: https://counterparty.io.

[13] O. Dib, K.-L. Brousmiche, A. Durand, E. Thea, and E. B. Hamida. “Consortium Blockchains: Overview, Applications and Challenges.” In: *International Journal on Advances in Telecommunications* (2018). DOI: https://www.researchgate.net/publication/328887130_Consortium_Blockchains_Overview_Applications_and_Challenges.

[14] *DigiNotar Incident Report*. URL: https://www.onderzoeksraad.nl/nl/media/attachment/2018/7/10/rapport_diginotar_en_summary.pdf.

[15] *EJBCA*. URL: https://www.ejbca.org/docs/EJBCA_6.15.1_Documentation.html.

[16] C. Ellison and B. Schneier. “Ten Risks of PKI: What You’re not Being Told about Public Key Infrastructure.” In: *Computer Security Journal Volume XVI, Number 1* (2000). URL: https://www.schneier.com/academic/paperfiles/paper-pki.pdf.

[17] *Ethereum Project*. URL: https://www.ethereum.org.

[18] *Ethereum Proof of Stake*. URL: https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQs.

[19] N. Fazio and A. Nicolosi. “Cryptographic Accumulators: Definitions, Constructions and Applications.” In: (2003). DOI: https://www.researchgate.net/publication/235923578_Cryptographic_Accumulators_Definitions_Constructions_and_Applications.

[20] C. Fromknecht and D. Velicanu. “CertCoin : A NameCoin Based Decentralized Authentication.” In: (2014). DOI: https://www.semanticscholar.org/paper/CertCoin-%3A-A-NameCoin-Based-Decentralized-System-6-Fromknecht-Velicanu/72889440eaeb7a1a17a8be830feff236b5a62b67.

[21] F. Godinho. “Bringing Order into Things. Decentralized and Scalable Ledgering for the Internet-of-Things.” In: (). URL: http://hdl.handle.net/10362/55172.

[22] G. Greenspan. “MultiChain Private Blockchain.” In: (). URL: https://www.multichain.com/download/MultiChain-White-Paper.pdf.

[23] *History of Risks Threat Events to CAs and PKI*. URL: http://wiki.cacert.org/Risk/History.

[24] *How Hackers Hijacked a Bank’s Entire Online Operation*. URL: https://www.wired.com/2017/04/hackers-hijacked-banks-entire-online-operation/.

[25] *HydraChain*. URL: https://github.com/HydraChain/hydrachain.

[26] *Hyperledger Fabric*. URL: https://www.hyperledger.org/projects/fabric.

[27] *Hyperledger Iroha*. URL: https://github.com/hyperledger/iroha.

[28] *IETF PKI Issues Draft*. URL: https://tools.ietf.org/html/draft-iab-web-pki-problems-01#section-3.2.1.

[29] *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. URL: https://tools.ietf.org/html/rfc5280.

[30] M. Y. Kubilay, M. S. Kiraz, and H. A. Mantar. "CertLedger: A New PKI Model with Certificate Transparency Based on Blockchain." In: (October, 2018). URL: https://arxiv.org/abs/1806.03914.

[31] C. Kuhlman, B. Bollen, S. Davis, and D. Middleton. "Hyperledger Burrow." In: (2017). URL: https://www.hyperledger.org/wp-content/uploads/2017/06/HIP_Burrowv2.pdf.

[32] B. Laurie, A. Langley, and E. Kasper. "Certificate Transparency." In: (2013). URL: http://www.rfc-editor.org/rfc/pdfrfc/rfc6962.txt.pdf.

[33] T. Lee, C. Pappas, P. Szalachowski, and A. Perrig. "Towards Sustainable Evolution for the TLS Public-Key Infrastructure." In: *Proceedings of the 2018 on Asia Conference on Computer and Communications Security* (2018), pp. 637–649. ISSN: 978-1-4503-5576-6. DOI: https://dl.acm.org/citation.cfm?id=3196520.

[34] K. Lewison and F. Corella. "PomCor: Backing Rich Credentials with a Blockchain PKI." In: (2016). DOI: https://pomcor.com/techreports/BlockchainPKI.pdf.

[35] *Livepeer: Open Source Video Infrastructure Services, Built On The Ethereum Blockchain*. URL: https://livepeer.org.

[36] *Maersk and IBM Introduce TradeLens Blockchain Shipping Solution*. URL: https://www.maersk.com/en/news/2018/06/29/maersk-and-ibm-introduce-tradelens-Blockchain-shipping-solution.

[37] P. Martins. *Introdução à Blockchain*. First. FCA, 2018. ISBN: 9789727228874.

[38] S. Matsumoto and R. M. Reischuk. "IKP: Turning a PKI Around with Decentralized Automated Incentives." In: *2017 IEEE Symposium on Security and Privacy* (2017). DOI: https://ieeexplore.ieee.org/document/7958590.

[39] P. Maymounkov and D. Mazières. "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric." In: *IPTPS '01 Revised Papers from the First International Workshop on Peer-to-Peer Systems* ().

[40] D. Mazières. "The Stellar Consensus Protocol: A Federated Model for Internet-level Consensus." In: (). URL: https://www.stellar.org/papers/stellar-consensus-protocol.pdf.

[41] *Medrec*. URL: https://medrec.media.mit.edu.

[42] S. Nakamoto. "Bitcoin: A Peer-to-Peer Electronic Cash System." In: (). URL: https://bitcoin.org/bitcoin.pdf.

[43] *Namecoin*. URL: https://www.namecoin.org.

[44] K. Olson, M. Bowman, J. Mitchell, S. Amundson, D. Middleton, and C. Montgomery. "Sawtooth: An Introduction." In: (2018). URL: https://www.hyperledger.org/wp-content/uploads/2018/01/Hyperledger_Sawtooth_WhitePaper.pdf.

[45] *On Public and Private Blockchains*. URL: https://blog.ethereum.org/2015/08/07/on-public-and-private-Blockchains/.

[46] *OpenCA Guide*. URL: https://www.openca.org/projects/openca/docs/openca-guide.pdf.

[47] *Openchain*. URL: https://www.openchain.org.

[48] *OpenPGP Message Format*. URL: https://tools.ietf.org/html/rfc4880.

[49] *OpenSSL*. URL: https://www.openssl.org.

[50] *OpenSSL PKI Documentation*. URL: https://pki-tutorial.readthedocs.io/en/latest/.

[51] *Provenance*. URL: https://www.provenance.org/whitepaper.

[52] *Public-Key Infrastructure (X.509) (PKIX)*. URL: https://datatracker.ietf.org/wg/pkix/about/.

[53] *Quorum*. URL: https://www.jpmorgan.com/global/Quorum.

[54] *Rebooting the Web of Trust Movement*. URL: https://www.weboftrust.info/.

[55] *RFC 4949 - Internet Security Glossary*. URL: https://tools.ietf.org/html/rfc4949.

[56] A. Shamir. "How to share a secret." In: *Communications of the ACM, Volume 22 Issue 11* (1979), pp. 612–613. DOI: https://dl.acm.org/citation.cfm?doid=359168.359176.

[57] *Smart Contracts on Blockchains*. URL: https://www.ibm.com/blogs/Blockchain/2018/07/what-are-smart-contracts-on-Blockchain/.

[58] *SMTP Extension for Internationalized Email*. URL: https://tools.ietf.org/html/rfc6531.

[59] W. Stallings. *Network Security Essentials - Applications and Standards*. Sixth. Pearson-Prentice Hall, 2016. ISBN: 0133370437.

[60] *Standard for the format of ARPA Internet Text Messages*. URL: https://tools.ietf.org/html/rfc822.

[61] C. Stathakopoulou and C. Cachin. "Threshold Signatures for Blockchain Systems." In: *RZ3910* (2017). DOI: https://domino.research.ibm.com/library/cyberdig.nsf/papers/CA80E201DE9C8A0A852580FA004D412F.

[62] P. Szalachowski, L. Chuat, T. Lee, and A. Perrig. “RITM: Revocation in the Middle.” In: (2016). URL: https://arxiv.org/abs/1604.08490.

[63] *Tezos*. URL: https://tezos.com.

[64] Z. Wang, J. Lin, Q. Cai, Q. Wang, J. Jing, and D. Zha. “Blockchain-based Certificate Transparency and Revocation Transparency.” In: (2018). DOI: https://fc18.ifca.ai/bitcoin/papers/bitcoin18-final29.pdf.

[65] *Web of Trust*. URL: https://www.mywot.com.

[66] A. Yakubov, W. Shbair, A. Wallbom, D. Sanda, and R. State. “A Blockchain-Based PKI Management Framework.” In: *IEEE/IFIP man2block 2018* (2018). DOI: https://www.researchgate.net/publication/323692746_A_Blockchain-Based_PKI_Management_Framework.

[67] P. Zimmerman. “Why I Wrote PGP.” In: *Original 1991 PGP User’s Guide (updated in 1999)* (1999). URL: https://www.philzimmermann.com/EN/essays/WhyIWrotePGP.html.

Annex I

Blockchain-Enabled DPKI Chaincode

Listing I.1: Annex: Excerpt of the PKI Chaincode Properties and Functions

```
// contract extended properties
type ExtendedContractProperties struct {
    ContractId          string          ...
    ContractVersion     int             ...
    Channel             string          ...
    AvailableFunctions  [][]string      ...
    InstalledOnNodes    []string        ...
    SignatureType       string          ...
    SigningNodes        []Node          ...
    ConsensusType       string          ...
    ConsensusNodes      []Node          ...
    ExpiresOn           string          ...
    ValidFrom           string          ...
    ProviderSignature   string          ...
    DeployedOn          string          ...
}

//More properties could be added
type PKIProperties struct {
    //Blockchain-Enabled DPKI Proxies Endpoints
    ProxiesEndpoints                map[string]string       ...
    //Certificates issued with this chaincode can belong to a CA
    CanBelongToCA                   bool                    ...
    //Max expiration time of issued certificates
    MaxExpirationTime               int64                   ...
    //X509v3 extensions that are accepted with this chaincode (e.g. Key usages)
    PossibleExtensions              []string                ...
    msTimeBetweenCRLUpdates         int64                   ...
    //Signature type used by endorsers to sign the issued certificates
```

```
30      //(multisignature or threshold signature)
31      PKISignatureType                    string                      ...
32      //Signature algorithm used by endorsers to sign PKI content
33      //(if using multisig)
34      MultisigSignatureAlgorithm          string                      ...
35      SigningNodesPKIMultiSigPublicKeys   map[string]string           ...
36      SigningNodesPubKeysSignatures       map[string]map[string]string     ...
37      //If all endorsers public keys are signed by all endorsers
38      //(so the PKI services may start)
39      //Inspired in PGP. Endorsers sign other endorser's public keys
40      //so these keys gain trust in the system. True if all endorsers
41      //public keys have been endorsed by all endorsers. PKI functions
42      //can be called if this is true.
43      //If using threshold sig this is true in the start
44      //We assume that all entities accept the agreed thresh pub key.
45      AllPublicKeysEndorsed               bool                        ...
46      //Threshold Public Key (if using threshold signatures)
47      SigningNodesPKIThreshSigPublicKey   string                      ...
48      //Serial number used for the next issued certificate.
49      //It is incremented at each process.
50      CurrentSerialNumber                 int64                       ...
51  }
52
53
54  /*
55   * This is called when the chaincode is instantiated by the Blockchain network
56   */
57  func (s *SmartContract) Init ... {
58
59      //Stores the properties
60
61      ...
62
63      return shim.Success(nil)
64  }
65
66
67  /*
68      To obtain the chaincode properties (e.g. extended and PKI-related)
69  */
70  func (s *SmartContract) getContractDefinition ... {
71      ...
72  }
73
74  ...
75
76  /**
77  * This function is to be called by system chaincode (escc)
78      to identify signing method for transaction proposals.
79   */
```

```
80 func (s *SmartContract) getEndorsementMethod ... {
81
82     // gen composite key (EXTENDED_CONTRACT_PROPERTIES)
83     ...
84
85     // return endorsement method
86     endorsementMethod := []byte(extProps.SignatureType)
87     return shim.Success(endorsementMethod)
88 }
89
90 /**
91 * This function is to be called by system chaincode (escc)
92     to identify signing method for PKI content.
93  */
94 func (s *SmartContract) getPKISignatureMethod ... {
95
96     // gen composite key (PKI_PROPERTIES)
97     ...
98
99     // read extended props
100    ...
101
102    // return endorsement method
103    endorsementMethod := []byte(pkiProps.PKISignatureType)
104    return shim.Success(endorsementMethod)
105 }
106
107
108 /*
109     Check if all endorsers have their public keys signed by all other endorsers
110 */
111 func (s *SmartContract) checkEndorsersPubKeySignatures ... {
112
113     //obtain signatures from endorsers regarding other endorsers public keys
114     ...
115     var pkiProps PKIProperties
116     pkiPropsAsBytes, err := APIstub.GetState(pkiPropsCompositeKey)
117     if err != nil {
118         return shim.Error(err.Error())
119     }
120     err = json.Unmarshal(pkiPropsAsBytes, &pkiProps)
121     if err != nil {
122         return shim.Error(err.Error())
123     }
124     ...
125
126     //check if all endorsers public keys are signed by all endorsers
127     //and return if they are or not
128     ...
129 }
```

```
130
131 /*
132     Only to be used by endorsers to sign the public keys of other endorsers
133     For multi-signatures. In threshold signatures, since there is only one
134     public key, it is assumed that all peers trust that public key
135     Args:
136     Arg0: peer id A of the owner of the public key to sign
137     Arg1: public key of A to sign
138     Arg2: peer id B of the peer signing the public key
139     Arg3: peer B signature of the public key of A (security)
140  */
141 func (s *SmartContract) signEndorserPubKey ... {
142
143     //verify args ...
144
145     //verify signature
146     ...
147
148
149     //store signature
150     ...
151
152     //if all endorsers public keys are signed by all other endorsers
153     //"activate" PKI services (functions)
154
155 }
156
157
158 /**
159 This function is used to gather signatures from the endorsers of x509 certificate
160 issued by a proxy or internal client
161 This function may also, and should, execute validations agreed
162 by the Blockchain-Enabled DPKI participants
163 */
164 func (s *SmartContract) signX509Certificate ... {
165
166     //verify args
167
168     //Check if all endorsers pub keys are signed by the other endorsers
169     // (in threshsig we assume that the public key is trusted
170     //       and accepted by all participants)
171
172     block, _ := pem.Decode([]byte(clientCertPemWithBreakLines))
173     clientCert, _ := x509.ParseCertificate(block.Bytes)
174
175     /*
176         Examples of Smart Contract validations
177     */
178     if clientCert.IsCA && !pkiProps.CanBelongToCA {
179         return shim.Error("Certificates for CA is possible with this chaincode")
```

```
180     }
181     ...
182
183
184     // E.g. test if max expiration time is accepted
185     //or if certain extensions are accepted
186     ...
187
188     // If all verifications are valid, the certificate PEM is returned so
189     the endorsers receive the output and sign it, endorsing the certificate
190     ...
191 }
192
193
194 /**
195     This function is used to gather signatures from the endorsers
196     of a client's self signed certificate.
197     This function may also, and should, execute validations agreed
198     by the Blockchain-Enabled DPKI participants
199     This is similar to the normal signX509Certificate function
200     but it can have different verifications
201 */
202 func (s *SmartContract) signClientSelfSignedX509Certificate ... {
203     //verify args
204
205     //Check if all endorsers pub keys are signed by the other endorsers
206     // (in threshsig we assume that the public key is trusted
207     // and accepted by all participants)
208     ...
209
210     //If using multisig, pki methods can only be called if all endorsers multisig
211     // pub keys are endorsed (signed by all endorsers)
212     ...
213
214     block, _ := pem.Decode([]byte(clientCertPemWithBreakLines))
215     clientCert, _ := x509.ParseCertificate(block.Bytes)
216     ...
217
218     /*
219         Check if it is in fact self signed
220      */
221     if clientCert.Issuer.String() != clientCert.Subject.String() {
222         logger.Debugf("Certificate is not self signed")
223         return shim.Error("Certificate is not self signed")
224     }
225
226
227     /*
228         Verify signature
229     */
```

```
230     ...
231
232
233     // Similar to the previous signX509Certificate function
234     ...
235 }
236
237 /**
238 This function is used to gather signatures from the endorsers of a client's CSR,
239 in case he wants to use the CSR to issue with another PKI
240 This function may also, and should, execute validations agreed
241 by the Blockchain-Enabled DPKI participants. This is similar to the normal
242 signX509Certificate function. It may have different validations
243 */
244 func (s *SmartContract) signClientCSR ... {
245     //verify args
246
247     //Check if all endorsers pub keys are signed by the other endorsers
248     // (in threshsig we assume that the public key is trusted
249     // and accepted by all participants)
250     ...
251
252     block, _ := pem.Decode([]byte(clientCSRPemWithBreakLines))
253     clientCSR, err := x509.ParseCertificateRequest(block.Bytes)
254     ...
255
256     /*
257         Verify signature
258      */
259     csrSigError := clientCSR.CheckSignature()
260     if csrSigError != nil {
261         logger.Debugf("Client's CSR signature is not valid")
262         return shim.Error("Client's CSR signature is not valid")
263     }
264
265     /*
266         Other validations
267     */
268
269         //.....
270
271     //
272
273     // Return CSR so the endorsers may sign it, endorsing it
274 }
275
276
277 /**
278     OCSP Request
279     Check certificate against the stored CRL
```

```
280
281     ocsp status: 0 - good ; 1 - revoked ; 2 - unkown
282     Response may also contain a trust level if the chaincode supports it
283
284     OCSP Response is then signed by each endorser in the ESCC component
285  */
286 func (s *SmartContract) ocspRequest ... {
287     //verify args
288
289     //certToCheckSerialNumber, err := strconv.Atoi(args[0])
290
291     //Check if all endorsers pub keys are signed by the other endorsers
292     // (in threshsig we assume that the public key is trusted
293     // and accepted by all participants)
294     ...
295
296     //Get CRL from the ledger
297     ...
298
299     //find certificate in CRL and check if it is present (revoked)
300     ...
301
302     //return OCSP status (and optionally trust level)
303     ...
304 }
305
306
307 /*
308     Get CRL
309  */
310 func (s *SmartContract) getCRL ... {
311
312     //Check if all endorsers pub keys are signed
313     //by the other endorsers (if multisig)
314     ...
315
316     // Get CRL from Blockchain's Ledger and return it
317     ...
318 }
319
320 /**
321     Revoke Certificate
322     Input args:
323         client's certificate PEM (serial number only could be used instead)
324         nonce used in signature
325         signature regarding the certificate pem or serialNumber with the client's
326             private key corresponding to the public key present in the
327             certificate to be revoked
328         time to use in the revocation (given by Proxy or
329         internal client so it is the same in all function simulations
```

```
330 */
331 func (s *SmartContract) revokeX509Certificate ... {
332
333     //Check if all endorsers pub keys are signed by the other endorsers (if multisig)
334     ...
335
336     clientCert, _ := x509.ParseCertificate(block.Bytes)
337
338     certPemAndNonce := clientCertPemWithBreakLines + requestNonce
339
340     //Verify client signature (client invoking this transaction must
341     //be the owner of the certificate to be revoked)
342     ....
343
344     // Update CRL, now with a new revoked certificate entry
345     ...
346
347
348     // Return new CRL so it is signed by the endorsers (by the ESCC component)
349     ...
350 }
351
352 /**
353 Get certificate
354 Input args:
355     subjectId of certificate to obtain
356 */
357 func (s *SmartContract) getX509CertificateBySerialNumber ... {
358
359
360     //Search certificate in the Blockchain's Ledger
361     //with the specified serial number
362     ...
363 }
364
365 /**
366 Store certificate. It is used by a Proxy or internal client to store a certificate
367 that was endorsed/signed by the endorsers, therefore being completely issued by
368 the Blockchain-Enabled DPKI. This is used because after a Proxy/Internal Client
369 gathers the endorsers signatures, it needs to send the issued certificate
370 with all the signatures back to the Blockchain to be stored in the ledger,
371 completing the issuing process.
372 Input args:
373     x509v3 certificate pem
374 */
375 func (s *SmartContract) storeX509v3Certificate ... {
376
377
378     //Verify that the endorsers in fact signed the received certificate
379     //before storing it in the ledger
```

```
380      ...
381
382 }
383
384 /**
385 Store Signed CSR
386 Store a CSR that was signed by the endorsers
387 Similar to the storeX509v3Certificate function
388 Input args:
389      csr pem
390      Map<String,String> of the endorsers signatures
391          (Format is <EndorserID,EndorserSignatureRegardingCSR>)
392 This function can verify if the CSR was in fact signed by the
393 endorsers by validating the received signatures
394 */
395 func (s *SmartContract) storeSignedCSR ... {
396
397      //It should verify that the endorsers in fact signed the
398      //received CSR before storing it in the ledger
399      ...
400
401 }
402
403 /**
404 Store Signed Self Signed Cert
405 Store a Self Signed Cert that was signed by the endorsers
406 Similar to the storeX509v3Certificate function
407 Input args:
408      self signed cert pem
409      Map<String,String> of the endorsers signatures
410          (Format is <EndorserID,EndorserSignatureRegardingSelfSignedCert>)
411 This function can verify if the CSR was in fact
412 signed by the endorsers by validating the received signatures
413 */
414 func (s *SmartContract) storeSignedSelfSignedCert ... {
415
416
417      //It should verify that the endorsers in fact signed the
418      //received CSR before storing it in the ledger
419      ...
420
421 }
422
423 //Support functions
424 ...
425
426 }
```